

124 FINAL

Note that an aid sheet will not be provided with the final exam. More obscure Java syntax details will be provided with the problem statement, if needed.

- Java Fundamentals
 - Primitive types, variable declaration and initialization, expressions, conditionals, loops. ✓
 - Type casting of primitive types and objects. ✓ Py 9
 - Screen (or "console") output using System.out.println() and System.out.printf(). ✓
 - Screen input using the Scanner class.
 - Arrays - single and multi-dimensional.
 - The ArrayList<T> class.
 - Aliasing objects.
 - Strings (String method signatures will be provided, if required).
 - StringTokenizer
 - Wrapper classes (Wrapper class method signatures will be provided, if required).
 - Enumerated types or enum's.
 - Importing packages.
 - Packages and use of static import.
 - Use of modules in Java.
 - Methods - use and declaration.
 - Passing parameters by value and by reference.
 - Object instantiation.
 - Use of instanceof, this, super, static, final, public, private, protected, extends and implements keywords.
 - Method refining, overloading and overriding.
 - File I/O (As in Exercise 5).
 - The File class (As in Exercise 5).
 - Lambda functions and method references.
- OOP in Java
 - Definition of a class, an object (instance), instance variables or attributes, static variables and static methods
 - Encapsulation or "Information Hiding"
 - Constructors.
 - Mutators, accessors.
 - Other methods - equals(), compareTo(), clone(), and toString().
 - Constructing an Object Hierarchy
 - Inheritance
 - Inner or nested classes.
 - Anonymous classes.
 - Abstract classes.
 - Interfaces. (Including changes made in Java versions 8 and 9.)
 - Polymorphism or Dynamic Binding or Late Binding.
 - Generic classes.
 - The Class<T> Object.
 - Generic methods.
 - Generic wildcards.
- Exceptions
 - Throwing.
 - Catching.
 - Writing your own Exception classes.
 - Try-with-resources structure (As in Exercise 5).
- GUI Design in Java
 - Using JavaFX to build a simple window:
 - All Panes except for TextFlow, StackPane, DialogPane and PopupControl.CSSBridge - how they work and what layouts they provide.
 - TextField, TextArea, Button, ChoiceBox, CheckBox, RadioButton and Label nodes.
 - How to attach a listener/event combination to a node, including the use of a Lambda function. Responding to a Mouse Click or a Change Event.

- A general understanding of the various files used to build a JavaFX GUI and what they are used for: *.java files including the controller file and Main.java. Also *.css and *.fxml files.
- Interpret the essence of a GUI structure from either java construction code or fxml code.
- Other Topics
 - Numeric representation.
 - Source and Effects of Roundoff Error.
 - Summations and Increasing Their Accuracy.
 - Objects in General.
 - History of Java.
 - JUnit testing.
- **Not** on final. Several of these topics were discussed in previous offerings of this course, but not in the current offering.
 - Use of whitespace and documentation in code.
 - Use of Eclipse, WindowBuilder Pro or SceneBuilder.
 - Javadoc tool.
 - Functional decomposition.
 - Class diagrams.
 - Debugging techniques and the use of the debugger in Eclipse.
 - Writing clean code.
 - Built-In JavaFX Dialogs.
 - JavaFX Nodes: ToggleButton, Slider, Spinner, Canvas and Progress Bars.
 - GUI "Look and Feels".
 - Building JFrame based windows using Swing and AWT.
 - CardLayout layout manager.
 - JOptionPane, JFileChooser, JColorChooser classes.
 - Swing Controls: JTextArea, JComboBox, JSpinner and JSlider Components
 - ImageIO
 - Graphics2D
 - Swing or JavaFX Animation
 - AffineTransform
 - Applets
 - BufferedImageOP
 - Java Web Start
 - Multi-threading using Timer and Thread classes.
 - The fork/join framework used to thread multi-core processes.

Old CISC124 exams show quite a shift in topics! And, many of the older exams were open-book, where ours is not. Basically, ignore anything to do with C++, UML diagrams and data structures. And, older exams are focussed on GUI construction using JFrame and Swing, where this offering of the course is using JavaFX.

- [December 2017 Final](#) and [Solution](#)
- [December 2016 Final](#) and [Solution](#)
- [December 2015 Final](#) and [Solution](#)
- [December 2014 Final](#) and [Solution](#)
- [December 2013 Final](#) and [Solution](#)
- [December 2012 Final](#) and [Solution](#)
- [April 2012 Final](#) and [Solution](#)
- [April 2011 Final](#) and [Solution](#)
- [April 2010 Final](#) and [Solution](#)
- [April 2009 Final](#) and [Solution](#)
- Dec 2007, Q1, Q4
- Dec 2005, Q2 and Q4
- [Dec 2004, Q1 to Q3, Q6](#)
- Dec 2003, Q1, Q2 and Q3 (note the comprehensive aid sheets supplied with this exam!)
- Apr 2004, Q5
- Dec 2002, Q2 and Q5
- Dec 2001, Q1 to Q6, Q9 to Q11
- Don't use any older exams

Here are some old CISC212 exams that might be useful:

- The [2007 final exam](#) and its [solution](#). Q3 to Q6
 - The [2006 final exam](#) and its [solution](#). Q3 and Q5
 - The [2005 final exam](#) and its [solution](#). Entire exam is good.
 - The [2004 final exam](#) and its [solution](#). Q2 to Q4
-
- Work through the old exams first, without looking at the solutions. Time yourself.
 - Go over your quizzes and their solutions. Make sure you understand what you did wrong.
 - Go through the exercises. Examine assignment solutions, especially if you had problems with the assignment.
 - Make sure you can **write code!** If you can't write Java code by now, you will have problems on the final exam.
 - Be prepared to read code, write code, answer short answer questions and true/false questions.
 - Ask questions in an onQ forum while you are studying.
 - Any questions Emailed to the prof. may still end up (anonymously) in the forum.
 - Get a good sleep the night before!

[Home](#)

Last modified: 12/01/2018 13:09:44

"quiz 1 prep part 1"

Notes for introduction

History of Java

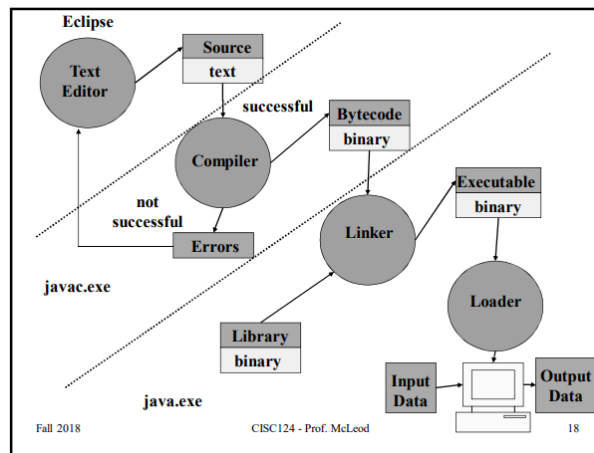
- The language was first developed by **James Gosling** at **Sun Microsystems** in **1991**
 - He was designing a language, called "**Oak**", for the "**Green Project**"
 - – The Green Project envisaged the centralized control of many processor-based devices in the home
 - "**Oak**" was designed to be a robust, efficient language with maximum portability to different processors
 - The Green Project flopped
- In the early 90's, Needed a **robust, compact, multiplatform** language, so let's dust off Oak and call it something racy like "Java"
- In **1994**, **Sun Microsystem** demonstrated the use of Java in small bundles of code embedded in a web page - called **applets**
- Netscape browsers started supporting applets in **1995**, starting Java's rise to fame
- Early in **2010 Oracle** acquired **Sun Microsystems**.

How Java Works?

- A **compiler** (part of the "**JDK**", or "**Java Development Kit**" – sometimes called **javac.exe**) which is designed to run on your development platform, compiles your **source code (*.java file)** to a **byte code file (*.class file)**.
 - The **byte code file** is platform-independent, and is the thing you would attach to your web page as an applet
 - Two components of the **JDK** are the programs "**javac.exe**" and "**java.exe**".
 - **javac.exe** is the **byte code compiler**, and **java.exe** is the **JRE** which **executes the byte code file**.
 - The java.exe program:
 - accepts the byte code file
 - links in any required libraries
 - creates executable code in memory
 - converts it to machine language
 - sends it to the CPU.
 - "**Compilation**" is the process of converting the ***.java file** to a ***.class file (the byte code file)**. This is done by calling **javac.exe** in the background, and supplying that program with all the required command line parameters.
 - Every browser written for every platform and OS, can have an embedded code processor called a **JVM**, or "**Java Virtual Machine**", built-in.
 - The **JVM** takes the **byte code** and executes it by generating the machine code that will be

recognized by the platform that is running the browser

- **JVM** could run stand-alone Java applications. This is the **JRE** or “**Java Runtime Engine**” (**java.exe**).
- So, Java can be used either to create **applets** for use in web pages or for **stand-alone applications**.
- However applets have fallen out of favour in the last few years due to security concerns



Class Structure

- A **class** or “**object definition**” or an “**object**”, consists of **instance variables and/or methods**
- By convention, instance variables are all declared **before the methods**

```
1 public class ShowStructure {  
2     // instance variables or “attributes” here  
3     // methods here  
4 } // end class ShowStructure
```

- In Java, a **class** is an **Object**, and an **Object** is a **class**
- **Code and attributes cannot be defined outside of a class**
- The only code that can exist outside a method are attributes or other (“inner”) class definitions
- Attributes

```
1 [private|public] [static] [final] type attributeName [= literalValue];  
2 //这里有[]的都代表的是optional的
```

- Also called “**class variables**” or “**instance variables**” or “**fields**”
- Declared within a class at the same level as the method declarations, these variables are known to all methods in the same class
- You can control their privacy and the way they are stored in memory (using **public/private/protected and static**).
 - **public** means the attribute or method is available **to any external class (as well as inside the class)**
 - **private** means that the attribute or method, the “member”, **is only available inside the class in which it is declared.**
 - **protected** means the member is only public to **classes in the same package** as the class

in which the member is declared

- static: static means different things depending on where it is used.
 - public static members are available outside the class **without the need to instantiate the class**
 - Any static member remains in memory until the program is complete
 - Since main is static, it can **only invoke other static methods when they are in the same class**
- type part is not optional – this is why java is a **declarative language**, And, a variable cannot change its type later, called **static typing**, cannot use a variable unless you have declared it first.
- Variable Declaration
 - Declaring a variable inside a method gives that variable the scope of **just inside the method, not outside the method**

```
1 variabletype variablename = value;
```

- Method Declaration

```
1 [private|public] [static] [final] returnType methodName ([parameterList]) {...}
```

- The **returnType** can be any single Object or a primitive type, 如果什么都不想return的话，那就在 returnType那里写上void

```
1 public static void main (String[] args) {...} //declaration for main
```

- starting point of the whole program is always the execution of the main method
- Each **parameter type must be declared in the parameter list**, as in type parameterName, type parameterName, ...
- Unless the return type is void, the method must contain at least one return statement在method里面的, The type of “literal|expression” must **match** the return type specified in the method declaration statement.

Primitive Types

- Everything else in Java is an Object, A variable declared as one of the types shown below is not an Object

```
char
byte
short
int
long
float
double
boolean
```

- Integer Primitive Types
 - **byte, short, int, long**

- For byte, from -128 to 127, inclusive (1 byte).
- For short, from -32768 to 32767, inclusive (2 bytes).
- For int, from -2147483648 to 2147483647, inclusive (4 bytes).
- For long, from -9223372036854775808 to 9223372036854775807, inclusive (8 bytes).
- A “**byte**” is **8 bits**, where a “**bit**” is **either 1 or 0**.
- Real Primitive Types/“Floating Point” Types
 - **float, double**
 - For float, (4 bytes) roughly $\pm 1.4 \times 10^{-38}$ to $\pm 3.4 \times 10^{38}$ to 7 significant digits
 - For double, (8 bytes) roughly $\pm 4.9 \times 10^{-308}$ to $\pm 1.7 \times 10^{308}$ to 15 significant digits
- Character Primitive Type
 - **char**
 - From '\u0000' to '\uffff' inclusive, that is, from 0 to 65535 (base 10) or 0 to ffff (base 16, or “hexadecimal”). A variable of the char type represents a Unicode character. Can also be represented as 'a' or '8', etc.
- Boolean Primitive Type: **true/false**
- String Objects
 - **String’s are not primitive data types, but are Objects**

```
1 String example = "I am a string!";
2 //or
3 String example = new String("I am a string!"); //This is called instantiation
```

- Array Declaration 

```
1 int[] data = new int[10]; // 这里面这个10指的是你declare的这个array的size
2 //or
3 int[] data;
4 data = new int[10];
```

- **new** is always involved with the **instantiation** of an Object.
- Arrays are **Objects** in Java
- Java array can only hold items of **all the same type** (even if they are objects)
- The size of a Java array **must be declared and is then fixed**
- In Java you can only get at a single element at a time

```
1 int[] anArray = {1, 2, 3, 4}; // Array literal!
2 anArray[2] = 10; // Changes third element to 10
```

- To get the size of an array, use the **.length attribute**
- For example anArray.length would provide 4 for the array on the previous code
- In Java, you **cannot** use **pointers** to access array elements, only the indices
- The **first** array element is at index **zero**
- 只要输进去了正数，然后没有declare是什么type的话，Java都会把他们存为int

- 在数字后面加一个L就可以换成long
 - ex: 43 >>> 43L
- Binary, Octal and Hex Literals
 - Use the prefix **0b (or 0B)** in front of the numbers to get a binary literal
 - Octal – just use **0**
 - Hex use **0x**
- 只要输进去了一个小数，那么Java就会把这个小数看作是一个double，在数字后面加F可以存为float
- Java names may contain any number of letters, numbers and underscore (“_”) characters, but they must begin with a letter
- var
 - var is not a Java keyword instead it is called a “reserved type name”

```
1 var aNum = 10;
```

- aNum will be of type int
- The type for the variable is **inferred from the type of the literal value used to initialize the variable**
- var can reduce repetition in longer declarations
- Constant Attribute Declaration

```
1 [private|public] [static] final type ATTRIBUTE_NAME = literal_value; //constant的名字一定要大写
```

- The Java keyword, **final** can be used to **make sure a variable value is no longer “variable”**.
- Usually these are declared public static
- Java will not allow your program to change a constant’s value once it has been declared

Type Casting *

- When a value of one type is stored into a variable of another type
- A value to the left can be assigned to a variable to the right without explicit casting

byte > short > int > long > float > double

- 因为越左边的占的空间就越小，所以从一个小的占空间到一个大的占空间不需要理由
- 如果相反方向：`int anotherVar = 345.892;`
- the compiler would protest loudly because a double cannot be stored in an int variable **without loss of precision**. Wrong direction!
- 如果真的想从反方向转的话，就要做好准备这个数字会loss precision了
 - `int anotherVar = (int)345.892;`
 - The variable anotherVar would hold the value 345

Arithmetic Operation

Binary arithmetic operators



- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)
- Modulus (%) ($12 \% 5 = 12 \div 5$ 的余数 = 2)

之所以叫做 Binary arithmetic operators 是因为在 operator 的两边都需要数字才可以使式子成立

* 当把 integer 放在两边时, 产生的结果肯定也是一个 integer

* 当式子的两边是 mixed types 的时候, java 会把一个 type cast 到另一个 type, 并前结果会是剩下的 type, 所以, 如果是 $\text{int } 3 + \text{double } 3.5$ 的话, java 会先把 3 cast 成 double 3, 然后最终的结果肯定是一个 double

* 把 + 用在 String 中间, 所以把 String 连成一个 String

* 计算的顺序是从左到右

Arithmetic Operators

- "+" → Positive
- "-" → Negative
- ++ → increment, increase the number by 1
- → decrement, decrease the number by 1

Pre-increment: ++i, they increment the variable before it's used in the expression

Post-increment: i++, after

Pre-decrement: --i, they decrement before

Post-decrement: i--, after

Assignment Operators {

- = 赋值
- * = multiply and set equal to
- / = divide and set equal to
- = subtract and set equal to
- + = add and set equal to

Logical Binary Operators {

- ==
- !=
- >
- <
- >=
- <=
- & and, always evaluates both sides of the expression
- && and, stops when the left side is false
- | or, always evaluates both side of the expression
- || or, stops when the left side is true

Logical operator : !, not

Precedence rules :

- Unary operators , casting
- *, /, %
- +, -
- <, >, <=, >=
- ==, !=
- &, &&, |, ||
- =, *=, +=, -=, /=



* Expression are combinations of variable, literal values, operators, keywords, method calls, etc.

* Keywords

Method invocations { Naming the method
Providing arguments or not
Doing something with the return value or not

Method Invocations – 2. Providing Arguments for the Parameters

- If the method has been declared to accept arguments, *i.e.* it has a non-empty parameter list, then you must supply a matching list of arguments.
- If the method does not have any parameters, then you must still use empty brackets, (), when you invoke the method.

Fall 2018

CISC124 - Prof. McLeod

7

Method Invocations – 3. Using a Return Value

- A non-void method will return something.
- You can use that “something” in an expression, or just store it in a variable – your choice.
- The method has declared the type of that “something”.
- If the method was declared as void, you will not get a return value and you can only invoke the method by itself, not as part of an expression or an assignment statement.

Fall 2018

CISC124 - Prof. McLeod

8

Method Invocations - Examples

- See the MethodInvocation.java program.

Fall 2018

CISC124 - Prof. McLeod

9

Java Punctuation - Whitespace

- Multiple spaces are treated as one space.
- Leading and trailing spaces are ignored.
- Tabs and empty lines (line feeds) are ignored.
- Line continuation:
 - Long lines can be continued on the line below – break the line anywhere there is a space, but not in the middle of a string (!).
 - For good style, indent the line to show it is a continuation.

Fall 2018

CISC124 - Prof. McLeod

10

Java Punctuation, Cont.

- Comments: inline `//` , block `/* */`
- Comma `,` used in parameter lists and array literals.
- Semi-colon `;` used to end a statement and with for loop syntax.
- Colon `:` used with switch statements and “for each” loop.
- Period or “dot operator” `.` used with objects to obtain members.
- Also `[]` , `()` and `{ }`
- And `->` along with `::`

Fall 2018

CISC124 - Prof. McLeod

11

Conditionals or “Selection Statements”

- We will consider `if`, `if-else` and `switch` statements.
- Simple `if` statement syntax: *simple if statement*

```
if (boolean_expression)
    statement_when_true;
```

- Example:

```
if (capacitance < 0)
    System.out.println("Illegal capacitance");
```

Fall 2018

CISC124 - Prof. McLeod

12

if-else Statement

- Syntax of **if-else** statement:

```
if (boolean_expression)
    statement_when_true;
else
    statement_when_false;
```

- Example:

```
if (stress > maxStress / 1.5)
    result = "failure";
else
    result = "pass";
```

Fall 2018

CISC124 - Prof. McLeod

13

if-else Statement, Cont.

- With statement blocks:

```
if (boolean_expression) {
    block_of_code_when_true
}
else {
    block_of_code_when_false
}
```

- Note indentation and bracket alignment for style.*

Fall 2018

CISC124 - Prof. McLeod

14

Abbreviated if Statement

- Uses the "ternary operator" - ?
expression1 ? expression2 : expression3

- expression1 must evaluate to a boolean
- expression2 (when true) and expression3 (when false) must evaluate to the same type.
- You could use this with an assignment, for example:

```
int smaller = a <= b ? a : b;
```

- stores the smaller of the two numbers in `smaller`.

Fall 2018

CISC124 - Prof. McLeod

15

"Chained" if Statements

- Syntax:

```
if (condition1) { block1 }
else if (condition2) { block2 }
else if (condition3) { block3 }
else if (condition4) { block4 }
...
else { blockn }
```

- Each *condition* is tested in turn, until one is evaluated to **true**. If none of them are **true** then the **else** block is executed.

Fall 2018

CISC124 - Prof. McLeod

16

Dangling else Problem

- It is not unusual to nest if statements inside each other.
- One issue that can arise is the "Dangling else" problem.
- See DanglingElse.java
- Indentation might give a visual clue, but has no syntactic meaning.
- The else should be associated with the closest if.
- Use { } if necessary.

Fall 2018

CISC124 - Prof. McLeod

17

switch Statement

- Syntax:

```
switch (expression) {
    case val1:
        // statements if expression produces val1
        break;
    case val2:
        // statements if expression produces val2
        break;
    case val3:
        ...
    default:
        // statements if none of the above is true
        break;
} // end switch
```

Fall 2018

CISC124 - Prof. McLeod

18

switch Statement - Cont.

- The code to be run depends on which **val#** value matches **expression**.
- If none match, the statements after the **default:** clause are run.
- The **expression** and **val#** values (or “Case Labels”) must all be of the same integer (including char) or String type.
- The **break;** statements make sure that following cases are not executed after a match has been made.
- It is possible to do multiple cases on one line, but it is clumsy:

Fall 2018

CISC124 - Prof. McLeod

19

switch Statement - Cont.

```
switch (expression) {
    case val1: case val2: case val3:
        // statements if expression is val1, val2 or val3
        break;
    case val4: case val5:
        // statements if expression is val4 or val5
        break;
    case val6:
        ...
    default:
        // statements if none of the above is true
        break;
} // end switch
```

Fall 2018

CISC124 - Prof. McLeod

20

switch Statement - Cont.

- Not too useful a construct.
- Menu coding is a possible use:
 - Provide a number of options to the user, like “(A)dd, (E)dit or (D)elete”.
 - The user presses a, e, d, A, E, D, or some other key.
 - In a switch statement, you would have:

Fall 2018

CISC124 - Prof. McLeod

21

switch Statement - Cont.

```
switch (userResponse) { // userResponse is a char
    case 'a': case 'A':
        // Add operation code
        break;
    case 'e': case 'E':
        // Edit operation code
        break;
    case 'd': case 'D':
        // Delete operation code
        break;
    default:
        // Tell user wrong key pressed
        break;
} // end switch
```

Fall 2018

CISC124 - Prof. McLeod

22

switch Statement in Java 7+

- Can now use **Strings**.
- See Switch.java.
- Comparisons are case sensitive. As if **.equals()** is being used.
- Generates more efficient bytecode than what you would get from a chained if construct.

Fall 2018

CISC124 - Prof. McLeod

23

Repetition or Using “Loops”

- Java has:
 - **while**
 - **do/while**
 - **for**
 - The “for each” loop
- Will discuss the use of **break** and **continue**

Fall 2018

CISC124 - Prof. McLeod

24

“while” loop - Cont.

- **while** loop syntax:

```
while ( boolean_expression ) {
    block_of_code
}
```

- As long as *boolean_expression* evaluates to *true* the statements in the *block_of_code* continue to execute.
- One statement inside the loop does not need { }.
- By mistake, you might write the following - what would happen?

```
while ( boolean_expression );
    line_of_code
```

Fall 2018

CISC124 - Prof. McLeod

25

“do/while” loop

- Syntax:

```
do {
    block_of_code
} while ( boolean_expression );
```

- Note the “;” at the end of the while statement.
- Since the conditional test is at the end of the loop, it will always execute the loop at least once.

Fall 2018

CISC124 - Prof. McLeod

26

“for” loop

- The kind of **while** loop shown below:

```
int i = 1;
while ( i < 21 ) {
    // other statements
    i = i + 1;
}
```

is used so often, that Java has provided another looping structure that does all that is shown above, but needs only one line:

```
for (int i = 1; i < 21; i = i + 1) {
    // other statements
}
```

OR i++

Fall 2018

CISC124 - Prof. McLeod

27

“for” loop - Cont.

- Syntax:

```
for (initialization; boolean_expression; update) {
    block_of_code
}
```

- **for** loops are used when you know, in advance, the number of repetitions desired.
- If there is only one statement inside the loop you don't need the { } brackets.

Fall 2018

CISC124 - Prof. McLeod

28

“for” loop - Cont.

- You don't have to declare the counter inside the **for** loop, if you have declared it earlier in your program.
- But if you do declare it in the **for** statement then the scope of that variable will only be inside the loop block.

Fall 2018

CISC124 - Prof. McLeod

29

“for each” Loop

- Often, you will want to visit every element in a collection, not just a part.
- Syntax of the “for each” loop:

```
for (type variable : collection) {
    // statements
}
```

Fall 2018

CISC124 - Prof. McLeod

30

“for each” Loop, Cont.

- For example, suppose we have an array called `data`, containing a collection of `double` type numbers, and you want to add them all up:

```
double sum = 0;
for (double e : data)
    sum = sum + e;
```

Or `sum += e;`

- `var` can be used to type the element variable in a for each loop.

Fall 2018

CISC124 - Prof. McLeod

31

“for each” Loop, Cont.

- Equivalent normal `for` loop:

```
double sum = 0;
for (int i = 0; i < data.length; i++)
    sum = sum + data[i];
```

- The “for each” loop is a bit easier with arrays, but is even better suited for other kinds of collections.

Fall 2018

CISC124 - Prof. McLeod

32

Loops - Misc.

- Don't declare variables inside loops, as the repeated declaration process uses up time and memory unnecessarily.
- There is no limit in Java to how many levels you can nest loops.
- It is customary, but not necessary, to use the variables `i`, `j`, `k` as loop counters when the counter has no intrinsic meaning.

Fall 2018

CISC124 - Prof. McLeod

33

Notes for Numeric Representation

2018.9.19

- “for each” loops
- Multi-Dimensions Arrays

“for each” loops

- When you want to visit every element in a collection
- Below is the “for each” syntax and an example
- Easier for array

```
1 for (type variable : collection) {
2     // statements
3 }
4
5 //suppose we have an array called data, containing a collection of double type numbers, and
  you want to add them all up.
6
7 double sum = 0;
8 for (double e : data) // var can be used
9     sum = sum + e; // sum += e
```

- Customary but not necessary, to use variable i, j, k to use as loop counter, 最外面的一层用i，然后是j，然后是k

Continue and Break

- Continue是用来停止loop中的某一行，并且重新开始loop
- Break是用来结束整个loop的

```
1 for (i = 1; i <= 5; i++) {
2     if ( i == 3 ) // 在i=3时，loop结束了并直接开始当i=4
3         continue;
4     System.out.println("i = " + i);
5 }
6 System.out.println("End of Loop!");
7
8 for (i = 1; i <= 5; i++) {
9     if ( i == 3 ) // 在i=3时，loop结束了并直接开始“End of Loop!”
10        break;
11    System.out.println("i = " + i);
```



```

12 }
13 System.out.println("End of Loop!");

```

- 不要过于依赖break/continue，要用更加好的办法

Multi-Dimensional Arrays*

- An array containing one or more arrays
- 把2D Array想象成一张表格, you can think of the first dimension as the rows(行), and the second dimension as the columns(列)
- You can use three sets of [] to get a 3 dimensional array. Using the spreadsheet analogy, the third dimension could be the sheet number, where each sheet contains a table

```

1 int[][] twoD = new int[4][20]; //This array has room for 80 values(4*20).
2 int row, col;
3 for (row = 0; row < twoD.length; row++)
4     for (col = 0; col < twoD[row].length; col++)
5         twoD[row][col] = row * col;

```

}

```

1 // Try using a for/each loop to generate an array
2 public static int[][] generateArrayForEach(int numRows, int numCols) {
3     // Try replacing array type with var:
4     int[][] anArray = new int[numRows][numCols];
5     int counter = 0;
6     // Try replacing types with var in loops:
7     for (int[] aRow : anArray)
8         for (int aVal : aRow) {
9             aVal = 10 * counter;
10            counter++;
11        }
12     return anArray;
13 } // end generateArray method

```

*Style will be on another page

*以下是一个good style的范例

```

1 /*
2  * A program to demonstrate good style and documentation.
3  * The program prompts the user for a series of numbers and then prints out the
4  * average of the numbers to the console window. The entry process stops when
5  * the user enters a negative number.
6  *
7  * for CISC124, by Alan McLeod, version 2.3, 20 Sept. 2018
8  */
9 // Note that even better style would use Javadoc comments – but we don't know how to do
10 // this yet...
11 public class GoodStyle {
12
13     // This method displays the instructions for the user.
14     public static void showInstructions() {

```

```

15         String instructions = "This program provides the average of the integer numbers you
enter.\n" +
16                                     "You must enter at least one number.\n" +
17                                     "Enter a negative number to quit. This negative number is not
included " +
18                                     "in the calculation.\n";
19         System.out.println(instructions);
20     } // end showInstructions method
21
22     // This method obtains integer values from the user and returns their average as a
double.
23     // If the user does not supply any numbers – just a negative number, then the method
returns NaN.
24     public static double getAverage() {
25         int sum      = 0;    // Holds the sum of the numbers supplied.
26         int aNum     = 0;    // A number provided by the user.
27         int numNums  = 0;    // The number of numbers provided by the user.
28         // Loop until the user provides a negative number
29         while (aNum >= 0) {
30             // Use the IOHelper class to get an int value from the user
31             aNum = IOHelper.getInt("Enter number " + (numNums + 1) + ": ");
32             // If the number is >= zero, add it to the sum and count it
33             if (aNum >= 0) {
34                 sum = sum + aNum;
35                 numNums++;
36             } // end if
37         } // end while
38         // Return the average
39         return (double)sum / numNums;
40     } // end getAverage method
41
42     public static void main(String[] args) {
43         // Show the instructions
44         showInstructions();
45         // Display the average
46         System.out.printf("\nThe average is: %.2f", getAverage());
47         // Sincere and unnecessary program completion message
48         System.out.println("\n\nAll done!");
49     } // end main method
50
51 } // end GoodStyle class

```

2018.9.23

Design A Method

- Advantage for modularity(模块性):
 - Easier to built
 - Easier to build
 - Easier to test

- Easier to debug
- Easier to modify
- Easier to share
- Methods are written to avoid repeating code, make sure one thing for one method and make sure it do it well
- Methods should be short
 - If it can satisfy all the other rules and still explain itself, it's short enough
- Keep all code within the method at the same level of abstraction
- 整个程序应该是一个Top-down narrative的结构, The most abstract methods will be at the top of the program, leading to the least abstract methods further down
- 尽可能的少用parameters，最多只能用三个，没有parameter是最好的
- 使用object/list去把多个相同类型的parameter圈起来再放进去用比直接把这些parameter全部一起用要好

```

1 drawCircle(Point centrePoint, int radius)
2
3 drawCircle(int centreX, int centreY, int radius)
4
5 \\上面这个要比下面这个好，因为他把同类的东西都划在了一起

```

- A method should either do something or answer something, or both.

Numeric Representation

- 二进制 >>> 十进制
 - 把二进制的数字拿出来，从左数第一个开始，(左数第一个数字) * $2^{(小数点后数字个数-1)}$ + (左数第二个数字) * $2^{(小数点后数字个数)}$ 一直加到小数点左边第一个数字，和这个数字相乘的2的power应该是0，小数点右边的，就(小数点右边第一个数字) * $2^{(-1)}$ + (小数点右边第一个数字) * $2^{(-2)}$
- 十六进制 >>> 十进制
 - 十六进制(Hexadecimal Numbers) : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A(10), B(11), C(12), D(13), E(14), F(15)
 - 十六进制转十进制的话，举个例子
 - $B65F >>> 11*16^3 + 6*16^2 + 5*16^1 + 15*16^0 >>> 46687$
- 十进制 >>> 二进制 (在附上的课件上有)
- 十六进制 >>> 二进制
 - 举个例子: 2C6B.F06
 - 这个时候，把这些单个数字分别看成十进制的数
 - 2(2) C(12) 6(6) B(11).F(15) 0(0) 6(6)
 - 再分别把这些十进制的数字化为二进制的
 - (0010) (1100) (0110) (1011). (1111) (0000) (0110)
- 二进制 >>> 十六进制
 - Ex: 10110011101.11011
 - 小数点左边的数字，从右边开始数，四个一组，写下来，到最后不够四个的时候，就在前面补零，小数点右边的数字，从左边开始数，四个一组，写下来，到最后不够四个的时候，就在后面补零
 - 0101 1001 1101. 1101 1000
 - 再把他们化成十进制，然后直接写成十进制就好了
 - 5 9 D. D 8

- Roundoff Error
 - Only sum of power of 2 can be stored properly
 - (0.1)十进制 = (0.0 0011 0011 0011 0011 0011...)二进制
 - 所以电脑不可能把二进制延伸下去的东西全部存完，肯定会截取掉一部分，而被截取掉的，就叫做Roundoff Error

Storage of Integers

- An “un-signed” 8 digit (one byte) **binary number** can range from 00000000(0 in base 10) to 11111111(255 in base 10)
- Two’s Complement
 - Make the most significant bit(最左边的那个) a negative number
 - 例子: 最小的“signed”的数字是10000000, 换算成十进制是 $(-1) \times 2^7 \gg -128$ in base 10
 - 下表体现了一些基础的Two’s Complement的二进制与十进制对应的数字，

Two’s Complement System for 1 byte:	
binary	base 10
10000000	-128
10000001	-127
11111111	-1
00000000	0
00000001	1
01111111	127

- 如果是在Two’s Complement里面，那么他的计算是这样的
 - 举个例子：10010101
 - 这时候，只需把最左边的数字，即1，写成-1，然后其他的计算方式都是照旧的
 - 那么这个数字在十进制里面是-107
- 所以这时候就知道, byte 这个integer type为什么是range from -128-127
- 如果我们处于Two’s Complement的规则里面，想要在最大的byte number，即01111111的基础上再加1
 - 如果是在普通的加法运算的话，这个加上1会变成10000000，即在十进制中是128
 - 但是在Two’s Complement里面，这个加上1之后会变成10000000, 这时候会变成负数，即-128 in base 10
 - So integer numbers wrap around, in the case of overflow. No warning from Java!
- An int is stored in 4 bytes using Two’s Complement
- An int range from:
 - 10000000 00000000 00000000 00000000 to
 - 01111111 11111111 11111111 11111111
 - 即在十进制里面是 -2147483648 to 2147483647
- 假设你写一个factorial的程序：
 - $n! = n * (n-1) * (n-2) * \dots * 2 * 1$
 - 然后分别把结果存成int, long

```

1 public static int intFactorial(int n) {
2     int f = 1;

```

```

3      if (n > 0)
4          for (int i = 2; i <= n; i++)
5              f *= i;
6      return f;
7  } // end intFactorial method

```

```

1 public static long longFactorial(int n) {
2     long f = 1;
3     if (n > 0) {
4         for (int i = 2; i <= n; i++) {
5             f *= i;
6         }
7     }
8     return f;
9 } // end longFactorial method

```



- 当你把结果存成int的时候，在12!和13!之间，就会出错，因为这时候结果已经超出了int这个格式的范围
- 当你把结果存成长的时候，在21!的时候，结果会变成负数，在Two's Complement的条件下，当超出范围，是会Wrap Around变成负数的
- 这时候，我们可以使用一个叫做 BigInteger class的Class

```

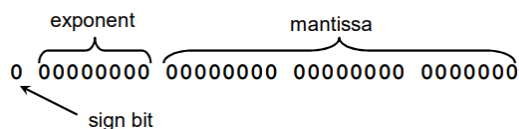
1 public static BigInteger bigIFactorial(int n) {
2     BigInteger f = new BigInteger("1");
3     if (n > 0) {
4         for (int i = 2; i <= n; i++) {
5             f = f.multiply(new BigInteger(Integer.toString(i)));
6         }
7     }
8     return f;
9 } // end bigIFactorial

```

- 使用了BigInteger之后数字会变的精准了很多，但是因为使用它会消耗过多的内存已经更加长的运行时间，所以不会把BigInteger当成第一选择

Storage of Real Numbers

- Real Numbers(实数)可以直观的看成是小数，他们可以把数轴填满
- 在Java里面，用来存实数的体系叫做IEEE standard number 754
- Like an float, is stored in 4 bytes or 32 bits
 - These bits consist of 23 bits for the mantissa, 8 bits for the exponent, 1 bit for the sign



- The sign bit, s, can be 0 for positive and 1 for negative.

- The exponent, e , is unsigned. The standard says the exponent is biased by 127 and that the values 0 and 255 are reserved. So the exponent can range from -126 to +127. E is the unbiased value and e is the biased value ($E = e - 127$).
- Mantissa指的是在小数点右边的全部数字，always less than 1.

[illegible]

- 这个根据计算，会看出来他的值是0.011, 然后在十进制里面是0.375
- Maximum float最大的float值出现在当
 - $s = 0 \ggg \text{positive}$
 - $e = 254 \ggg E = e - 127 = 127$
 - 数字是3.4028235E38
 - Float.MAX_VALUE
- Minimum normal float最小的normal float值出现在当
 - $e = 1 \ggg E = 1 - 127 = -126$
 - $f(\text{mantissa}) = 000000000000000000000000$
 - 数字是1.17549435E-38
 - Float.MIN_NORMAL
- These are all “normalized” numbers because we are not using the reserved exponent values, 0 and 255
- A “denormalized” (or “subnormal”) number has $e = 0$

$$\text{value} = (-1)^s \times 0.f \times 2^{-126}$$

- 有了denormalized数字，float的范围又可以延长了(to 1.4×10^{-45} , Float.MIN_VALUE)
 - $e = 0, f = 0 \gg -0/+0$ (depending on sign bit)
 - $e = 255, f = 0 \gg -\text{Infinity}/+\text{Infinity}$ (depending on sign bit) displayed as -inf/+inf
 - $e = 255, f \neq 0 \gg \text{NaN}$ (Not a Number)
- Double, (8 bytes) roughly $\pm 4.9 \times 10^{-308}$ to 15 significant digits(同样也是使用IEEE754)
 - 8 bytes 包含了52位的mantissa, 11位的exponent, 还有一位的sign bit

Invalid Operation	Provide NaN
Division by Zero	Provide +Infinity or -Infinity
Overflow	Provide the largest possible normalized value or ±Infinity
Underflow	Provide smallest possible normalized number, a denormalized number or ±0
Inexact Value	Provide the nearest rounded value

>>>>>>>>>>

- Standard同时还建议我们最好设立一些flag(就像expectation)去防止此类事情的发生
- Numbers like 0.1 that can be written exactly in base 10, but cannot be stored exactly in base 2.
- Real numbers (like pi or e) that have an infinite number of digits in their “real” representation can only be stored in a finite number of digits in memory
- Windows Calculator:
 - Can provide 32 accurate digit
 - It is the CPU that may have built-in support for 10 or 12 byte numbers
 - Calculator also stores rational numbers as fractions to retain accuracy. For example, 1/3 is stored as 1/3, rather than .333
- strictfp
 - This is a Java modifier that can be used in either class or method headers
 - 在计算过程中，他会把所有计算过程中出现的数字都用除了IEEE754以外的更加精准的方法去储存，只有到了结果的时候，才会换回IEEE754的方法去储存

```
1 float sum = 0;
2 for (int i = 0; i < 10000; i++)
3     sum += 0.1f;
4 System.out.println(sum)
```

像上面这个代码，是为了计算10000个0.1的和，对于人类来说，我们很容易计算出来这个的结果是1000，但是对于电脑来说，0.1不能被完整储存，所以每次计算都会损失一些精确值

Roundoff Error

- For the code above, it prints a result of 999.9029 to the screen
- If sum is declared to be a double then the value: 1000.0000000001588 is printed to the screen.
- the individual roundoff errors have piled up to contribute to a **cumulative error** in this calculation
- the **roundoff error** is smaller for a double than for a float.

Big Decimal Class 可以把精确度提高的非常厉害，但是在使用BigDecimal的时候，efficiency会降低

Machine Epsilon

- ϵ is the largest positive value that when added to 1 still produces 1, ie: $1 + \epsilon == 1$
- “Too small to make a change”
- For float: 5.9604645E-8
- For double: 1.102230246251565E-16
- Observations
 - If you are subtracting two numbers and the difference between the numbers is less than ϵ , then you will get zero
 - If you are adding two numbers, n_1 and n_2 , and $n_2/n_1 < \epsilon$, then you will get n_1 .

CISC124

- Assignment 1 due this Friday, 7pm.
- Quiz 1 Next Week. Topics and format of quiz in last lecture's notes.
 - The quiz will cover the material discussed up to the end of today's lecture. *(Even if I don't finish what is in the slides.)*

Fall 2018

CISC124 - Prof. McLeod

1

Today

- Finish Numeric Representation:
 - Roundoff Error Demo – Summing 0.1
 - Machine Epsilon.
 - Effects of Roundoff Error.
 - Kahan Summation Algorithm.
 - Alternating Sign Summations.

Fall 2018

CISC124 - Prof. McLeod

2

Roundoff Error – Cont.

- Compute: $\sum_{i=1}^{10000} 0.1$
- And, compare to 1000.

```
float sum = 0;
for (int i = 0; i < 10000; i++)
    sum += 0.1f;
System.out.println(sum);
```

Fall 2018

CISC124 - Prof. McLeod

3

Aside - BigDecimal Class

- For when you need a floating point numeric type that does not have any limit to the number of digits you can store (*how much RAM you have!*).
- However, it is usually a good idea to specify the scale of the numbers you wish to use.
- See SumPointOne.java.
- Why not use BigDecimal for everything?

↓ using bigdecimal will affect the efficiency, also, it consumes memory

Fall 2018

CISC124 - Prof. McLeod

Machine Epsilon - ϵ

- ϵ is the largest positive value that when added to 1 still produces 1, ie:

$$1 + \epsilon == 1$$

- Let's find out what this value is for `floats` and `doubles` (see `DetermineEpsilon.java`).

Fall 2018

CISC124 - Prof. McLeod

5

Machine Epsilon – ϵ , Cont.

- For float: 5.9604645E-8
- For double: 1.1102230246251565E-16

- If you are subtracting two numbers and the difference between the numbers is less than ϵ , then you will get zero.
- If you are adding two numbers, n_1 and n_2 , and $n_2/n_1 < \epsilon$, then you will get n_1 .

⇒ n_2 is too smaller than n_1 , therefore, $n_2 + n_1 = n_1$

Fall 2018

CISC124 - Prof. McLeod

↗ "too small to make a change"

虽然两个式子在数学上的逻辑是完全一样的, 但是, 后者在计算机中没有那么容易被 roundoff error 影响

The Effects of Roundoff Error

- Consider subtracting two numbers that are very close together:
- Use the function

$$f(x) = 1 - \cos(x)$$

for example. As x approaches zero, $\cos(x)$ approaches 1.

Fall 2018

CISC124 - Prof. McLeod

7

The Effects of Roundoff Error – Cont.

- Using `double` variables, and a value of x of $1.0\text{E-}12$, $f(x)$ evaluates to 0.0.
- But, it can be shown that the function $f(x)$ can also be represented by $f'(x)$:

$$f'(x) = f(x) = \frac{\sin^2(x)}{1 + \cos(x)}$$

- For $x = 1.0\text{E-}12$, $f(x)$ evaluates to $5.0\text{E-}25$.
- The $f'(x)$ function is less **susceptible** to roundoff error.

Fall 2018

CISC124 - Prof. McLeod

8

两个式子数学意义相同, 但后者没那么受 roundoff error 影响

The Effects of Roundoff Error - Cont.

- Another example. Consider the smallest root of the polynomial: $ax^2 + bx + c = 0$:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

- What happens when ac is small, compared to b ?
- It is known that for the two roots, x_1 and x_2 :

$$x_1 x_2 = \frac{c}{a}$$

Fall 2018

CISC124 - Prof. McLeod

9

The Effects of Roundoff Error - Cont.

- Which leads to an equation for the root which is not as susceptible to roundoff error in this case:

$$x_1 = \frac{-2c}{b + \sqrt{b^2 - 4ac}}$$

- This equation approaches $-c/b$ instead of zero when $ac \ll b^2$.

Fall 2018

CISC124 - Prof. McLeod

10

The Effects of Roundoff Error - Cont.

- These are examples of the two rules shown on the Epsilon slide (#6). What to watch for:

- A small difference between two large numbers.
- Adding a number that is too small to make a difference to a large number.

roundoff error 对结果上的两个层面上的影响

Fall 2018

CISC124 - Prof. McLeod

11

Summations - Convergence

- The rules on slide 6 also affect summations.
- Before looking at this – we should look at how you do summations on a computer.
- Since you can't sum to infinity (why not?), when do you stop a summation?
 - See Part 6 in Exercise 4 for one simple technique:
 - Stop the summation when the previous sum equals the current sum.
 - In other words the latest term in the summation follows the second rule on slide 6.

如何避免重复的计算过程

Fall 2018

CISC124 - Prof. McLeod

12

Summations Convergence, Cont.

- The exercise also demonstrated the need to choose summation formulae that provide rapid convergence.
- This minimizes the effect of roundoff error and greatly reduces the computation time.
- You can also make algorithmic choices that will improve the accuracy of your summations:
 - Kahan summation algorithm.
 - Alternating +/- summation trick.

Fall 2018

CISC124 - Prof. McLeod

13

Summations: Relative Magnitude Problem

- For example, consider this simple arithmetic sum:

$$S_n = \sum_{i=1}^n i = \frac{n(n+1)}{2}$$

- Let's fiddle with it a bit:

$$1 = \sum_{i=1}^n \frac{i}{S_n}$$

- Print out the values of the sum for $n = 10, 100, 1000, \text{etc.}$ (see FiddledArithmeticSum.java)

Fall 2018

CISC124 - Prof. McLeod

14

无论n变成多大,其结果都是1

在计算 $\frac{1}{S_n}$ 中, S_n 的结果会变得越来越大,以至 $\frac{1}{S_n}$ 会变得越来越小,直到小于 epsilon, 对结果无法造成影响

Fiddled Arithmetic Sum, Cont.

- Math tells us that the summation should be 1 regardless of the value of n .
- For a float, results show:

```
n =      10, sum = 1.000000
n =     100, sum = 1.000000
n =    1000, sum = 1.000000
n =   10000, sum = 1.000000
n =  100000, sum = 0.999999
n = 1000000, sum = 0.999900
n = 10000000, sum = 1.002663
n = 100000000, sum = 0.500000
```

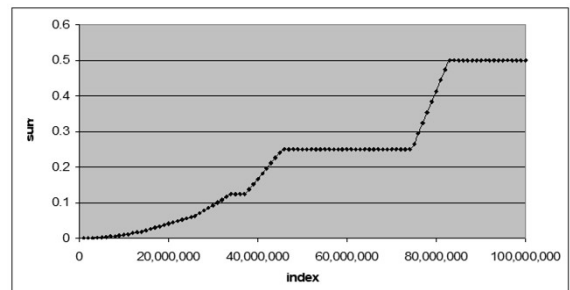
- What is going on for large value of n ?

Fall 2018

CISC124 - Prof. McLeod

15

Plot of sum versus index, for 100,000,000



- So the sum stalls out at 0.125, 0.25 and 0.5

Fall 2018

CISC124 - Prof. McLeod

16

Fiddled Arithmetic Sum, Cont.

- On the plateau areas of the plot, for a while, terms are too small to make a difference to sum, so these terms are lost.
- If you had some way to grab these "lost" terms and shove them back into the sum it should end up being closer to 1.0 again...

Fall 2018

CISC124 - Prof. McLeod

17

Kahan Summation Algorithm

- (William Kahan, a Comp. Sci. Prof at Berkeley, was into numerical computing and in the 80's led the committee that developed the IEEE754 standard.)
- Also called "Compensated Summation":

Calculate the portion of the term that does not contribute to the sum and then carry that portion to the next summation.

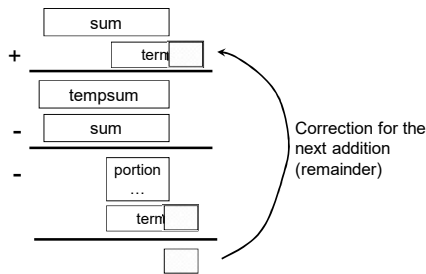
Fall 2018

CISC124 - Prof. McLeod

18

Kahan Summation Algorithm, Cont.

Calculate the portion of the term that does not contribute to the sum and then carry that portion to the next summation.

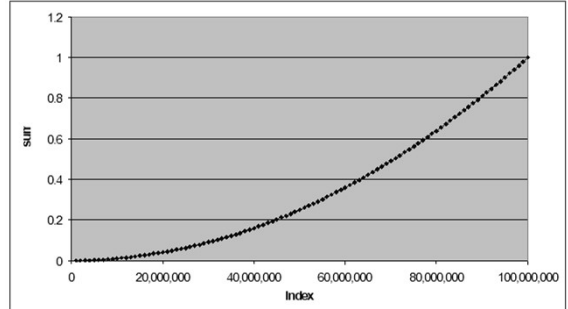


Fall 2018

CISC124 - Prof. McLeod

19

Using Kahan Sum



- See KahanSum.java

Fall 2018

CISC124 - Prof. McLeod

20

Remember our “Trouble Spots”?

- If you are subtracting two numbers and the difference between the numbers is less than ϵ , then you will get zero.
- If you are adding two numbers, $n1$ and $n2$, and $n2/n1 < \epsilon$, then you will get $n1$.
- We now have a technique to help us with the second problem for summations.
- How about the first one?

Fall 2018

CISC124 - Prof. McLeod

21

Small Difference Problem

- Here's another “demo” series, this one created by Ramanujan (a famous Indian mathematician, who created many amazing formulae in the early 1900's.)

$$0 = \sum_{k=0}^{\infty} (-1)^k \frac{(2k+1)^2 + (2k+1)^3}{k!}$$

$$\text{or, } 0 = \frac{1^2 + 1^3}{0!} - \frac{3^2 + 3^3}{1!} + \frac{5^2 + 5^3}{2!} - \frac{7^2 + 7^3}{3!} + \dots$$

- Note the alternating signs, and $0!$ is 1.
- See RamanujanSum.java.

Fall 2018

CISC124 - Prof. McLeod

22

Small Difference Problem, Cont.

- Using `double`'s the sum converges to:
4.960779927438127E-15
not zero.
- What's going on?
 - Hint: remember the alternating sign.
 - The sum is trying to get the difference between successive terms that are close in magnitude towards the end of the summation.
- How to get around the problem?

Fall 2018

CISC124 - Prof. McLeod

23

Small Difference Problem, Cont.

- How about adding all the negative terms to one sum, and all the positive terms to another sum, until neither sum is changing and then take the difference?
- See RamanujanSumFixed.java.
- This technique works well for these alternating sign sums.

Fall 2018

CISC124 - Prof. McLeod

24

Numeric Representation, Cont.

- Your understanding of the limitations of how numbers are stored on a computer will help you design computational code wisely!
- Work, work, work!:
 - You are now ready for exercise 4 and assignment 2.
 - The file I/O techniques covered in exercise 5 will not be discussed in class, but you can do this exercise now.

QUIZ 2 Prep

Everything up to and including tomorrow's material:

- Quiz 1 Java topics, but not Java History or Background (“How Java Works”).
- System, String, StringTokenizer, Wrapper classes.
- Method Overloading. – Catching, Building and Throwing Exceptions.
- Aliasing Objects, Passing by Reference.
- 2D Arrays. – Objects in general. Instantiation.
- Encapsulation
- private attributes, constructors, accessors, mutators and other standard methods.
- Exercises 1 to 9.
- File I/O from Exercise 5 – concepts only
- you won't have to write file I/O code.
- Unit Testing – concepts only – you don't need to write testing code.
- TDD.

System, String, StringTokenizer, Wrapper classes.

- The class defined in the **java.lang package** are automatically imported for you, since they are used quite often
- Java.lang package includes:
 - **The Wrapper classes**
 - Math
 - Object
 - **String**
 - **System**
 - Thread
 - —> 这些资料都可以从API那里知道 (java doc)
- **Aside: static Methods:**
 - static attributes and methods are **loaded once into memory and not garbage collected until main is finished**. These methods will **run faster** the second time (and later) they are invoked.
 - static methods can be invoked without instantiation of the Object that owns them. Math.random(), for example.
 - **static methods and attributes are shared by all instances of a class – there is only one copy of these methods in memory.**
 - **A static method can only invoke other static methods in its own class – you can't have pieces of code disappearing from a static method in memory...**
- 静态修饰符 (static)
 - 用来修饰成员变量：
 - 成员变量被static修饰，就叫做静态成员变量，如果没有修饰叫非静态成员变量

- 静态成员变量的访问方式：
 - 用对象进行访问，知道点语法拿到：对象名.变量名
 - 通过类名直接访问：类名.变量名
- 不要认为方便就把所有的变量设置为静态，只有当数据真正需要共享时才需要用static访问
 - 非静态变量只能用对象来访问，不能用类来访问
- 静态变量随着会类文件的加载而加载，随着类文件的消失而消失
- **Instantiation (实例化)**：
 - 在面向对象的编程中，通常把用类创建对象的过程称为实例化，其格式如下：

如 `Date date=new Date();`就是用日期类创建了一个日期的对象，就叫对象的实例化。

- **Math Class**

- A collection of static constants and static mathematical methods

- **Wrapper Classes***

- Sometimes it is necessary for a primitive type value to be an Object, rather than just a primitive type.
- Some data structures only store Objects.
- Some Java methods only work on Objects.
- 在java里面，有一些method或者是结构只能用在object上面，这个class是为了把primitive type存为一个object
- Wrapper classes also contain some useful constants and a few handy methods.
- Each primitive type has an associated wrapper class：

char	Character
int	Integer
long	Long
float	Float
double	Double

- 左边的是primitive type，右边对应的是他的wrapper class
- 每个不同的wrapper class的object都可以储存放在primitive type里的数据，但是现在我们可以应用一些本来就放在了这个class的method

```
Integer number = new Integer(46); // "Wrapping"
Integer num = new Integer("908");
```

```
Integer.MAX_VALUE // gives maximum integer
Integer.MIN_VALUE // gives minimum integer
Integer.parseInt("453") // returns 453
Integer.toString(653) // returns "653"
number.equals(num) // returns false
```

```
int aNumber = number.intValue(); // aNumber
is 46
```

- 上面这些是针对Integer的，而在double里面也有一样的method
- 在以上的方法中，第一个用来instantiate的方法，已经depreciated掉了

- 现在API更推荐的是用Integer.valueOf()
 - Do: Integer aTest = Integer.valueOf(42);
- Aside - depreciation:
 - something that has been taken out from language
- The Character wrapper class:
 - has methods to convert between ASCII and Unicode numeric values and characters.
 - isDigit(character) returns true if character is a digit.
 - isLetter(character)
 - isLetterOrDigit(character)
 - isUpperCase(character)
 - isLowerCase(character)
 - isWhitespace(character)
 - toLowerCase()
 - toUpperCase()
- **System Class***
 - System.currentTimeMillis()
 - Returns, as a long, the number of milliseconds elapsed since midnight Jan. 1, 1970.
 - 这个方法是在你invoke的这个method的时候，the number of milliseconds elapsed since midnight Jan. 1, 1970，如果你在程序一开始写一个，在程序结束的时候也写一个，那么把他们互相减去，就可以得到一个你的程序运行的时间
 - 还有另一个conversion用来extract the day from the value
 - System.exit(o)
 - Immediate termination of your program.
 - System.getProperties()
 - All kinds of system specific info
 - System.nanoTime()
 - Time in nanoseconds
- **String Class**
 - **Escape sequences** in Strings:
 - These sequences can be used to **put special characters into a String**:
 - \" a double quote
 - \' a single quote
 - \\ a backslash
 - \n a linefeed
 - \r a carriage return
 - \t a tab character

- **String** literals:
`"Press <enter> to continue."`
- **String** variable declaration:
`String testStuff;`
or:
`String testStuff = "A testing string.";`
- **String** concatenation ("addition"):
`String testStuff = "Hello";`
`System.out.println(testStuff + " to me!");`
Would print the following to the console window:

Hello to me!

- String method(总共有67个):
 - `length()`
 - `equals(OtherString)`
 - `equalsIgnoreCase(OtherString)` // 不在乎大小写的比较
 - `toLowerCase()`
 - `toUpperCase()`
 - `trim()`
 - `charAt(Position)`
 - `substring(Start)`
 - `substring(Start, End)` // stops one location before end
 - `indexOf(SearchString)` // return -1/exception if it can't find the string
 - `replace(oldChar, newChar)`
 - `startsWith(PrefixString)`
 - `endsWith(SuffixString)`
 - `valueOf(integer)`
- String is **immutable** - they cannot be altered, only can be re-assigned
- However, **Arrays are mutable**, in contrast - any element can be changed

• Other java.lang Classes

- Object class
- Thread : a base class used to create threads in multi-threaded program.

• StringTokenizer Class*

- This class is not in java.lang, this is in java.util.
- you need to have an import java.util.*; or import java.util.StringTokenizer;
- This class provides an easy way of **parsing strings up into pieces, called "tokens"**
- **Tokens are separated by "delimiters", that you can specify, or you can accept a list of default delimiters.**
- The constructor method for this class is overloaded.
- So, when you create an Object of type StringTokenizer, you have three options:
 - `new StringTokenizer(String s)` //这里设置的delimiter是default delimiter : \t\n\r, space, tab, line feed, carriage return
 - `new StringTokenizer(String s, String delim)` //delim就是你自己具体设置的delimiter
 - `new StringTokenizer(String s, String delim, boolean returnTokens)` //后面的boolean如果是true的话，那么最后return回来的也包括delimiter本身

```
String aString = "This is a String - Wow!";
StringTokenizer st = new StringTokenizer(aString);
System.out.println("The String has " +
    st.countTokens() + " tokens.");
System.out.println("\nThe tokens are:");
while (st.hasMoreTokens()) {
    System.out.println(st.nextToken());
} // end while
```

The String has 6 tokens.

The tokens are:

This
is
a
String
-
Wow!

- 当token remove了之后，StringTokenizer Object 就被清空了，想要重新tokenize的话，只能重新创造
- Scanner Class Tokenizer
 - The Scanner class has a tokenizer built into it
 - **Scanner uses a regular expression or “regex” instead of the (easier to understand, but less powerful!) delimiter list.**
 - The default regex is: "\p{javaWhitespace}+" which means “any number of whitespace characters”
 - A whitespace character is a space, a tab, a linefeed, formfeed or a carriage return
 - "\t\n\f\r" in other words.

• Method Overloading

- **“Overloading” is when a method name is used more than once in method declarations within the same class. (also like println(...))**
- 这个指的是，在这个**method declare**的**class**里面，同样的**method name**被匹配到了不同数量/种类的**variable**，**method name**被多次使用
- The rule is that no two methods with the same name within a class can have the same number and/or types of parameters in the method declarations. (The “NOT” rule.)
- 目的：
 - 允许用户在使用这个method的时候可以不用supply那么多argument
 - 一个method可以对不同的数据类型做出action
 - Allows the programmer to keep an old method definition in the class for “backwards compatibility”.
 - 如果语言的改变导致程序员对这个method不了解，overloading可以保证之前的definition从而提高程序员的使用
- **How does it work**
 - Java looks through all methods until the parameter types match with the list of arguments supplied by the user. **If none match, Java tries to cast types in order to get a match.**

(Only “widening” casting like int to double, however)(即使java要cast，也只会顺着那个顺序cast)

- Do not change the return type!!

- **Exceptions**

- Exception is another way to get something out of a method.
- Exception is thrown, not returned
- Exception is Objects
- When an error condition is encountered, a method can throw an instance of a pre-defined exception Object.
- A method can throw several exceptions, one for each possible kind of error condition.
- If a method throws an exception, then that **method is immediately halted** and there is no need for any return value, even if the method is non-void.
- Exception本身是不可以对程序作出任何的修改的，但是他可以停止这个程序
- 下面的method会receives 这个被throw出来的exception的，如果这个method没有抓住这个exception的话，那么就会invoke下一个method，如果一直没抓到，会一直到main，这个过程叫做cascading，如果到了main还没有被抓到的话，那程序就会crash，Finally, if main does not catch the exception, your program crashes and a message is sent to the console window.
- Exception Object:
 - Type of the Object:
 - IOException
 - NumberFormatException
 - FileNotFoundException
 - ArrayIndexOutOfBoundsException
 - ...
 - 所以对于设计者来说，应该设计一个与之相关的exception
 - Exception还可以包含String message
 - 只要是invoke了一个throw exception的method的话，compiler会force你去用try/block

```
1 try {
2 // block of statements that might
3 // generate an exception
4 } catch (exception_type identifier) {
5 // block of statements
6 }[ catch (exception_type identifier) {
7 // block of statements
8 ...
9 }][ finally {
10 // block of statements
11 }
```

- 在try block后面，必须要有至少一个exception
- The code in the “finally” block is always executed, whether an exception is thrown, caught, or not.
- Checked vs Unchecked Exceptions
 - Unchecked Exception就是出现在Error Class的Error，这些错误不是Exception，serious

problem，不是通过修改你的这一个particular code就可以改过来的。Unchecked exceptions occur only at runtime and the compiler does not care about them.(OutOfMemoryError, StackOverflowError, VirtualMachineError)

- Checked Exception是一定要被你的code里面的一个method抓住的，是可以通过修改code从而避免的(IOException, FileNotFoundException, ClassNotFoundException)
- Try With Resources

```
1 try (instantiation; instantiation; ...) {
2 // other statements that might
3 // generate an exception
4 }[ catch (exception_type identifier) {
5 // block of statements
6 }][ catch (exception_type identifier) {
7 // block of statements
8 ...
9 }][ finally {
10 // block of statements
11 }]
```

- 有的时候如果你的代码具体是对哪个文件作操作的话，如果操作的这个method把文件打开了，遇到了exception，method停止，可是文件也没有关上
- 但是在Try with里面，try旁边的括号打开了这个文件，只要这个代码try一结束，那么就会关上这个file

```
1 public class IllegalHalloweenException extends Exception {
2
3     /**
4      * Supplies a default message.
5      */
6     public IllegalHalloweenException() {
7         super("Illegal parameter value supplied to Halloween object.");
8     }
9
10    /**
11     * Passes along the message supplied to the exception.
12     * @param message A more specific message.
13     */
14    public IllegalHalloweenException(String message) {
15        super(message);
16    }
17
18 }
```

• Array

- To create an array to hold 10 integers: `int[] testArray = new int[10];`
 - `var testArray = new int[10];`
- 在上面的declare之后，知道这时候，testArray只是一个pointer，他point to an area of memory

that holds locations for 10 integers

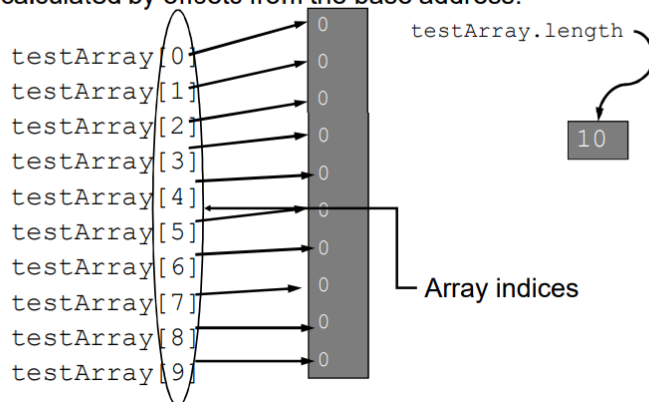
- It also points to one location that holds `testArray.length` which is an attribute of the array, that is equal to the number of elements.

- 同时我们在declare array的时候，也可以分开写

- `int[] testArray;`
- `testArray = new int[10];`

- 这样子写下来的时候我们知道，`testArray` is now an object of type `int[]` that contains an “object reference” or a pointer. **The object reference is null after the first statement.**
- 当我们到了第二行的时候，这个时候，*he point to an area of memory that holds locations for 10 integers*

- The array indices allow mutability. Memory locations are calculated by offsets from the base address:

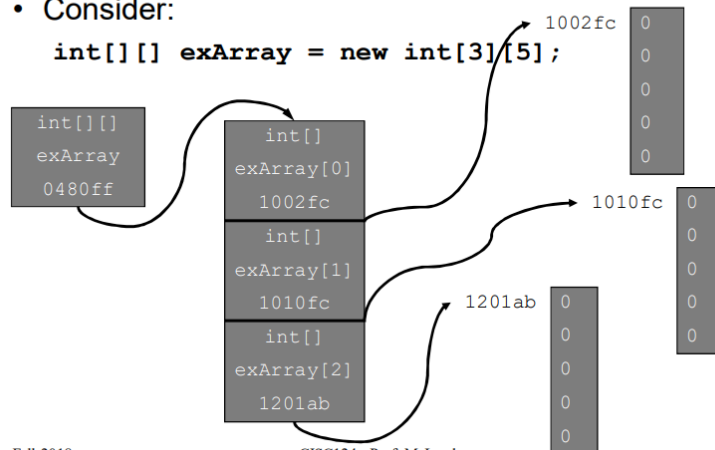


- 整个java在进行这个操作的过程是这样的，这个时候，java知道我们这个array里面存的肯定是integer，每一个是4 byte，那么如果存了九个数字，就是 $9 \times 4 = 36$ byte，当我们输入一个数字，比如说4，java会算出在那个时候，内存占用是16 byte，那么就会跳出对应的数字
- Multi-Dimensional Arrays

Multi-Dimensional Arrays

- Consider:

```
int[][] exArray = new int[3][5];
```



Fall 2018

CISC124 - Prof. McLeod

30

- 理解他的时候就可以理解成，一个大的pointer，point到一个array，array里面每一个elem都是一个pointer，然后再分别point到array中
- 对于第一个存pointer的array里面存的pointer来说，他们point去的array不一定要一个size的

- Array initializer for two-D array, for example:

```
int[][] twoDArray = {{1, 2, 3},
                    {4, 5, 6},
                    {7, 8, 9},
                    {10, 11, 12}};

System.out.println(twoDArray.length); // 4
System.out.println(twoDArray[0].length); // 3
```

■

• Aliasing Object

- 这个的意思是说，如果有两个Array，first和second都是两个已经被指向了的pointer，在这种情况下，让second = first，这样的话，second也会成为一个指向first所指向的Array的pointer，那么这个时候，second本来指向的Array就变成了garbage
 - Java has an automatic Garbage Collection system: – Variables are garbage collected once you move outside their scope. – Object contents are garbage collected when there are no pointers pointing to the contents.

• Passing Parameters by Reference

- 就是说把一个Array pass进一个method里面，这样的话，在parameter的时候，parameter里面的array就会和外面的array aliasing起来，所以改变里面的pointer对应的array就会改变外面的
- The rule for parameter passing into methods is: – Objects are passed by reference, primitive types are passed by value
- So, mutable Objects (like arrays) can be passed into and out of a method through the parameter list. If a method changes the contents of a mutable Object passed into it – those changes “stick” even when the method is complete.

• Array Equality

- This test will only give a true when both objects have been aliased, using the assignment operator “=”.
- So, even if both arrays have identical contents, “==” will return false, unless both arrays point to the same location.
- This means that comparing Objects with “==” only compares pointers, not contents.

• Null Pointer or Null Reference

- Null is not a keyword in Java – more like a literal constant.
- Use Null == Arrayname to test if Arrayname is null pointer

• Objects in general

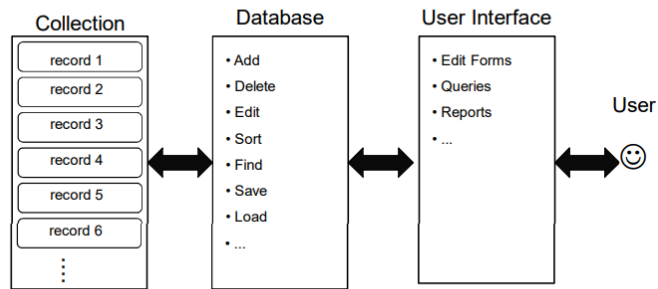
- What is an Object?
 - An entity that exists in an operating computer program that has:
 - – State
 - is the collection of information held in that object. This information may change over time, as a result of operations carried out on the Object.
 - – Behaviour
 - is the collection of operations that an Object supports.
 - – Identity
 - allows the program access to a specific Object.
 - 一个可以储存信息，并对信息进行操作，并可以返还一些信息的对象

- What is a class?
 - **If each of these Objects has the same set of possible behaviours then you can group these Objects together into a Class**
 - A class is defined in the source code of a program.
 - The operations that are allowed on instances of this class (the methods).
 - The possible categories of state that are allowed for instances of this class (the attributes).
 - However, **attributes cannot be added or removed and behavior cannot be added or removed.** These are defined in the Class.
- Object Categories
 - – Tangible things (ex: Cat) >>> a real object
 - – Agents (StringTokenizer) >>> acting on other object
 - – Events and transactions (MouseEvent)>>>interact with user
 - – Users and roles (Administrator)
 - – Systems (MailSystem)
 - – System interfaces and devices (File)
 - – Foundational classes (String)
- Object Structure
 - Two extremes of object structure:
 - – Utility classes:
 - All static methods and attributes
 - – The Math class, for example.
 - – You do not instantiate these classes – there is no point.
 - – Customizable classes:
 - All non-static methods and attributes.
 - Attribute values (some or all) must be set at the time of instantiation before the class can be used.
 - Scanner class for example.
 - And many classes fall in-between these two extremes:
 - – A mix of static and non-static methods.
 - – static methods have nothing to do with the attributes and so can be used without instantiation of the class.
 - – Non-static methods depend on the attributes which must be set through instantiation.
 - – Wrapper classes for example: Double, Integer, etc.

- **Encapsulation**

- **Encapsulation is the process of defining a Class that has at least one customizable attribute.**
- In Java, methods and attributes must be encapsulated or contained in a class definition.

- Consider the general model (or “pattern”...) of a database:



- • Each record is an instance of an Object, too.

IllegalHalloweenException.java

这是一个 Exception class 的例子

```
1 /**
2  * An Exception thrown by the Halloween4 Object if parameters are not legal.
3  * <ul>
4  * <li>The year must be between 1959 and 2016 inclusive.</li>
5  * <li>The number of kids must be between 0 and 500.</li>
6  * <li>The temperature values must lie between -30 and 30.</li>
7  * <li>The weather condition can be "rain", "snow" or "clear". The default condition is
   "unknown".</li>
8  * </ul>
9  * @author Alan McLeod
10 * @version 1.0
11 */
12
13 public class IllegalHalloweenException extends Exception {
14
15     /**
16      * Supplies a default message.
17      */
18     public IllegalHalloweenException() {
19         super("Illegal parameter value supplied to Halloween object.");
20     }
21
22     /**
23      * Passes along the message supplied to the exception.
24      * @param message A more specific message.
25      */
26     public IllegalHalloweenException(String message) {
27         super(message);
28     }
29
30 } // end IllegalHalloweenException
```

这是一个 default 的 message, 也就是说
在接下来的使用中可以不输入任何的
String message

Halloween5.java

```
1 import java.io.Serializable;
2
3 /**
4  * A class to store Halloween information.
5  * <p>
6  * The year, the number of visitors, hourly temperatures in deg C and the weather condition
   is recorded. This class
7  * has been created as a lecture example, and is not otherwise particularly useful!
8  * <p>
9  * This version demonstrates the implementation of Comparable (for sorting) and Serializable
   (for
10 * filing). Also, the mutators for temperature and weather condition have been combined, so
   both
11 * attributes have to be set at the same time. In this way they cannot be set to an illegal
   value
12 * independently.
13 *
14 * @author Alan McLeod
15 * @version 3.2
16 */
17 public class Halloween5 implements Comparable<Halloween5>, Serializable {
18
19     private static final long serialVersionUID = 4705089863030936649L;
20     private int year;
21     private int numMunchkins;
22     private int[] temperatures;
23     private String weatherCondition;
24
25     /**
26      * Full parameter constructor.
27      * @param yr The year when the data was collected.
28      * @param numKids The number of Trick or Treaters!
29      * @param temps The air temperatures in degrees Centigrade in an array of int of any
   size.
30      * @param weather The weather condition: "clear", "snow" or "rain".
31      * @throws IllegalHalloweenException If arguments are not legal.
32      */
33     // 4 parameter constructor invokes mutators
34     public Halloween5(int yr, int numKids, int[] temps, String weather) throws
   IllegalHalloweenException {
35         setYear(yr);
36         setNumMunchkins(numKids);
37         setWeather(temps, weather);
38     }
39 }
```

Attributes must be declared as private, so that the class that owns them can control how they are set

must
be
public

→ constructor, to set all parameters

```

38     } // end Halloween4 4 parameter constructor
39
40     /**
41     * Three parameter constructor. The weather condition does not have to be supplied.
42     * @param yr The year when the data was collected.
43     * @param numKids The number of Trick or Treaters.
44     * @param temps The air temperatures in degrees Centigrade in an array of int of any
size.
45     * @throws IllegalHalloweenException if arguments are not legal.
46     */
47     // 3 parameter constructor invokes 3 parameter constructor with an assumption about the
48     // weatherCondition attribute
49     public Halloween5(int yr, int numKids, int[] temps) throws IllegalHalloweenException {
50         this(yr, numKids, temps, "unknown");
51     } // end Halloween4 3 parameter constructor
52
53     /**
54     * Sets the year the data was recorded.
55     * @param yr The calendar year.
56     * @throws IllegalHalloweenException if the year does not lie between 1959 and 2016
57     */
58     public void setYear(int year) throws IllegalHalloweenException {
59         if (year < 1950 || year > 2018)
60             throw new IllegalHalloweenException("Illegal year: " + year);
61         this.year = year;
62     } // end year mutator
63
64     /**
65     * Sets the number of kids.
66     * @param numKids The number of kids arriving at the door.
67     * @throws IllegalHalloweenException if the number of kids is less than zero or greater
68     * than 500.
69     */
70     public void setNumMunchkins(int numKids) throws IllegalHalloweenException {
71         if (numKids < 0 || numKids > 500)
72             throw new IllegalHalloweenException("Illegal number of kids: " + numKids);
73         numMunchkins = numKids;
74     } // end numMunchkinds mutator
75
76     /**
77     * Sets the temperatures array and the weather condition String. The temperatures are
78     * recorded with one temperature per hour.
79     * @param temps An array of temperatures between -30 and 30 degrees C.
80     * @param weather The weather condition as a String.
81     * @throws IllegalHalloweenException if the condition is not "rain", "snow", "clear" or
"unknown",
82     * or if the array is empty or any temperatures are not legal.
83     */
84     public void setWeather(int[] temps, String weather) throws IllegalHalloweenException {
85         double avgTemperature = 0;
86         if (temps.length == 0)

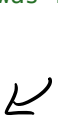
```

这些就是 mutator, 可以设
 置成 private —— 如果不
 想之外又对某个 attribute 进
 行更改的话

```

87         throw new IllegalArgumentException("No temperatures supplied");
88     for (int temperature : temps) {
89         if (temperature > 30 || temperature < -30)
90             throw new IllegalArgumentException("Illegal temperature in array: " +
temperature);
91         avgTemperature += temperature;
92     }
93     temperatures = temps.clone();
94     avgTemperature = Math.round(10 * avgTemperature / temperatures.length) / 10.0;
95     if ((weather.equalsIgnoreCase("rain") && avgTemperature > -5) ||
96         (weather.equalsIgnoreCase("snow") && avgTemperature < 5) ||
97         weather.equalsIgnoreCase("clear") || weather.equalsIgnoreCase("unknown")) {
98         weatherCondition = weather;
99     } else
100         throw new IllegalArgumentException("Illegal weather/temperature combination: "
+
101         weather + ", " + avgTemperature + " deg C.");
102     } // end setWeather mutator
103
104     /**
105     * Returns the calendar year the data was recorded.
106     * @return The year the data was recorded.
107     */
108     public int getYear() {
109         return year;
110     } // end getYear
111
112     /**
113     * Returns the number of visitors.
114     * @return the number of Trick or Treaters.
115     */
116     public int getNumMunchkins() {
117         return numMunchkins;
118     } // end getNumMunchkins Accessor
119
120     /**
121     * Returns the temperatures array.
122     * @return The temperatures in degrees Centigrade.
123     */
124     public int[] getTemperatures() {
125         return temperatures.clone();
126     } // end getTemperature Accessor
127
128     /**
129     * Returns the weather condition.
130     * @return The weather condition as a String.
131     */
132     public String getWeatherCondition() {
133         return weatherCondition;
134     } // end getWeatherCondition Accessor
135

```

 这些是 accessor, 设置成 private 没有意义

```

136  /**
137  * A String representation of the current object.
138  * @return A String representation of the contents of the object containing the values
of
139  * all the attributes.
140  */
141  // Overrides (replaces) the toString method of the Object class.
142  @Override
143  public String toString() {
144      String s = "In " + year + " there were " + numMunchkins + " kids. Temperatures each
hour were: ";
145      for(int i = 0; i < temperatures.length - 1; i++)
146          s += temperatures[i] + ", ";
147      s += "and " + temperatures[temperatures.length - 1];
148      s += " deg C., and the weather was ";
149      s += weatherCondition + ".";
150      return s;
151  } // end toString
152
153  /**
154  * Tests two Halloween5 objects for equality.
155  * @return <code>true</code> if all the attributes of both objects are exactly equal,
<code>false</code>
156  * otherwise.
157  * @param otherObject The other Halloween5 object.
158  */
159  // Overrides the equals method of the Object class.
160  @Override
161  public boolean equals(Object otherObject) {
162      if (otherObject instanceof Halloween5) {
163          Halloween5 otherH = (Halloween5)otherObject;
164          boolean arrayCheck = true;
165          if (otherH.temperatures.length != temperatures.length)
166              return false;
167          for(int i = 0; i < temperatures.length && arrayCheck; i++)
168              arrayCheck = temperatures[i] == otherH.temperatures[i];
169          if (arrayCheck)
170              return year == otherH.year &&
171                  numMunchkins == otherH.numMunchkins &&
172                  weatherCondition.equalsIgnoreCase(otherH.weatherCondition);
173      } // end if
174      return false;
175  } // end equals
176
177  /**
178  * Compares Halloween5 objects on the basis of the number of visitors only.
179  * @param otherH The other Halloween5 object.
180  * @return A negative <code>int</code> if the supplied object had more vistor, zero if
they have the same
181  * number and a positive number if the current object has more visitors.
182  */

```

javac

```
183     public int compareTo(Halloween5 otherH) {
184         return numMunchkins - otherH.numMunchkins;
185     } // end compareTo
186
187     /**
188     * Returns a copy of the of the current Halloween5 object.
189     * @return A copy of the current object.
190     */
191     // Overrides the clone method in the Object class.
192     @Override
193     public Halloween5 clone() {
194         Halloween5 hwCopy = null;
195         try {
196             hwCopy = new Halloween5(year, numMunchkins, temperatures, weatherCondition);
197         } catch (IllegalHalloweenException e) {
198             // Should never get here!
199             return null;
200         } // end try/catch
201         return hwCopy;
202     } // end clone
203
204 } // end Halloween5
```

-
- Everything up to and including Thursday's lecture:
 - Enumerated Types.
 - Inheritance.
 - Polymorphism.
 - Interfaces, Anonymous Classes, Inner Classes, Abstract Classes.
 - ArrayList<T> Collection Type.
 - Generics: Generic Classes & Methods, Use of Wildcards. The Class<T> Object.
 - Lambda Functions.
 - Method References.

Method Reference

假设你要在别的地方使用某个 Object 中的 method / content

现有的 class 中, 有了一个 ArrayList of type Object, ArrayList of db,
db.sort (Object :: compareby Age) ;



这是一个在 "Object" class 中的 method

Lambda Function

假设有一个 interface, 它里面有一个 abstract method (只有 method header, 但是没有内容的), 那么我们需要一个 Anonymous Class / Lambda Function 来补充解释这个 method

例:

```
public interface MessageSender {  
    void sendGreeting (String name);  
}  
// end MessageSender interface
```

这是早就定义的 interface, 其中, sendGreeting 这个 method 是一个 abstract method
∴ 接下来需要对其补充

```
public class LambdaDemo {  
    public static void main(String[] args) {  
        MessageSender ms =  
            name -> System.out.println("Hello " + name);  
        ms.sendGreeting("Steve");  
    }  
} // end LambdaDemo
```

→ 这就是 lambda function 的具体使用

图起来的是
entire sentence

使用

① 几个 parameter → 直接写就好, 不用 {}
多个 parameter → (a, b, c, ...), 不需要将 param 的 type
没有 parameter → {} 代替

② Arrow: ->

③ → 后面跟 method body, body 只有一句的话, 需要加 {}

* 只能应用在只拥有一个 abstract method 的 Functional Interface 上, 这个 Interface 中的 abstract method 决定了 Lambda Function 的结构

* Functional Interface: 只有一个 abstract method 的 interface

```
displaySome(db, pers -> {return pers.getAge() >= 20;});
```

Genetic Class

* Kind of like having a parameter in class header, 但这个 "parameter" 不能 pass 任何 reference, 可以 specifies a type

* An Example

```
public class Sample<T> {  
    private T data;  
  
    public void setData(T newData) {  
        data = newData.clone();  
    }  
  
    public T getData () {  
        return data;  
    }  
} // end Sample<T>
```

CISC124 - Prof. McLeod
Fall 2018

=> 使用 =>

```
Sample<Integer> num2 = new Sample<>();  
num2.setData(67);  
System.out.println(num2.getData());
```

=> 可以用在任何 Object type 上, 除了 Exception

=> 不可以用在 primitive type 上

=> Can we interface and abstract class

如果要在这个 class 中加一个 constructor 的话 =>

```
public Sample (T newData) {  
    data = newData.clone();  
}
```

Automatic boxing: 这是一个属于 Generic class 的一个 advantage, 就是我们对某个 object 进行操作时, 我们可以不用 instantiate 那个 object, 而是直接使用, generic class 会自动帮你 box 起来

例:

```
Sample<Double> num3 = new Sample<>(56.7);  
System.out.println(num3.getData());
```

 -> 不需要 instantiate

Automatic unboxing: 当你有了一个 ArrayList 中的东西, 你可以直接把它提出来, 存入一个 primitive type

例:

```
double aval = myList.get(0);
```

Generic Class:

Bounding:

* 在 Generic class header 中可以 supply any number of parameters: `public class Sample<T1, T2>{`

* 可以利用 `extends / super` 这两个 key word 去限制放进来的 object type

使用 { `<T extends RootClass>` : 只能放 RootClass 的 } upper bound
 `<T extends RootClass & Pizza>` : 只能放 RootClass 和 Pizza 的 } bound
 `<T super Integer>` : 只能放 Integer, Number, Object 的 → lower bound

⇒ bounding 了之后, 也会确保在接下来的 method body 中, T 只有你 bound 中的 class 的 method

Generic Method:

* 和 Generic class 的定义差不多, 不过是对 method 而言

```
public static <T> T getMidpoint(T[] a) {  
    return a[a.length/2];  
}
```

⇒ invoke ⇒

`String mid = Utility.getMidpoint(cb);`

不需要 泛型

Some Exception:


`Integer midI = (Integer) Utility.getMidpoint(cs-rings);` → "ClassCastException"

`Integer midI = Utility.getMidpoint(cs-rings);` → Compilation Error

* Constructors can also be generic. Even in a non-generic class

Generic Wildcards:

* A wildcard is the ? within the < >, A wildcard can provide a super (超级) type generic class for other generic classes

例: `ArrayList<?> aObj = ArrayList<Child>();`


* Binding

可以把一个 wildcard 与 其他 class bind 起来 用 extend/super key word

* Immutable:

一个添加? wildcard 的 generic class 是 immutable 的

* `ArrayList<?>[]` is the only possible type that allows the creation of an array of generic classes

* The object class contains a method, `getClass()` that returns a class<T> object

* Type Introspection: process of discovering class structure at run time (methods, annotations, attributes, inner classes, etc.) through a class<T> object

* Reflection: what happens when you use this information at run-time

ArrayList Collection Type

* It's a generic data structure, belongs to `java.util.package`

* Declaration: `ArrayList <type> list_name = new ArrayList <type> ();`

↓
- `type` is an object type

size: 可以自己定义一个 size, 但是其实没必要, 因为对于 ArrayList, 它的 size 会自动增加

type: 在 declaration 式的左边, 这个 type 可以省略, 只写 `<>`

* Add element:

`myList.add(456.78);`

不需要提前 instantiate → Automatic Boxing

* get the size of the collection

`myList.size()` → return 一个 int

* `get(index)` → returns the element of type T at the given index
The index positions are numbered from zero

* `set(position, new-value)` → changes the element at the given position

* `add(position, new-value)` → new value is inserted to the position

* `remove(position)` → removes the element at the provided position

* `trimToSize` → remove unallocated position

* `clone()` → 只会出一个 pointer

* `toArray()` → 把一个 `ArrayList` 的 array 转换成一个 `Object` 类型的 `ArrayList`

在 declare ArrayList 的时候还可以:

ArrayList <Type> List_name = new ArrayList <T> ()

↓
里面可以是空的, 这种叫做
diamond, this is an aspect
of "type inference", just
like when java sees the
"Type", it fills the blank
for you

* { ArrayList <T> 是 mutable 的
* { Regular List 是 mutable 的
* { String 是 immutable 的

Inner Class, Anonymous Class, Abstract

Inner Class

* A class defined within a class

例:

```
public class OuterClass {  
  
    private class InnerClass {  
        Attributes of innerclass  
        Methods of innerClass  
    } // end InnerClass  
  
    Attributes of OuterClass  
    Methods of OuterClass  
  
} // end OuterClass
```

* 一般 defined as private
* inner class 可以直接使用 outer class 的内容

* outer class 要先 instantiate inner class, 才可以使用 inner class 的内容

* 其他的 class 要引用 inner class 的 method 的话: OuterClass.InnerClass.method

Anonymous Class

假设有一个 interface, 它里面有一个 abstract method (只有 method header, 但是没有内容的), 那么我们需要一个 Anonymous Class / Lambda Function 来补充解释这个 method

```
public interface MessageSender {  
  
    void sendGreeting (String name);  
  
} // end MessageSender interface
```

→ interface



on Anonymous Class



```
public class AnonymousClassDemo {  
  
    public static void main(String[] args) {  
        MessageSender ms = new MessageSender() {  
            public void sendGreeting (String name) {  
                System.out.println("Hello " + name + "!");  
            } // end sendGreeting  
        };  
        ms.sendGreeting("Alan");  
    } // end main  
} // end AnonymousClassDemo
```

格式



Abstract Class

* A class can also have concrete method definitions and any kind of attribute but it can only be extended.

例: `public abstract class MyClass ...`

Abstract method

* 一个只有 header 却没有任何内容的 method

例: `public abstract int multiply (int a, int b);`

* 如果一个 class 有 abstract method 的话, 就一定要被 defined as abstract class

* 一个 extended 了 abstract class 的 class 必须要么是一个 abstract class, 要么 override 所有 abstract class 中的 abstract method

Interface

Interface 是一个可以有

{	Constant attribute	的 class, 它要被 implemented
	abstract method	
	Default method	
	Static method	

* Interface 本身不可以 extend object, 但可以 extend interface

* Interface 可以被 implemented, 可以 implemented 几个

例:

```
public class Test implements interface1,  
    interface2, interface3, ... {}
```

* 如果两个被 implement 的 interface 中有一样的 method name, 那么在 implement 的时候也必须也 implement method

Inheritance

Parent class / Super class



- * Child class 可以直接继承 parent class 的 public attribute and method
- * 其实所有的新 Object 都算是 Object class 的 child class
- 格式: `Public class child extend Parent { ... }`
- * Compiler will force you to invoke super in a child class' constructor, and it must be the first line in the constructor
- * Belongs to polymorphism

Subclass 的 method

- 一个新的 method
- 直接继承 inherit parent class 的 method
- Override the method
- Overload the method
- Refine, 在 sub class 的 method 中先 invoke parent class 的 method 再写, 从而达到修改的目的

- * final : 说明这个 class 不会再被 extend 了
- * 一个 class 只能 extend 一个 class

Enumerated Type

enum is a key word, and is a type

例

```
enum IceCream {CHOCOLATE, VANILLA,  
    STRAWBERRY, GOLD_MEDAL_RIBBON, WOLF_PAWS,  
    BUBBLE_GUM};
```

→ *collection of Constant
*is immutable

* equals() , 也可以直接用 ==

* toString() ,

* compareTo() ,

* ordinal() returns the numeric location of the value in enum list

* values() 把 enum array 转化成一个普通 array
array of this object

~~enum~~

* values() : IceCream[] flavours = IceCream.values();

```

public class Missing extends Abs implements Interface{

    @Override
    public int remainder(int a, int b) {
        // TODO Auto-generated method stub
        return a % b;
    }

    @Override
    public int subtract(int a, int b) {
        // TODO Auto-generated method stub
        return a - b;
    }

    public int sum(int a, int b) {
        return a + b;
    }
}

```

```

public static <T extends Number & Comparable<T>>
ArrayList<T> lessThan(ArrayList<T> array1, T obj) {
    ArrayList<T> newArray = new ArrayList<>();
    for (T value : array1) {
        if (value.compareTo(obj) > 0)
            newArray.add(value);
    }
    return newArray;
}

```


* 不熟悉的点:

Screen input using the Scanner class:

① 首先需要 import 这个 class: `import java.util.Scanner;`

注意: import statement 是写在 class definition 上面的

② instantiate 一个新的 Scanner class object

`Scanner screen = new Scanner(System.in);`

然后写上以为需要 print 出在屏幕上的文字:

`System.out.println(" ~ ");`

然后再抓住 user 输入的东西

* `int num = screen.nextInt();` → 抓 integer

`String str = screen.nextLine();` → 抓 string

* Integer Wrapper Class: instantiate: `Integer aNum = Integer.valueOf(~);`

* String Class 的 method: pg 35

* String tokenizer: 可以把一个 string 变成许多 little pieces → pg 35

使用 StringTokenizer `st = new StringTokenizer(aString);`

`while (st.hasMoreTokens()) {`

`System.out.println(st.nextToken());`

`}`

→ immutable

* Enum: a keyword, a type, 相当于一个 collection of constant string.

declar: `enum whatever { HARRY, BABY, HIDEO }`

→ 全是大写

* 这时, whatever 是一个 object 了

可以这样用: `whatever anything = whatever.HARRY`

那么在此时, anything 就直接指代 HARRY 了

其它参见 pg 60

非常弱项:

- ① generic class
- ② ArrayList <T>
- ③ Early Binding / Late Binding
- ④ Java FX

* Generic method:

method header: public static <T> return type method name (args) {
}

* Abstract method: public abstract return type method name (args);

* Abstract class header: public abstract class classname { ... }

* Early Binding / Late Binding

- 都属于 Polymorphism → when a pointer of a parent class type ends up pointing to different child class objects at runtime (dynamic binding),
- Early Binding 指的是 when the parent class also owns the method from the child class

* Lambda Function:

格式 (Example): MessageSender ms = name → method 的内容;
↑
"argument"
↓
这是单个的 argument 的

多个 argument 的: displaySome (db, pers → { method 内容 });

Generic Class 的限制 格式

Bounding:

* 在 Generic class header 中可以 supply any number of parameters: `public class Sample<T1, T2>{`

* 可以利用 extends / super 这两个 key word 去限制放进来的 object type

使用 { `<T extends RootClass>` : 只能放 RootClass 的 } upper bound
 `<T extends RootClass & Pizza>` : 只能放 RootClass 和 Pizza 的 } bound
 `<T super Integer>` : 只能放 Integer, Number, Object 的 → lower bound

⇒ bounding 之后, 也会确保在接下来的 method body 中, T 只有你 bound 中的 class 的 method

* 在写 inheritance class 时的一些注意事项:

* 看清楚题目的要求, 有时题目中并没有 clearly state 具体要写哪个 method, 但在后面 provide 的 page 中会看出来

* Method header 上面是 extend, 没有 s, 不论哪个 method header 都不要加 throw Exception, 下面的 constructor, mutator 的 header 中要加 throw Exception, 没有 s, 再下面的 throws new ("...") 中, 是有 s 的.

* Accessor 的 return type 决不是 void

@Override

* .equals → 返回一个 boolean; takes in (Object ~)
↳ instance of 的使用

* .compareTo → 返回一个 int; takes in (current object ~)

* .clone(), {

* 先创建一个空的 current object: Current lol = null;

* 一个 try/catch: try {

lol = new Current (~) } ^{→ constructor}

catch (Exception e) {

return null; }

return lol; }

* .toString() → 返回一个 String

极弱项: JavaFx

* Event-Driven: 通过硬件上的 event 来驱动程序, Event 可以有很多种

* To respond to an event — attach an Event Handler object to a component

* GUI construction 的进化:

AWT (Abstract Window Toolkit)



java.awt.Swing



JavaFX

考两年的问题, 为什么用 JavaFX 而不

是 Swing?

① Swing is deprecated now

② JavaFX nodes are fancier and perform better than Swing controls.

③ JavaFX has a larger set of features for media manipulation and playback than Swing does.

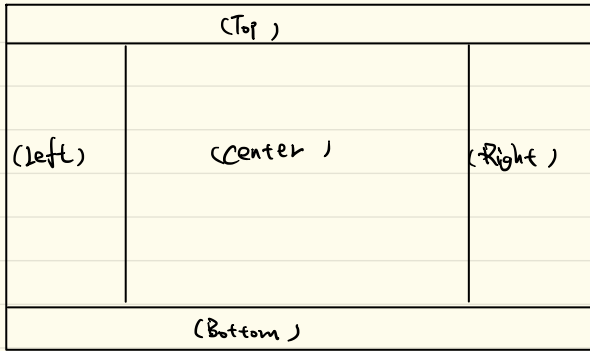
④ Swing was developed mostly for enterprise/business use, not for personal use and certainly not for mobile devices.

* .css file → Cascading style sheet, "skin" the scene using CSS style specification for the window. (Not have to have it)

* Pane object: a place to place your nodes

① Border Pane :

Layout 被分为五个
部分： →



Aside- Component



* Anchor Pane

- Component are anchored at a specified distance from the edge of the pane using the static methods :

```
static void setBottomAnchor(Node child, Double value) → in pixels as a distance from the border of the pane.  
static void setLeftAnchor(Node child, Double value) Sets the left anchor for the child when contained by an anchor pane.  
static void setRightAnchor(Node child, Double value) Sets the right anchor for the child when contained by an anchor pane.  
static void setTopAnchor(Node child, Double value) Sets the top anchor for the child when contained by an anchor pane.
```

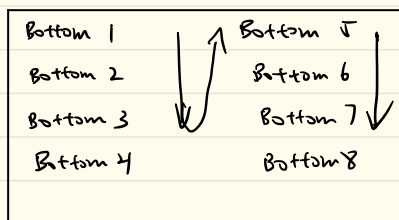
→ in pixels as a distance from the border of the pane.

- when changing size of the window, the node changes flexibly. ★★

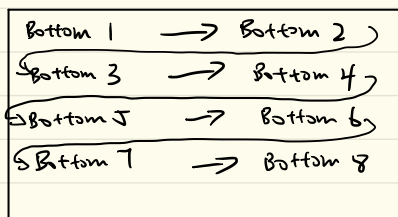
* Flow Pane

- Piles children into the pane in the order in which they are added from left to right (by default), if the available space is full, the children wrap around.
- when re-sizing the window, if the available space is full, the children wrap around.

* 如果 Flow Pane 是被设置成 Orientation. VERTICAL 的话, 它的 children 的排列是这样的



Orientation. HORIZONTAL 的话:



→ 网格

* Grid Pane → "Hardest one"

- You need to choose which position in the grid you'll use for your component
- 整列的宽度是由其中最宽的 component 来决定的, 而行的高度是由其中最高的 component 来决定的

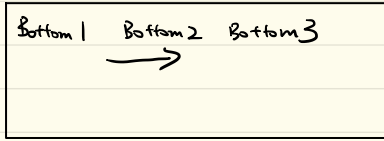
- Can add gaps between rows / columns using `.setHgap()` / `.setVgap()`, set gaps on the outside of the pane using `.setPadding()`

- Re-sizing → stable, not changing component size, relatively position maintain

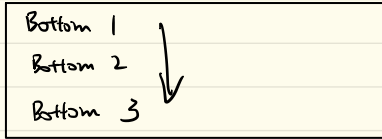
HBox and VBox panes

* Pretty straight forward, no wrapping, 有一点像是 Flow Pane, 都是通过一个存满了 label / bottom 的 ArrayList 把它们 pile up (Horizontally / vertically)

HBox:



VBox



No wrap around

* StackPane (doesn't need this one)

- Pile controls (nodes) on top of each other like a deck of cards

* TilePane

- Another grid based layout, but simpler than GridPane, more like FlowPane.

- Controls are laid down in the order in which they are added.

abstract class: public abstract class name () {

abstract method: public abstract return type class name (args) ;

* Generic class : public class name < T > {

Generic method:

public < T extends ... & ... & ... > return type class name (args) {
}

CISC 124

QUEEN'S UNIVERSITY
SCHOOL OF COMPUTING

HAND IN
Answers Are
Recorded on
Question Paper

CISC124, FALL TERM, 2017
FINAL EXAMINATION
9am to Noon, 8 DECEMBER 2017

Instructor: Alan McLeod

If the instructor is unavailable in the examination room and if doubt exists as to the interpretation of any problem, the candidate is urged to submit with the answer paper a clear statement of any assumptions made.

Proctors are unable to respond to queries about the interpretation of exam questions. Do your best to answer exam questions as written.

Please write your answers in the boxes provided. Extra space is available on the last page of the exam. The back of any page can be used for rough work. Please do not take the exam apart! This exam is three hours long and refers exclusively to the use of the Java language. Comments are not required in the code you write. For full marks, code must be efficient as well as correct.

This is a closed book exam. No computers or calculators are allowed.

Student Number:

20060593

Problem 1:	/ 30	Problem 4:	/ 50
Problem 2:	/ 10	Problem 5:	/ 10
Problem 3:	/ 20		

TOTAL:

/ 120

This material is copyrighted and is for the sole use of students registered in CISC124 and writing this exam. This material shall not be distributed or disseminated. Failure to abide by these conditions is a breach of copyright and may also constitute a breach of academic integrity under the University Senate's Academic Integrity Policy Statement.

Problem 1) [30 marks]

Mark each of the following as True or False using "T" or "F":

- F Java was initially designed to be used to construct stand-alone applications. ✓
- T Java is case-sensitive. ✓
- F A Java byte code file is already in machine language ready to be executed by the processor. ✓
- F Variables declared inside a method are available to any other methods in the same class. ✓
- F A variable declared outside a class is available to any class in the same package. ✓
- F Only static methods can be declared outside a class definition. ✓
- F Overloaded methods share the same name and parameter list, but differ by return type. ✓
- F A for/each loop can be used to re-assign array elements. ✓
- F Addition and subtraction have higher precedence than multiplication and division. ✓
- F The operator, %, provides floor division. ✓
- F { and } are used to control precedence in an expression. ✓
- T String objects are immutable. ✓
- F ArrayList<T> objects are immutable. ✗ ✓
- F Primitive type values are passed by reference into methods. ✓
- F An exception class must implement the Exception interface. ✓

Problem 1, Cont.)

- F Every instance of a class will have its own copy of a static method. ✓
- T Non-final attributes should be declared private. ✓
- F A method declared as "final" is designed to be overridden. ✓
- F Constructors cannot be overloaded. ✓
- F The return type of a constructor is void. ✓
- F A mutator should also return the value that is being assigned to an attribute. ✓
- F Only constructors can throw exceptions. ✓
- F A class header must contain a "throws" decoration if any of the methods in that class throw exceptions. ✓
- T An immutable class could have private mutators. ✓
- T For an equals method to override the Object.equals() method it must accept a parameter of type Object. ✓
- F For the compareTo method to override the Object.compareTo() method it must accept a parameter of type Object. ✗
- F The standard toString method is a void method. ✓
- F Private attributes and methods are also inherited from a parent class. ✓
- F The extends keyword is used in a class header to implement multiple interfaces. ✓
- F A class can extend more than one parent class. ✓

Problem 1, Cont.)

- F Optimal hierarchy construction encourages attribute re-declaration. ✓
- F Early binding is satisfied as a program is running. ✓
- T Polymorphism means that a base class type can be used as a parameter type for child class arguments. ✓
- F If class Child extends class Parent then ArrayList<Parent> can be a parent class for ArrayList<Child>. ✓
- T A well-designed inheritance structure will be extensible. ✓
- F Elements can be added to and removed from an Enumerated type after it has been declared. ✓
- F The type T in a class declared as GenericClass<T> can be instantiated inside the class. ✓
- F Primitive types can be supplied as types to generic classes and methods. ✓
- T A generic method can be invoked without supplying a type within < > because of type inference. ✓
- T A generic method can be invoked without using < > because of type inference. ✓
- F A generic class can only be typed with a single type. ✓
- T The use of a parameter of type Class<T> can be used to allow an instance of any Object type to be supplied for the parameter. T ✓
- F An interface that uses the @FunctionalInterface annotation can have one or two abstract methods only. ✓
- T A functional interface supplies the method signature for a lambda function. ✓
- T Any lambda function implementation can be replaced with an anonymous class implementation. ✓

Problem 1, Cont.)

- T A generic interface cannot also be a functional interface. **XF**
- F In Java versions 8 and 9 an interface can contain non-final attributes. ✓
- F A lambda function can only be written for method signatures that have a single parameter. ✓
- F All JavaFX GUI classes are taken from the javax.swing package. ✓
- T A stylesheet, a *.css file, can be used to modify the appearance of just a single node in a JavaFX window. ✓
- F A stylesheet can also be used to attach events to nodes like buttons. ✓
- T JavaFX contains node classes for charts like x-y plots. ✓
- F The child nodes of a FlowPane container will always maintain their relative positions even when the window using that pane is re-sized. ✓
- T When using a GridPane, nodes can be added to any row, column position in the pane in any order. ✓
- F A child node in an AnchorPane can only be anchored to one side of the pane at a time. ✓
- T The controller *.java class is identified in the Main.java class in a JavaFX application. **F**
- T The @FXML annotation is used in the controller *.java class to identify attributes that have fxid's and need to be injected. ✓
- T A font specified in the stylesheet file will override a font specified in the fxml file for the same node in a JavaFX project. ✓
- F Event listeners can only be added to nodes by using the fxml file. ✓
- T A radio button must be part of a ToggleGroup object in order for only one button in that group to be selected at a time. ✓

Problem 2) [10 marks]:

Answer the following in the space provided:

What is a "privacy leak"? Describe an example of when it would occur: *f*

Privacy leak happens when the contain that you don't want to shows turns out shows up, sometimes it happen when a pointer points to an private array

What is the process of "automatic boxing"? Provide a single example of its use:

when we need to use something in a Generic class, we don't need to instantiate it, the class itself will help you instantiate it.

Name two advantages of coding with inheritance:

① better structure

Why should all non-final class attributes be declared as "private"?

When we use non-final class attributes, the attribute is accessible for other people to change it, when we declared it as private, people who wants to change it gotta go through the mutator designed by designers which prevent the possible illegal input

Name two reasons why you would use JavaFX to build a GUI rather than swing:

Problem 3) [20 marks]

a) In the box provided below write a static method called "factorial" that calculates and returns the factorial of an integer value supplied as an argument. The factorial of a number, n , is shown in mathematical format as " $n!$ ". It is calculated using the cumulative product as:

$$n! = n * (n - 1) * (n - 2) * ... * 2 * 1$$

The easiest way to prevent numeric overflow of this calculation is to calculate and return the value as a double. If the factorial method is supplied with an argument that is less than or equal to 1, return 1.

```
public static double factorial (int supplied) {  
    result = 1 → int define as double  
    if (supplied > 0) {  
        for (i=1 ; i ≤ n ; i++) {  
            result = result * i  
        }  
        return result  
    }  
    else  
        system.out.println ("Not legal input")  
}
```

Problem 3, Cont.)

Type Casting

b) In the box provided below write a static method, to be included in the same class as the method written for part a), called "combinations". This method will be supplied with two integer arguments which represent the size of a set of "things", call it "all", as well as the size of a sub-set of "things" which could be called "some". The number of possible combinations of "some" when taken from "all" can be calculated using the formula:

$$\text{numCombinations} = \frac{\text{all}!}{\text{some!} (\text{all} - \text{some})!}$$

For example, there are six different combinations of two "things" when taken from a set of four "things". If the "things" are letters, you would have the set {A, B, C, D} for example. The possible combinations of two letters would be: {A, B}, {A, C}, {A, D}, {B, C}, {B, D} and {C, D}. So, all is 4, some is 2 and numCombinations would be 6.

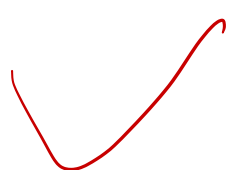
As you can see, this formula involves the calculation of factorials, so your method should use the method you wrote for part a). If either argument to combinations is less than 1 or if "some" is greater than or equal to "all", return 1. The combinations method should return a long type value. Do not worry about upper limits for the argument values.

```

public static long combinations (int some, int all) {
    if (some < 1 || all < 1) {
        return 1L;
    }
    else if (some >= all) {
        return 1L;
    }
    else {
        use = factorial(all) / factorial(some) * factorial(all - some);
        initial = (long) use;
        return initial;
    }
}

```

long initial;
double use;



Problem 3, Cont.) ✖

c) In the box provided on the next page write a static method, to be included in the same class as the methods written for parts a) and b), called “makeArray”. The integer argument supplied to makeArray will be the desired number of rows in the 2D array of type long that will be generated and returned by the method. The returned array will contain the possible combination results calculated using the method from part b). The row number will be the “all” argument and the column number will be the “some” argument. Since you cannot make a proper calculation of combinations for all array positions, the array must be a “ragged” array, where the number of columns in a row will equal the row index number plus one.

Here is an example that shows the contents and structure of the array that would be returned by makeArray if it was supplied the argument 12:

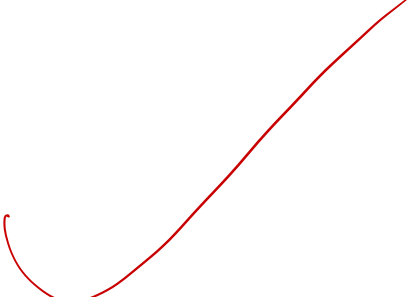
Col:	0	1	2	3	4	5	6	7	8	9	10	11
Row 0:	1											
Row 1:	1	1										
Row 2:	1	2	1									
Row 3:	1	3	3	1								
Row 4:	1	4	6	4	1							
Row 5:	1	5	10	10	5	1						
Row 6:	1	6	15	20	15	6	1					
Row 7:	1	7	21	35	35	21	7	1				
Row 8:	1	8	28	56	70	56	28	8	1			
Row 9:	1	9	36	84	126	126	84	36	9	1		
Row 10:	1	10	45	120	210	252	210	120	45	10	1	
Row 11:	1	11	55	165	330	462	462	330	165	55	11	1

Row and column numbers in the array have been added to the printout shown above for clarity – they are not part of the contents of the array. For simplicity, it is assumed that there is one way to obtain zero elements from any set, regardless of its size. A small box has been drawn around the result used as an example in part b) for the 2 letters taken from a set of 4 letters.

If makeArray is supplied with an argument that is less than 1, it can just return null.

Problem 3c, Cont.)

```
public static long[][] makeArray(int size) {  
    if (size < 1)  
        return null;  
    long[][] theArray = new long[size][size];  
    int row, column;  
    for (row = 0; row < theArray.length; row++)  
        for (column = 0; column < theArray[row].length;  
            column++)  
            theArray[row][column] = Combination(row, column);  
    return theArray;  
}
```



Problem 4) [50 marks]

For this problem you need to write four classes, called "IllegalOil", "Oil", "CookingOil" and "MotorOil".

IllegalOil will be the exception class used by the other three. IllegalOil only needs the one constructor that accepts a String type message.

Oil will be the base class for CookingOil and MotorOil. Here are the attributes that will be used in the hierarchy:

- Density, a float, must lie between 0.8 and 1.0 g/cc ("grams per cubic cm.") inclusive.
- Type, a String. It cannot be null or of zero length.
- Viscosity, a float, in Pa.s ("Pascal seconds"). For cooking oils this must lie between 0.01 and 0.1 inclusive. For motor oils this value must lie between 0.01 and 1.5 inclusive. Cooking oils only store a single value for viscosity, which would be the viscosity at room temperature. A motor oil will store viscosities in an array of float of any non-zero size, where the viscosities have been measured between 0 and 100 deg. C.
- Saturated Fat Content, an int. Only for cooking oils. A percent value that must lie between 0 and 100 inclusive.
- Grade, a String. Only for motor oils. It would look like "10W30". The number before the "W" (the winter grade) must be 0, 5, 10, 15 or 20. The number after the "W" (the summer grade) must be 8, 12, 16, 20, 30, 40 or 50.

All classes must be immutable. You may write any accessors that you need. Constructors must throw the exception object if any attempt is made to create an object with illegal argument(s).

You must also write equals, compareTo and clone methods. Equality, for the equals method, is defined as the Type string being the same, ignoring case. Comparison, for the compareTo method, is based only on density. Each concrete class must also have a clone() method.

The demonstration code on the next page illustrates the polymorphic behavior of the hierarchy and how constructors are invoked. When getViscosity() is invoked on a MotorOil object it returns the average viscosity calculated from the array of values in the object.

You may find some or all of the following methods useful:

```
aString.charAt(pos1) //returns the char at position pos1 in aString.
aString.length()     //returns the number of characters in aString.
aString.indexOf(searchString) //returns the location of searchString in
                               //aString, or -1 if it is not found.
aString.substring(pos1, pos2) //returns the sub-string from aString
                               //starting at location pos1 and going to
                               //location pos2-1.
Integer.parseInt(aString) //Attempts to convert aString to an int value,
                           //throwing a NumberFormatException if it cannot do
                           //so.
Character.isDigit(aChar)  //Returns true if aChar is a digit.
```

Problem 4, Cont.)

Demonstration code:

```
import java.util.ArrayList;

public class TestOils {

    public static void main(String[] args) {
        // Density units are g/cc and viscosity values are in Pa.s
        // Fake array values, used for all motor oils:
        float[] testV = {1.1F, 0.98F, 0.75F, 0.63F, 0.31F, 0.27F, 0.11F};
        ArrayList<Oil> db = new ArrayList<>();
        // All argument values are legal:
        try {
            //          density, type, viscosity, % saturated fats
            db.add(new CookingOil(0.911F, "Olive Oil", 0.034F, 14));
            db.add(new CookingOil(0.925F, "Coconut Oil", 0.080F, 92));
            db.add(new CookingOil(0.920F, "Peanut Oil", 0.04F, 18));
            //          density, type, viscosities, grade
            db.add(new MotorOil(0.888F, "Multi-grade", testV, "10W30"));
            db.add(new MotorOil(0.891F, "Multi-grade", testV, "5W16"));

        } catch (IllegalOil io) {
            System.out.println(io.getMessage());
        }
        for (Oil element : db) {
            System.out.println(element);
            System.out.println("Viscosity = " + String.format("%.3f",
                element.getViscosity()) + " Pa.s\n");
        }
    } // end main

} // end TestOils
/* OUTPUT:
Cooking oil, type: Olive Oil, density: 0.911 g/cc, 14% saturated fat content.
Viscosity = 0.034 Pa.s

Cooking oil, type: Coconut Oil, density: 0.925 g/cc, 92% saturated fat content.
Viscosity = 0.080 Pa.s

Cooking oil, type: Peanut Oil, density: 0.92 g/cc, 18% saturated fat content.
Viscosity = 0.040 Pa.s

Motor oil, type: Multi-grade, density: 0.888 g/cc, grade: 10W30.
Viscosity = 0.593 Pa.s

Motor oil, type: Multi-grade, density: 0.891 g/cc, grade: 5W16.
Viscosity = 0.593 Pa.s

*/
```

Problem 4, Cont.)

The exception class, IllegalOil:

```
public class IllegalOil extends Exception {  
  
    public IllegalOil (String message) {  
        super(message)  
    }  
  
}
```

Handwritten notes: The word "extends" is written in red above the code. A red circle is drawn around the word "extends" in the code, with a red arrow pointing from the handwritten "extends" to it.

Problem 4, Cont.)

The base class, Oil:

→ 不要加!

```
public class Oil throw IllegalOil {  
    private float density;  
    private String type;  
  
    public Oil (float den, String typ) {  
        setDensity (den);  
        setType (typ);  
    }  
  
    private void setDensity (float aNum) throw IllegalOil {  
        if ((aNum < 0.8) || (aNum > 1.0))  
            throws new IllegalOil ("Your input is not legal");  
        else  
            this.density = aNum;  
    }  
  
    private void setType (String aStr) throw IllegalOil {  
        if ((aStr == null) || (aStr.length == 0))  
            throws new IllegalOil ("Your input is not legal");  
        else  
            this.type = aStr;  
    }  
}
```

Problem 4, Cont.)

The base class, Oil, continued:

accessor 的 return type 绝不是 void

```
public void getType() {
    return this.type;
}
```

```
public void getDensity() {
    return this.density;
}
```

@ Override

```
public boolean equals (Obj anObject) {
    if (anObject instanceof Oil) {
        if (anObject.getType().equalsIgnoreCase(this.type))
            return true;
        else
            return false;
    }
    else
        return false;
}
```

@ Override

```
public int compareTo (Oil otherOil) {
    int returnVal;
    if (Math.abs(otherOil.getDensity() - this.Density) < 0)
        return 0;
    else
        returnVal = (int)otherOil.getDensity() - this.Density;
    return returnVal;
}
```

⇒ 可以写得简单点的, 既

- > return 1
- < return -1
- = return 0

Problem 4, Cont.)

The CookingOil class:

```

public class CookingOil extends Oil throw IllegalOil {
    private float viscosity;
    private int fat;

    public CookingOil (float den, string typ, float vis,
                      int fa) throw IllegalOil {
        super (den, typ);
        setVis (vis);
        setFat (fa);
    }

```

```

    private void setVis (float aNum) throw IllegalOil {
        if ((aNum < 0.01) || (aNum > 1.5))
            throws new IllegalOil ("Not legal");
        else
            this. viscosity = aNum;
    }

```

```

    private void setFat (int aNum) throw IllegalOil {
        if ((aNum < 0) || (aNum > 100))
            throws new IllegalOil ("Not legal");
        else
            this. fat = aNum;
    }

```

@ override

```

public CookingOil clone () {
    CookingOil aNewone = null;
    try {
        aNewone = new CookingOil (density, type, viscosity, fat);
    } catch (IllegalOil e) {
        return null;
    }
}

```

Problem 4, Cont.)

The MotorOil class:

```
public class MotorOil extends Oil {  
    private float[] viscosityArray;  
    private String grade;  
  
    public MotorOil (float den, String typ, float[] visA, String gra)  
        throw IllegalOil {  
  
        super (den, typ);  
        setVisArray (visA);  
        setGrade (gra);  
  
    }  
  
    private void setVisArray (float[] anArray) throw IllegalOil {  
        if (anArray.length == 0)  
            throw new IllegalOil ("Not legal");  
  
        for (i=0; i < anArray.length; i++) {  
            if ((anArray[i] < 0.01) || (anArray[i] > 1.5))  
                throw new IllegalOil ("Not legal");  
        }  
  
        this.viscosityArray = anArray;  
    }  
}
```

Problem 4, Cont.)

The MotorOil class, continued:

```

private void setGrade (String astr) throws Illegal Oil {
    int pos ;
    String pre, post;
    pos = astr.indexOf ("r");
    if (pos == -1 )
        throw new Illegal Oil ("Not legal");
    pre = astr.substring (0, pos);
    post = astr.substring (pos, astr.length);
    if (! pre.equalsIgnoreCase ("O")) {}
    ~
    ~
    ~
}

```

@Override

```

public Motor Oil clone () {

```

```

    Motor Oil anowone = null;

```

```

    try {

```

```

        anowone = new (density, type, viscosityArray,
                        grade);

```

```

    catch (Illegal Oil e)

```

```

        return null;

```

```

    }

```

```

}

```


Problem 5) [10 marks]

Here is a complete JavaFX program, starting below and ending on the next page:

```
package application;

import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;

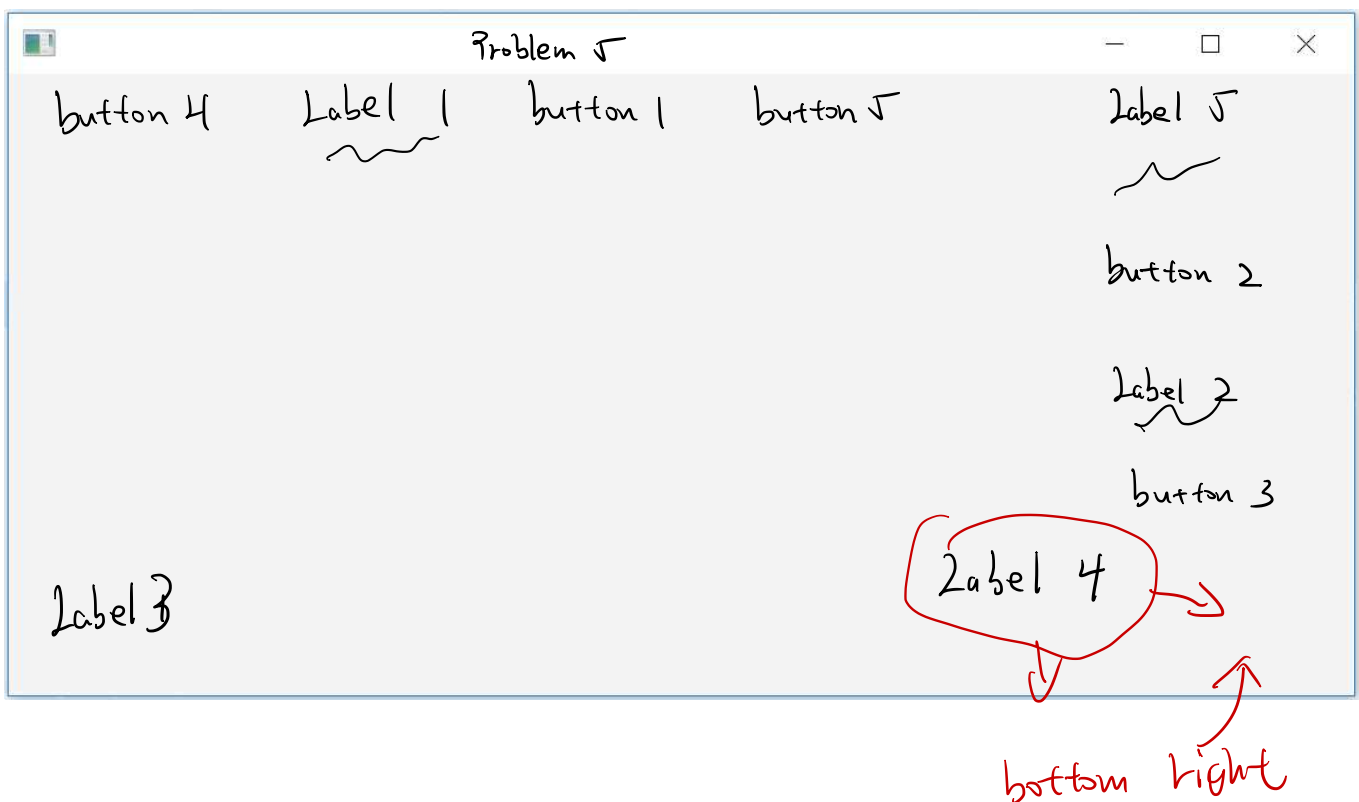
public class Main extends Application {
    @Override
    public void start(Stage primaryStage) {
        try {
            AnchorPane root = new AnchorPane(); ✓
            HBox hPane = new HBox(10);
            VBox vPane = new VBox(10);
            Button button1 = new Button("Button 1");
            Button button2 = new Button("Button 2");
            Button button3 = new Button("Button 3");
            Button button4 = new Button("Button 4");
            Button button5 = new Button("Button 5");
            Label label1 = new Label("Label 1");
            Label label2 = new Label("Label 2");
            Label label3 = new Label("Label 3");
            Label label4 = new Label("Label 4");
            Label label5 = new Label("Label 5");
            hPane.getChildren().addAll(button4, label1, button1, button5); ✓
            vPane.getChildren().addAll(label5, button2, label2, button3); ✓
            AnchorPane.setTopAnchor(hPane, 10.0);
            AnchorPane.setRightAnchor(vPane, 10.0);
            AnchorPane.setBottomAnchor(label3, 10.0);
            AnchorPane.setLeftAnchor(label3, 10.0);
            AnchorPane.setBottomAnchor(label4, 10.0);
            AnchorPane.setRightAnchor(label4, 10.0);
            root.getChildren().addAll(hPane, vPane, label3, label4);
            Scene scene = new Scene(root, 600, 300);
            scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());
            primaryStage.setScene(scene);
            primaryStage.setTitle("Problem 5");
            primaryStage.show();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
} // end start
```

```
public static void main(String[] args) {  
    Launch(args);  
} // end main  
  
} // end Main.java
```

Problem 5, Cont.)

a) Using the empty window shown below, sketch the appearance of this window as it appears when run. Draw a button as text inside a drawn rectangle and a label as just text. Don't worry about imitating fonts or even drawing straight lines. Concentrate on getting the relative positions of buttons and labels and their associated text correct. Note that the pane objects used do not display their borders. A ten pixel gap would be about 2mm on this drawing.

You are not given the contents of the stylesheet, but you don't need to see this stuff – the contents of this file will not affect the layout of the nodes and the text they contain.



Problem 5, Cont.)

b) A Label owns a method called `setText(String arg)` that can be used to change the text displayed in a Label to the String contained in `arg`. A Button owns a method called `setOnAction(EventHandler<ActionEvent> arg)` that can be used to attach an event listener to a Button. The functional interface `EventHandler<ActionEvent>` contains a single abstract method called "handle" with the following signature:

```
void handle(ActionEvent event);
```

In the box below, write code that uses a lambda function to add an event listener to the `button1` node that changes the text in all of the labels in the VBox and HBox panes to an empty String.

Write just the code that you would add to the code shown above. Do not repeat any of the code shown above.

```
button1.setOnAction(event -> {  
    label1.setText("");  
    label5.setText("");  
    label2.setText("");  
    } );
```

(*extra page*)

CISC 124

QUEEN'S UNIVERSITY
SCHOOL OF COMPUTING

HAND IN
Answers Are
Recorded on
Question Paper

CISC124, FALL TERM, 2016
FINAL EXAMINATION
7pm to 10pm, 19 DECEMBER 2016

Instructor: Alan McLeod

If the instructor is unavailable in the examination room and if doubt exists as to the interpretation of any problem, the candidate is urged to submit with the answer paper a clear statement of any assumptions made.

Proctors are unable to respond to queries about the interpretation of exam questions. Do your best to answer exam questions as written.

Please write your answers in the boxes provided. Extra space is available on the last page of the exam. The back of any page can be used for rough work. Please do not take the exam apart! This exam is three hours long and refers exclusively to the use of the Java language. Comments are not required in the code you write. For full marks, code must be efficient as well as correct.

This is a closed book exam. No computers or calculators are allowed.

Student Number:

Problem 1:	/ 30	Problem 4:	/ 40
Problem 2:	/ 10	Problem 5:	/ 10
Problem 3:	/ 20	Problem 6:	/ 10

TOTAL:

/ 120

This material is copyrighted and is for the sole use of students registered in CISC124 and writing this exam. This material shall not be distributed or disseminated. Failure to abide by these conditions is a breach of copyright and may also constitute a breach of academic integrity under the University Senate's Academic Integrity Policy Statement.

Problem 1) [30 marks]

Mark each of the following as True or False using "T" or "F":

- T A "for each" loop can visit every element in an array. ✓
- F Normal for loops cannot be used with ArrayList<T> collections. ✓
- F A "chained if" construct uses a switch statement. ✓
- T Java switch statements can be used to compare Strings for equality. ✓
- F The Boolean comparators < and > can be used in the case statements inside a switch statement construct. ✓
- T A counter variable declared inside a for loop statement is scoped within the loop only. ✓
- F Variables can be declared outside a class. ✓
- F A static variable declared inside a method is known to all other methods in the same class. ✓
- T A method declared as "final" cannot be overridden. ✓
- F A class is declared "final" if it contains one or more abstract methods. ✓
- F A static method declared in another class can be invoked without naming that other class. ✓
- T The main method can invoke static methods in the same class directly without needing to instantiate the class that it belongs to. ✓
- F A method's parameter list can contain default arguments. ✓
- F A method that does not return anything, like main, does not need to state any return type. ✓
- F A method declared "private" can be invoked in another class from an instance of the class that owns that method. ✓

Problem 1, Cont.)

- F An abstract method is declared using an empty set of {} (curly braces). ✓
- T A child class that extends an abstract parent class must implement all inherited abstract methods unless it is to be abstract itself. ✓
- F An abstract class can only contain abstract methods. ✓
- F Abstract methods cannot have a return type. ✓
- F An abstract class cannot be instantiated. ✗ T
- F An abstract class must have an empty, default constructor. ✓
- F An interface can contain non-final attributes. ✓
- T A class can implement more than one interface. ✓
- F A child class can extend more than one parent class. ✓
- T An interface can be used as a type when declaring variables. ✓
- T A child class' constructor must invoke the parent class' constructor as the first line of code in the child class' constructor. ✓
- F Constructors cannot be overloaded in the same class. ✓
- F The return type of a constructor is "void". ✗ ✓
- T Polymorphism is also referred to as "dynamic binding". ✓
- F A class cannot extend another class and implement an interface at the same time. ✓

Problem 1, Cont.)

- T In order to override the equals method inherited from Object, an equals method should accept an Object as a parameter. ✓
- F The compareTo method should return a Boolean true/false result. ✓
- F The toString method in an object is automatically invoked whenever a String representation of the object is required. ✗ T
- F The clone method works properly when invoked from a pointer of type 2D array. ✓
- F An immutable object contains public mutators for every private attribute. ✓
- T An ArrayList<T> collection is a mutable object. ✗
- F If Number is a parent class for Double, then ArrayList<Number> can be a parent class type for objects of type ArrayList<Double>. ✓
- F Collection<?> can be a base class for any object of type ArrayList<T>. ✗ T
- F Generic methods must be contained in generic classes. ✓
- F The use of syntax like "<T extends MyClass>" in the context of a generic method or class ensures that objects of type T will have access to methods declared in MyClass. ✗ T
- F The super keyword is used to supply a reference to the current object from inside an instance of that object. ✓
- T An ArrayList<Double> object can be supplied to a parameter of type ArrayList<? extends Number> when invoking a generic method. ✗
- T A generic class can be declared to use more than one generic type. ✓
- F The Comparable<T> interface specifies the method header for the equals method. ✗
- F A "Functional Interface" used with lambda functions can contain more than one abstract method. ✓

Problem 1, Cont.)

- T JavaFX requires the use of a *.fxml file for component layout and sizing. ~~XF~~
- F The stylesheet, or *.css file, can be used to attach an event listener to a JavaFX node. ✓
- F An event listener must be added to a JavaFX node each time the user clicks on it. ✗
- T A stylesheet, or *.css file, can be used to configure the appearance of all nodes of a certain type or just a single node of that type. ✓
- ~~F~~ T A change listener will only respond when a user clicks on a component. ✗
- F All event listener types will only respond to mouse actions such as button clicks. ✓
- T The "@FXML" annotation is used in a controller file to identify pieces of Java code that will need to be injected after the fxml file is processed. ✓
- T Nodes can be created and named in the fxml file and those names are available in the Java controller file, provided the fxml file knows the name of the controller file. ✓
- F Fonts and font sizes can only be assigned in the fxml file. ✓
- ~~T~~ F An EventListener interface is a Functional Interface. ✗
- T An EventListener can be implemented using a lambda function or an anonymous class. ✓
- ~~T~~ F A TextField node can be configured so that the user cannot change the text shown in the node. ✗
- T The Pane class does not contain a layout manager algorithm, but uses absolute sizing and positioning instead. ✓
- T The BorderPane class breaks a region into five parts. ✗ ✓
- ~~T~~ F Nodes in a FlowPane can change their relative positions if the pane's size changes. ✗

Problem 2) [10 marks]:

Answer the following in the space provided:

What is the process of “type inference” in the context of the use of generic methods and classes?

How would you bound a generic type when declaring a generic class or method?

What is the advantage of using a generic method over a method that is typed to use Objects instead?

Name one advantage of using JavaFX over Swing for GUI construction.

What is the process of “early binding” and when does it happen?

Problem 3) [20 marks]*2nd time without j*

a) Write a static method called "gcd" that accepts two int values as arguments and then returns the number that is the greatest common divisor of the two supplied numbers. A divisor is a number that divides another number without any remainder. The greatest common divisor or "GCD" is the largest int value that divides two numbers evenly. For example, if the gcd method was supplied with the values 10 and 25 it would return 5. If it was supplied with 43 and 100 it would return 1. If it was supplied with 10 and 100 it would return 10. You can assume that the method will only be supplied with integer values that are greater than zero.

```
public static int gcd (int aNum, int bNum) {  
    int count, divisor;  
    if (aNum < bNum)  
        count = aNum;  
    else  
        count = bNum; 2 =  
    for (i = 1; i ≤ count; i++) {  
        if (aNum % i == 0 && bNum % i == 0)  
            divisor = i;  
    }  
    return divisor;  
}
```

Problem 3, Cont)

b) Write a static method called "gcdArray" that accepts two 1D arrays of type int as arguments. If either array is empty (of zero size), null, or if they have different sizes, return null. Otherwise this method will return an array of the greatest common divisors of the two numbers from each array in the equivalent index positions. Use the gcd method you wrote for Part a). If either number in the same index location is less than one, skip that index position. This means that the returned array will be smaller than the supplied arrays if any of the values in either array is less than one.

For example, if the gcdArray method was supplied with the two arrays:

```
{4, -1, 10, 30, 27, 0, 100, 55}
{0, 10, 30, 6, 18, 20, 32, -2}
```

it would return the array:

```
{10, 6, 9, 4}
```

```
public static int[] gcdArray (int[] a, int[] b) {
    int[] divisorArray = new int [a.length];
    int count = 0;

    if ( (a.length == 0) || (b.length == 0) || (a == null) || (b == null)
        || (!(a.length == b.length)) )
        return null;

    for (i = 0; i < a.length; i++)
        if ((a[i] == 0) || (b[i] == 0))
            continue;
        else {
            divisorArray[count] = gcd(a[i], b[i]);
            count++;
        }

    return divisorArray;
}
```

Problem 4) [40 marks]

For this problem you need to write four classes, called “Battery”, “SingleUse”, “Rechargeable” and “BadBattery”. Both SingleUse and Rechargeable extend Battery. BadBattery is an exception class used by the other three.

A SingleUse object is described by the following attributes:

- name – The name of the battery as a String, such as “Alkaline D”. The string cannot be empty.
- voltage – The nominal operating voltage of the battery in volts as a double, which would be 1.5 V for the battery named above, for example. The voltage must be greater than zero and less than 50.
- cost – The estimated cost of the battery in units of \$ per kW-hr as a double. This value is estimated to be \$100 per kW-hr (“kilowatt hour”), for the Alkaline D battery, for example. The value must lie between 1 and 30,000.
- capacity – The estimated battery capacity in units of AH or “Amp Hours” as a double. For example a capacity of 10 AH means that a battery could deliver 10 amps for one hour or 1 amp for 10 hours before running dry. An Alkaline D battery has an estimated capacity of 8 AH, for example. The value must lie between 0.001 and 500.

A Rechargeable object is described by all of the above attributes plus:

- numCharges – An estimate of the number of possible charge cycles of the battery as an int. For example, it is estimated that a NiCad rechargeable battery can be recharged 500 times. This value must lie between 2 and 10,000.

All attributes must be declared private. The base Battery class must contain only the name, voltage and cost attributes which should not be re-declared in either of the child classes.

On the next page is some code in a main method in some other class that uses these four classes, along with the output of this code. You can see that polymorphism is used to invoke the toString and getLifetimeCost methods. The sample output shows the format of the String returned by the toString method. The getLifetimeCost method estimates and returns the per use cost of the battery over the entire lifetime of the battery. This is defined as:

$$\text{cost} * \text{voltage} * \text{capacity} / (1000 * \text{numUses})$$

The “1000” is to convert Watt-hours to kWatt-hours. This is probably a pretty *bogus* definition, but it is simple and this is the calculation that you must code into your implementations of the getLifetimeCost method. Of course, numUses is one for a SingleUse object.

You can see that the base class Battery cannot implement the getLifetimeCost method, but it must contain this method in some form in order for polymorphism to work. Write the exception class first followed by Battery and then the other two. No other classes are required. Write the just the minimum set of methods needed. All classes are to be immutable.

Problem 4, Cont.)

Demonstration code:

```
import java.util.ArrayList;

public class TestBatteries {

    public static void main(String[] args) {

        ArrayList<Battery> bats = new ArrayList<>();

        try {
            bats.add(new SingleUse("Alkaline D", 1.5, 100, 8.0));
            bats.add(new SingleUse("Alkaline 9V", 9.0, 600, 0.4));
            bats.add(new SingleUse("Silver Oxide Button", 1.55, 18000, 0.15));
            bats.add(new Rechargeable("Lead Acid", 2.1, 100, 225, 500));
            bats.add(new Rechargeable("NiCad", 1.2, 1000, 1.0, 500));
            bats.add(new Rechargeable("LiPo", 3.2, 350, 2.2, 1000));
        } catch (BadBattery e) {
            // All battery data used above is legal.
            System.err.println("Should not get here...");
        }

        for (Battery bat : bats) {
            System.out.print(bat);
            System.out.printf(", cost per use: $%.3f\n", bat.getLifetimeCost());
        }

    } // end main

} // end TestBatteries
/* OUTPUT:
Alkaline D single use battery, cost per use: $1.200
Alkaline 9V single use battery, cost per use: $2.160
Silver Oxide Button single use battery, cost per use: $4.185
Lead Acid rechargeable battery, cost per use: $0.095
NiCad rechargeable battery, cost per use: $0.002
LiPo rechargeable battery, cost per use: $0.002
*/
```

Problem 4, Cont.)

The exception class, BadBattery:

```
public class BadBattery extends Exception {  
    public BadBattery (String message) {  
        super (message);  
    }  
}
```

Problem 4, Cont.)

The base class, Battery:

```
public abstract class Battery {  
    private String name;  
    private double voltage;  
    private double cost;  
  
    public Battery (String nam, double vol, double cos) throw BadBattery {  
        if ((nam == null) || (nam.length == 0))  
            throw new BadBattery ("Not legal");  
  
        this.name = nam;  
  
        if ((vol < 0) || (vol > 50))  
            throw new BadBattery ("Not legal");  
  
        this.voltage = vol;  
  
        if ((cos < 1) || (cos > 30000))  
            throw new BadBattery ("Not legal");  
  
        this.cost = cos;  
    }  
  
    public String getName () {  
        return this.name;  
    }  
  
    public double getCost () {  
        return this.cost;  
    }  
  
    public abstract double getLifetime () ;  
}
```


Problem 4, Cont.)

The SingleUse class:

```

public class SingleUse extend Battery {
    private double capacity;

    public SingleUse (String nam, double vol, double cos, double cap) throw
        BadBattery
    {
        super(nam, vol, cos);
        if ((cap < 0.001) || (cap > 500))
            throw new BadBattery("Not legal");
        this.capacity = cap;
    }

    public double getCapacity() {
        return this.capacity;
    }

    @Override
    public double getLifetimeCost() {
        return ((this.getCosts() * (this.getVoltage() * this.capacity)
            / 1000) * 1

    }
}

```

Problem 4, Cont.)

The Rechargeable class:

Problem 5) [10 marks]

For this problem you will need to write a single generic method called "getLowCost" that would be included in the TestBatteries class shown on page 10 in Problem 4.

The getLowCost method accepts two arguments – an ArrayList collection and a double value. The ArrayList collection could be of type ArrayList<Battery>, ArrayList<SingleUse> or ArrayList<Rechargeable>, where the Battery, SingleUse and Rechargeable classes are described in Problem 4. The getLowCost method returns an ArrayList collection with the same element type as the one supplied as an argument with just those battery objects whose per use cost is less than or equal to the double value supplied to the method as the second argument. The per use cost is the value returned by the getLifetimeCost method described in Problem 4.

The generic getLowCost method:

```

public static <T extends Battery & SingleUse & Rechargeable>
ArrayList<T> getLowCost ( ArrayList<T> array1, double a ) {
    ArrayList<T> returnList = new ArrayList<T> ();
    int count = 0;
    for ( T obj : array1 ) {
        if ( obj.getCost() <= a )
            returnList[count] = obj;
            count++;
    }
    return returnList;
}

```

ArrayList 加东西是
arrayList.add()

Problem 6) [10 marks]

Here is a complete JavaFX program:

```
package application;

import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;

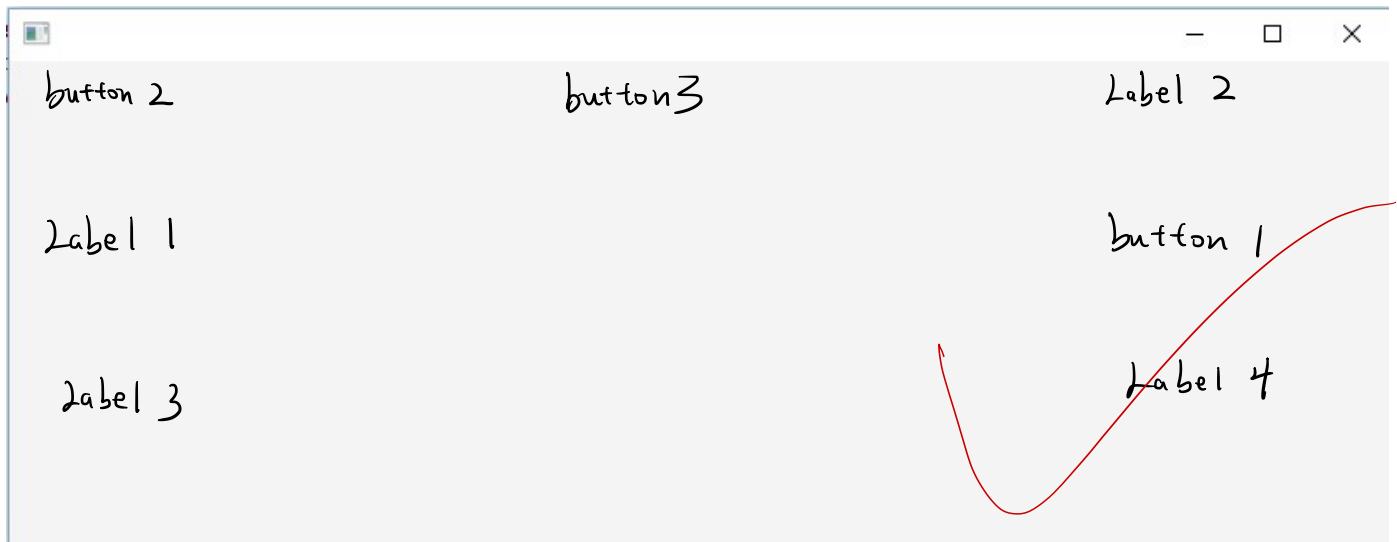
public class Main extends Application {
    @Override
    public void start(Stage primaryStage) {
        try {
            HBox root = new HBox(10);    // A 10 pixel gap
            VBox box1 = new VBox(10);
            VBox box2 = new VBox(10);
            VBox box3 = new VBox(10);
            Button button1 = new Button("Button 1");
            Button button2 = new Button("Button 2");
            Button button3 = new Button("Button 3");
            Label label1 = new Label("Label 1");
            Label label2 = new Label("Label 2");
            Label label3 = new Label("Label 3");
            Label label4 = new Label("Label 4");
            box1.getChildren().addAll(label2, button1, label4);
            box2.getChildren().addAll(button2, label1, label3);
            box3.getChildren().add(button3);
            root.getChildren().addAll(box2, box3, box1);
            Scene scene = new Scene(root, 400, 140);
            // Sorry about the formatting for this line, but now it fits...
            scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());
            primaryStage.setScene(scene);
            primaryStage.setTitle("Problem 6");
            primaryStage.show();
        } catch (Exception e) {
            e.printStackTrace();
        }
    } // end start method

    public static void main(String[] args) {
        launch(args);
    } // end main method
} // end Main class
```

Problem 6, Cont.)

a) Using the empty window shown below, sketch the appearance of this window as it first appears. Draw a button as text inside a drawn rectangle and a label as just text. Don't worry about imitating fonts or even drawing straight lines. Concentrate on getting the relative positions of buttons and labels and their associated text correct. Note that the pane objects used do not display their borders.

You are not given the contents of the stylesheet, but you don't need to see this stuff – the contents of this file will not affect the layout of the nodes and the text they contain.



b) A Label owns a method called `setText(String arg)` that can be used to change the text displayed in a Label to the String contained in `arg`. A Button owns a method called `setOnAction(EventHandler<ActionEvent> arg)` that can be used to attach an event listener to a Button. The interface `EventHandler<ActionEvent>` contains a single abstract method called "handle" with the following signature:

```
void handle(ActionEvent event);
```

In the box on the next page, write code that uses a lambda function to add an event listener to the `button3` node that changes the text in `label1` to "Label One" and the text in `label2` to "Label Two" when the user clicks on `button3`.

Write just the code that you would add to the code shown on the previous page. Do not repeat any of the code shown on the previous page.

c) Finally, draw an arrow in the previous page showing where you would add this code to the provided code.

Problem 6, Cont.)

The lambda function code needed to add the action listener to button3:

(*extra page*)

CISC 124

QUEEN'S UNIVERSITY
SCHOOL OF COMPUTING

HAND IN
Answers Are
Recorded on
Question Paper

CISC124, FALL TERM, 2015
FINAL EXAMINATION
7pm to 10pm, 15 DECEMBER 2015

Instructor: Alan McLeod

If the instructor is unavailable in the examination room and if doubt exists as to the interpretation of any problem, the candidate is urged to submit with the answer paper a clear statement of any assumptions made.

Proctors are unable to respond to queries about the interpretation of exam questions. Do your best to answer exam questions as written.

Please write your answers in the boxes provided. Extra space is available on the last page of the exam. The back of any page can be used for rough work. Please do not take the exam apart! This exam is three hours long and refers exclusively to the use of the Java language. Comments are not required in the code you write. For full marks, code must be efficient as well as correct.

This is a closed book exam. No computers or calculators are allowed.

Student Number:

Problem 1:	/ 30	Problem 5:	/ 10
Problem 2:	/ 20	Problem 6:	/ 30
Problem 3:	/ 10	Problem 7:	/ 10
Problem 4:	/ 10		

TOTAL: / 120

This material is copyrighted and is for the sole use of students registered in CISC124 and writing this exam. This material shall not be distributed or disseminated. Failure to abide by these conditions is a breach of copyright and may also constitute a breach of academic integrity under the University Senate's Academic Integrity Policy Statement.

-7 → 53/60 = 0.88

Problem 1) [30 marks]

Mark each of the following as True or False using "T" or "F":

- T Compilation of Java source code produces an intermediate byte code file which is then processed to produce an executable file. F
- F Every IDE uses its own unique Java compiler. ✓
- T Methods can only be declared within classes. ✓
- F Attributes can be declared outside classes. ✓
- T Static methods can be invoked without instantiating the class that owns them. ✓
- F Every class must be declared to contain at least one attribute. ✓
- F Switch statements can work with Strings as well as integer and floating point types. ✓
- T A Java class can only extend one class. ✓
- T Every class in Java is a sub-class of the Object class. ✓
- F An interface can only contain abstract methods and final static attributes. (T)
- F Both public and private methods are inherited by a child class from a parent class as long as it is in the same package as the parent class. ✓
- F A class with a parameterized constructor will no longer have a generated default constructor. T
- F Only constructors can throw exceptions. ✓
- F A class that contains constructors that throw exceptions must have a throws decoration in the class declaration line. ✓
- F An abstract class can contain normal attributes and only abstract methods. ✓

Problem 1, Cont.)

- T A child class can implement many interfaces. ✓
- T A sub-class that extends an abstract class must have a non-abstract implementation of every abstract method in the super-class, unless the sub-class is abstract as well. ✓
- T The first line of code in a child class' constructor must be a call to the parent class' constructor. ✓
- F Abstract classes can be instantiated. ✓
- F Interfaces cannot be instantiated. ✓
- T A variable can be declared to be of an interface type. ✓
- T A variable of an abstract class type can be assigned to point to an object of a concrete child class type that extends the abstract class. ✓
- T A variable of type Object can point to any object in Java. ✓
- T Early binding is satisfied at the time of compilation. ✓
- T Late binding is the process that provides polymorphism at run time. ✓
- T Non-final attributes should always be declared private. ✓
- F A mutator returns the value of the attribute it is changing. ✓
- F A standard clone() method throws an exception if a valid clone object cannot be created. ✓
- T For the equals() method to override the equals method inherited from the Object class it must accept a single Object type argument. ✓
- F A standard compareTo() method returns a boolean value. ✓

Problem 1, Cont.)

- T An overridden method refines the method it is overriding by invoking it. ✓
- F A child class can overload a method inherited from a parent class provided the name of the method is the same, the return type is the same and the parameter list is the same. ✓
- F Classes cannot contain both static and non-static methods declared in the same scope. ✓
- F A static method cannot use non-static attributes. T ✓
- T An ArrayList<T> object holds objects of type T. ✓
- T A generic class is also known as a "parameterized class". ✓
- T Array types can be provided as types to generic classes. ✓
- T An interface type can be provided as a type to generic classes and methods. ✓
- T The "< >" in a generic type can hold more than one type. ✓
- F Generic methods can enforce type safety where methods that accept and return Objects cannot. ✗ T
- T Type inference can be used with generic classes and methods. ✓
- F GUI programs can only respond to mouse events such as mouse button clicks and cursor movements. ✓
- T An EventHandler object must be attached to a JavaFX component in order for it to respond to an event. ✓
- F "AWT" stands for "Awful Windows Toolkit". ✓
- T The Swing GUI toolkit has been replaced by JavaFX. ✓

Problem 1, Cont.)

- T JavaFX contains a control that can act as a web browser. ✓
- T A JavaFX project can use separate stylesheet and fxml files along with normal Java source code files. ✓
- F Stylesheets, or *.css files, are used to create and position nodes belonging to a Scene Graph. ✓
- F Fxml files are stored in binary format. ✓
- T A single setting in a stylesheet can be used to alter the look of all controls belonging to a Scene Graph. ✓
- T A stylesheet can be used to alter the look of just one control at a time. ✓
- T An fxml file is designed to inject nodes into a *.java controller file when it is being instantiated. ✓
- T The use of nodes in an *.java controller class is indicated with the "@FXML" annotation. ✓
- F The controller class associated with an fxml file must instantiate the nodes used by a particular scene object. ✓
- F A scene object must be added to the stage object supplied to the overridden start method as an argument in a JavaFX application. T
- T The root object of a scene object is usually an instance of a class that extends the Pane class. ✓
- T The BorderPane object breaks up a region into five areas that can be used to contain nodes. ✓
- F A FlowPane object maintains the relative positions of its nodes independent of the size of the pane. ✓
- F JavaFX does not support animation. ✓
- T JavaFX does support drawing with 3D objects. ✓

Problem 2) [20 marks]:

Answer the following questions as briefly as possible:

Can you store an int value in an ArrayList<T> object and if so, how would you do it?

What is the purpose of implementing the Comparable<T> interface?

What is the reason for making non-final attributes private?

What is the advantage to declaring class members static?

What is one advantage of using polymorphism?

Problem 2, Cont.)

What does it mean to say that an “exception is propagated”?

Name one advantage of using inheritance.

Why is a try-with-resources block better than a normal try-catch block when used with File I/O?

Which file format – binary or text – produces the most compact file for storing numeric data and why?

What is a Scene Graph as used in JavaFX?

Problem 3) [10 marks]

Provide the console window output of each of the following println statements. If you think the statement will cause an error, write “error” instead.

System.out.println (15 / 2.0);	
System.out.println ((double)(7 / 2));	
System.out.println (6 * (2 / 3) + 7);	
System.out.println (true && 1);	
System.out.println (6 * 2 / 3.0 + 7);	
System.out.println (6 * 20 / (3 + 7));	
System.out.println (5 > 2 3 <= 1);	
System.out.println (2 ** 3);	
System.out.println (26 % 3);	
System.out.println (4 + 6 * 2 - 7 + 10 / 5 - 1);	
System.out.println (true && 9 != 7);	
System.out.println ("4" + 3 + 1);	
System.out.println (2 + 1 + "7" - 4);	
System.out.println (5 > 2 && 6 != 5 7 < 3);	
System.out.println (6 + 7 <> 13);	
System.out.println (2.4 / (int) 1.2);	
System.out.println ((int) (9.6 / 2.0));	
System.out.println (7 + 2 > 3 && 9 >= 2 + 2 + 2);	
System.out.println (Character.isDigit("123"));	
System.out.println (Character.toLowerCase('B'));	

Problem 4) [10 marks]

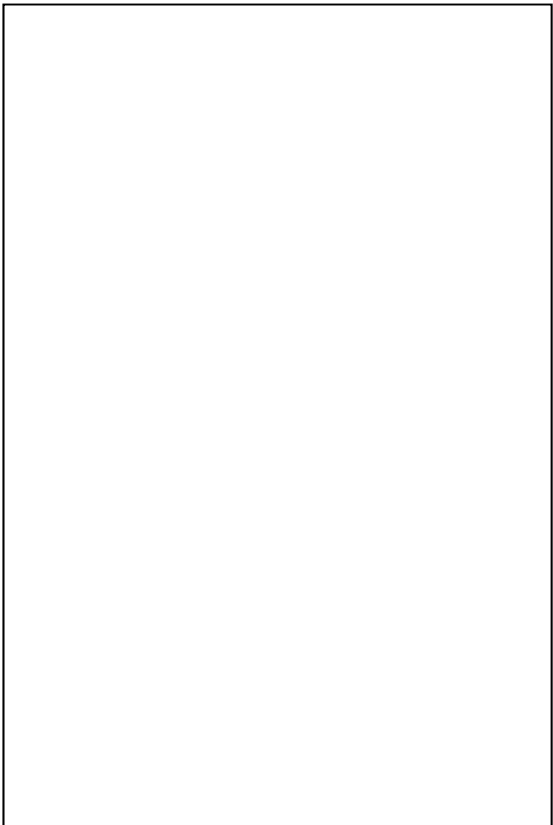
Write the output of the following complete program, which runs without error, in the box provided:

```
public class Problem1 {

    public static double fiddle(int num1, int[] nums1, int[] nums2, String str) {
        str = str.toLowerCase();
        int size = str.length();
        String flipped = str.charAt(size - 1) + str.substring(1, size - 1) +
                        str.charAt(0);
        System.out.println(flipped);
        num1 = size;
        for (int num : nums1)
            num *= 10;
        System.out.println(nums1[0]);
        double sum = 0;
        for (int i = 0; i < nums2.length; i++) {
            sum += nums2[i];
            nums2[i] *= 10;
        }
        System.out.println(nums2[0]);
        return sum;
    }

    // Displays numbers on one line
    public static void showArray(int[] array) {
        System.out.print("Array: ");
        for (int num : array)
            System.out.print(num + ", ");
        System.out.println();
    }

    public static void main(String[] args) {
        int[] array1 = {2, 3, 4, 5};
        int[] array2 = {1, 2, 3, 4};
        String aString = "Hello Class!";
        int aNum = 20;
        double aVal = fiddle(aNum, array1, array2, aString);
        System.out.println(aNum);
        showArray(array1);
        showArray(array2);
        System.out.println(aString);
        System.out.println(aVal);
    }
}
```



Problem 5) [10 marks]

Here are a bunch of classes and an interface from the same Java project:

```
public abstract class Base {

    public int sum(int a, int b) {
        return a + b;
    }

    public int multiply(int a, int b) {
        return a * b;
    }

    public abstract int subtract(int a, int b);
} // end Base

interface Dividing {

    int divide(int a, int b) throws DivideByZero;
} // end Dividing

public class DivideByZero extends Exception {

    public DivideByZero(String message) {
        super(message);
    }
} // end DivideByZero

public class Concrete extends Missing {

    public int multiply(int a, int b, int c) {
        return a * b * c;
    }
} // end Concrete
```

There are two more classes in this project. The one on the next page contains a main method that tests the other classes:

Problem 5, Cont.)

```
public class Demonstration {

    public static int remainder(int a, int b) {
        return a % b;
    }

    public static void main(String[] args) {

        Concrete test = new Concrete();           // Prints:
        System.out.println(test.sum(4, 5));        // 9
        System.out.println(test.sum(4, 5, 6));     // 15
        System.out.println(remainder(12, 5));     // 2
        System.out.println(test.multiply(4, 5));   // 20
        System.out.println(test.multiply(3, 4, 5)); // 60
        System.out.println(test.subtract(4, 5));   // -1

        try {
            System.out.println(test.divide(10, 0));
        } catch (DivideByZero e) {
            System.out.println(e.getMessage()); //Attempt to divide by 0!
        }
        try {
            System.out.println(test.divide(10, 2)); // 5
        } catch (DivideByZero e) {
            System.out.println(e.getMessage());
        }
        try {
            Dividing dTest = new Missing();
            System.out.println(dTest.divide(15, 2)); // 7
        } catch (DivideByZero e) {
            System.out.println(e.getMessage());
        }

    } // end main

} // end Demonstration
```

As you can see the listing shown above contains the output of the program when run as in-line comments. One class is Missing. The Missing class extends one class and implements an interface. Write the Missing class on the next page:

Problem 5, Cont.)

Problem 6) [30 marks]

For this problem you will need to write two classes, the first is an exception class called "ItemException" and the second an encapsulated class called "ItemRecord" that will use the ItemException class.

The exception class only needs the one constructor that takes a String type message.

The ItemRecord class is designed to hold an inventory count for a single item from a store. It has three attributes:

- A String type store code which must be at least five characters in length.
- A String type SKU code which must be exactly 10 characters in length and must consist of a combination of numeric digits (0 to 9, inclusive) and a single hyphen, -, at index position 5 in the String. For example, the SKU codes "45307-1239" and "02357-0012" would be legal, but the codes "89abc-3451", "89-1234567" and "023537-0012" would not be legal.
- An int count for the inventory of this item. The count must be greater than or equal to zero.

The ItemRecord class has a single constructor and is immutable. It has an accessor for each attribute. It also needs the standard equals(), compareTo() and toString() methods, but does not need a clone() method since it is immutable.

Equality of two ItemRecord objects is defined as both objects having the exact same store code and sku code. The two objects can have different item counts and still be equal. You must override the equals() method inherited from the Object class.

For sorting purposes, when the compareTo() method will be used, ItemRecord objects are compared on an alphabetical basis using the store code first, followed by the SKU code, also on an alphabetical basis. A negative return from compareTo() means the supplied ItemRecord object comes later in the alphabet than the current ItemRecord object.

Finally, the String returned by the toString() method would look like the following examples:

Store: KIN002, item: 45307-1239, count: 10

Store: LON001, item: 02357-0012, count: 20

No other public methods are required. Comments are not required. No other classes are required. A main method is not required.

Write the exception class first on the following page, then write the ItemRecord class starting on the middle of the next page and following onto the next three pages, as needed. If you run out of room you can continue the class on the last empty page of the exam.

The Character wrapper class owns the static method .isDigit() that returns true if the supplied char is a digit. The String class owns the methods .length(), .charAt() (supplies the char at the given index position) and .equals().

Please don't separate the pages of this exam or if you must, then **put them back in the same order!**

Problem 6, Cont.) The ItemException Class:

The ItemRecord class:

Problem 6, Cont.)

Problem 6, Cont.)

Problem 6, Cont.)

Problem 7) [10 marks] Do part A or part B, but not both.

Part A) The following code is the xml language contents of an fxml file attached to a JavaFX project:

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.*?>
<?import java.lang.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.layout.VBox?>

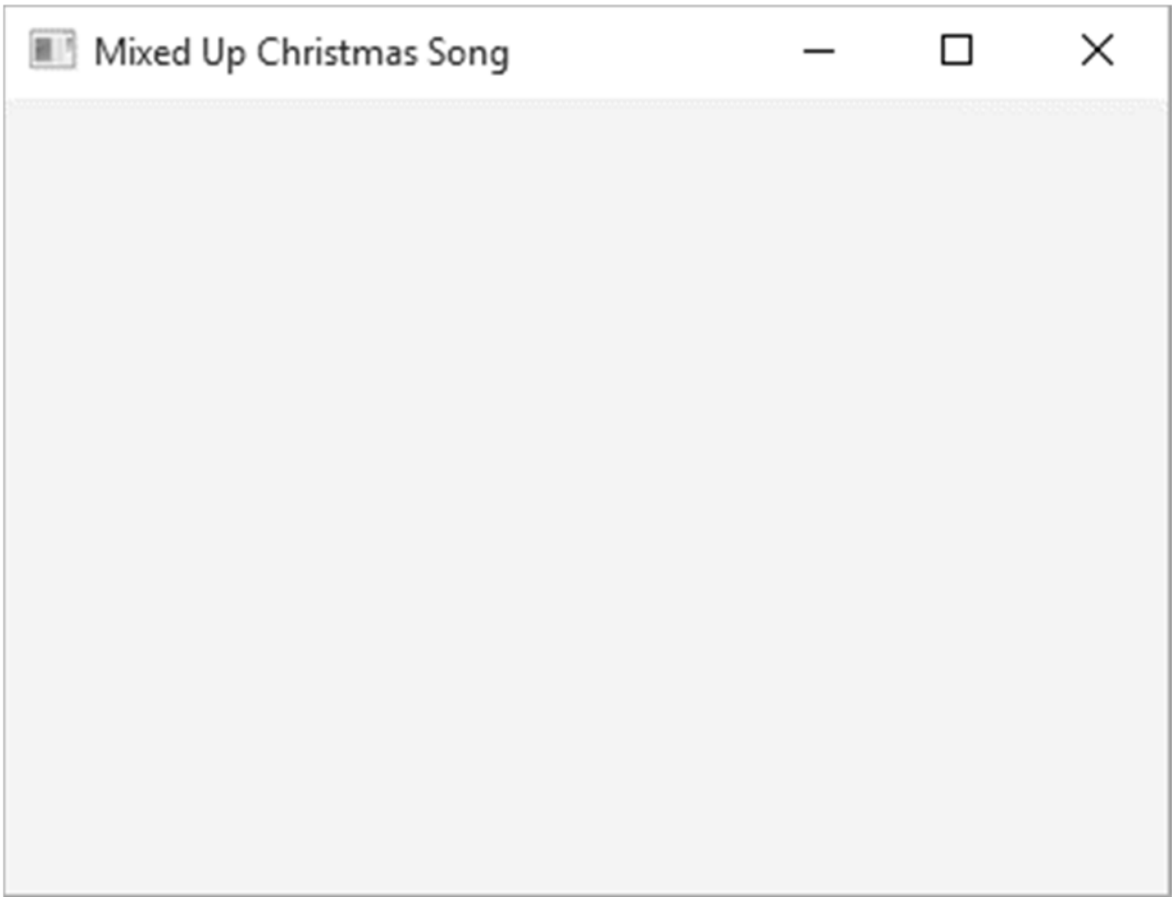
<VBox prefHeight="234.0" prefWidth="407.0" spacing="10.0"
stylesheet="@application.css" xmlns="http://javafx.com/javafx/8.0.40"
xmlns:fx="http://javafx.com/fxml/1">
    <children>
        <Label prefHeight="23.0" prefWidth="327.0" text="Four calling hens!" />
        <HBox prefHeight="100.0" prefWidth="299.0" spacing="10.0">
            <children>
                <Button fx:id="btn1" mnemonicParsing="false" text="Three" />
                <Button fx:id="btn3" mnemonicParsing="false" text="French" />
                <Button fx:id="btn2" mnemonicParsing="false" text="Birds!" />
            </children>
        </HBox>
        <HBox prefHeight="100.0" prefWidth="200.0" spacing="20.0">
            <children>
                <Label text="Two" />
                <Button fx:id="btn5" mnemonicParsing="false" text="Partridge" />
                <Label text="Doves!" />
            </children>
        </HBox>
        <HBox prefHeight="100.0" prefWidth="200.0">
            <children>
                <Label text="And a Turtle in a " />
                <Button fx:id="btn4" mnemonicParsing="false" text="Pine Tree!" />
            </children>
        </HBox>
    </children>
</VBox>
```

Sketch the window on the next page.

Do not worry about absolute pixel sizes, the shape of the letters or drawing straight lines, just try to get the relative positions of the components correct.

Draw a label as just text and a button as text inside a rectangular outline. Don't worry about colours or shading.

Part A, Cont.)



Part B) Here is the content of the Main.java file from a different JavaFX project than the one used in Part A:

```
package application;

import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.layout.GridPane;

public class Main extends Application {
    @Override
    public void start(Stage primaryStage) {
        try {
            GridPane root = new GridPane();
            Button button1 = new Button("And a Partridge");
            Button button2 = new Button("Tree");
            Button button3 = new Button("Turtle Doves");
            Button button4 = new Button("Three");
            Button button5 = new Button("Hens");
            Label label1 = new Label("Four Calling");
            Label label2 = new Label("Two");
            Label label3 = new Label("in a Pine Tree");
            Label label4 = new Label("French");
            Label label5 = new Label("in a");
            //          col, row
            root.add(label2, 0, 0);
            root.add(button3, 1, 0);
            root.add(label3, 2, 0);
            root.add(label1, 0, 1);
            root.add(button5, 1, 1);
            root.add(button1, 0, 2);
            root.add(label5, 1, 2);
            root.add(label4, 2, 2);
            root.add(button2, 3, 2);
            Scene scene = new Scene(root, 760, 260);
            scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());
            primaryStage.setScene(scene);
            primaryStage.setTitle("Another Mixed Up Song");
            primaryStage.show();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

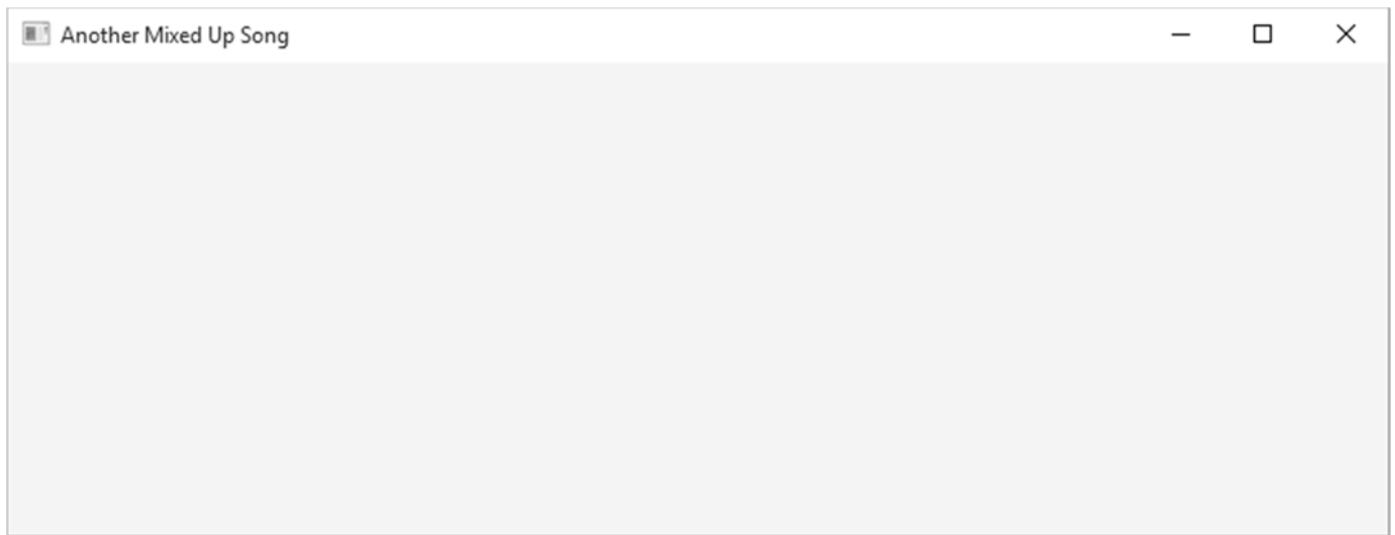
Part B, Cont.)

```
    public static void main(String[] args) {  
        Launch(args);  
    }  
} // end Main class
```

Sketch the window below.

Do not worry about absolute pixel sizes, the shape of the letters or drawing straight lines, just try to get the relative positions of the components correct.

Draw a label as just text and a button as text inside a rectangular outline. Don't worry about colours or shading.



Extra Page