# CSD-4464 Java EE

Class 1: Reviewing the Java Language

Lambton College

# Agenda

- Variables and Scope
- Control structures
- Methods and Overloading
- Interfaces and Polymorphism (inheritance)


Lambton College

# What are variables?

- Used to store changing (mutable) values in the program of a specified type
- Variables can be made immutable by using the key word final
- Making variables immutable can be a good way to help prevent unexpected bugs from occurring due to the program changing a variables value when it shouldn’t
- Variables tend to fall into two categories: primitive and Object

# Primitives and Objects

- Primitive variables are the lowest level type of variable
- Primitives **cannot** be null, setting a primitive to null will result in a NPE
- Primitive types – int, boolean, char, long, byte, ect.
- Objects are an instance of a class
- Objects **can** be a null value
- Objects enable generic programming (covered in later classes) and object-oriented programming paradigms

# Scope

- Private, Package-Private, Protected, Public

**Accessible/Visible**

| | Difference class, same package | Subclass but different package | Different package, no inheritance |
|---|---|---|---|
| Class A { | Class B { | Class C extends A { | Class D { |
| private int x; | | | |
| Int y; | x | | |
| protected int z; | x | x | |
| public int az; | x | x | x |
| } | } | | |

Lambton College

# Loops

- Do While – loops until condition is met, condition is evaluated at the end of the loop
- While – like the Do While, except the condition is evaluated before the loop
- For loop – loops for a specified (n) amount of times, maintains counter variable in the loop
- For each loop – loop runs for each of the values in an array/collection, gives access to a single item each loop iteration

# Decision Structures

- If / else
- Switch case
- Relational operators ( >, <, >=, <=, ==, !=)
  - Note **never** compare objects with relational operators
- Logical operators ( &&, ||, ! )
- Switch statement

# Methods

- Used to break problems into smaller subproblems
- void vs value-returning
- pass-by-value vs pass-by-reference
- Methods can call themselves (recursion)
- Static vs instance methods

# Method Overloading

- Method overloading allows for multiple methods to be named the same, but have different signatures
- Allows for providing default values to methods in java.
- It is different from overriding. In overriding, a method has the same method name, type, number of parameters, etc

Example

```
Public void Integer multiplyNumbers(Integer a, Integer b) {
        return multiplyNumbers(a, b, 1);
}
Public void Integer multiplyNumbers(Integer a, Integer b, Integer c) {
        return a * b  * c;
}

multiplyNumbers(2, 3) // outputs 6
multiplyNumbers(2, 3, 4) // outputs 24
```

# Interfaces

- Interfaces are a contract of methods a class / object must have
- Uses the keyword **implements** instead of extends
- Provides a powerful way to abstract dependences, which allows you to change components without changing your overall code.

```
interface DatabaseService {
        public Record getRecord();
}

Public class MySqlService implements DatabaseService {
        @Overrides
        public Record getRecord() {
                //fetches the record
        }
        …
}
```

# Polymorphism (Inheritance)

- Allows for classes to 'inherit' fields and methods from a parent class, as well as define it's own.
- Uses the keyword **extends**
- The Class that inherits properties is referred to as the **subclass**, and the base class that has its properties inherited is referred to as the **superclass**
- Inheritance can be a powerful tool that can enable code reuse, however it can also be a double edged sword as it can make code difficult to maintain and read, so use carefully!

Example

```
public class Person {
        int age;


        public int getAge() {
                return this.age;
        };
}
public class Student extends Person {
        public isOldEnoughToDrive() {
                super.getAge() >= 16
        }
}
```