ATLS 4120: Mobile Application Development
Week 11: Java

History

- Java was created by James Gosling, Mike Sheridan, and Patrick Naughton
- Sun Microsystems released Java 1.0 in 1995
- In 2006 and 2007 Sun released Java as free and open-source software, under the terms of the GNU General Public License (GPL)
- Sun was bought by Oracle in 2009/2010, and Oracle continues to support Java

Java

- Java is a fully object-oriented programming language used by many software developers
  - Java has many standard libraries
  - Android provides a subset of these only for Android
- Java is a compiled language that is platform independent
  - The compiler creates bytecode that the Java Virtual Machine(JVM) then interprets into machine language
    - Bytecode is similar to machine language but not associated to any specific CPU
  - Android uses Ahead of Time(AoT) compilation to convert the bytecode into optimized machine code using the Android Runtime(ART) virtual machine when installed on an Android device to improve run time performance
    - *AoT* compilation only occurs in Android KitKat (4.4) and above, but it also provides backwards compatibility. Prior versions of Android relied on another virtual machine known as *Dalvik*.
  - Processing is also build on top of Java

Data Types

- Java has standard primitive data types
  - int
  - float
  - double
  - char
  - boolean
- Java is a strongly typed language so you need to specify the data type when declaring a variable

Variables and Constants

Java is case sensitive and all the naming conventions you're used to apply.

- Start with a letter, $, or _ (not a number)
- Avoid key words
- Constants use the keyword **final** and are usually capitalized
  - Constants can't have their value changed
  - A final method means you can't override it
  - A final class means you can't create a subclass

https://repl.it/@aileenjp/Java-intro

```
double temp;
temp=50;
System.out.println(temp);
```

```
final int FREEZING = 32;
System.out.println(FREEZING);

FREEZING = 30; //error, can't change the value of a constant
```

If else

In Java the test condition must be in parenthesis. The body must be in curly brackets if there's more than one instruction.

- Java has all the typical logical and relational operators (==, !=, <, <=, >, >=, &&, ||)
- Java also supports the ternary operator shorthand
  - (expression) ? value if true : value if false

String equality

- Don't use == to test for String value equality, use **.equals()** instead
- == tests for reference equality (compares memory addresses to see if they are the same object)
- **.equals()** tests the string contents for value equality (whether they are logically equivalent).

```
if (temp > FREEZING){
    System.out.println("It's nice out");
  } else {
    System.out.println("It's cold out");
  }
```

Change temp to 30 to test the else statement.

Switch

A switch statement compares a variable's value to a series of case values and when a match is found the statements in that case are run.

- The variable used in a switch statement can only be integers, convertable integers (byte, short, char), strings and enums.
- The value for a case must be the same data type as the variable in the switch and it must be a constant or a literal.
- Each case can be terminated with a break statement which terminates the switch statement
- If a break statement is not supplied, the code will continue executing into the next case statement until a break is reached
- You can also supply a default option to execute if none other cases apply

```
String grade = "B";
switch (grade)
{
  case "A":
    System.out.println("Excellent!");
    break;
  case "B":
    System.out.println("Well done");
    break;
  case "C":
  case "D":
    System.out.println("You passed");
    break;
  case "F":
    System.out.println("Try again");
    break;
  default:
    System.out.println("Invalid grade");
    break;
}
```

Change grade to “C”. Note that case C has no instructions and no break so even if this case is a match it will go on to D and run the instructions in that case. This is called fall through and is fine if done intentionally.

Arrays

In Java arrays are a fixed size collection of elements all of the same data type.

- Arrays can contain both primitive data types as well as objects of a class
- array members are accessed using []

```
  char[] sizes = {'s', 'm', 'l'};

  String[] more_sizes = new String[4];
  more_sizes[0] = "extra small";
  more_sizes[1] = "small";
  more_sizes[2] = "medium";
  more_sizes[3] = "large";
```

If you try to add an item to an array that would go over its size you will get an array index out of bounds exception. `java.lang.ArrayIndexOutOfBoundsException`

```
more_sizes[4] = "extra large";
```

ArrayList is a dynamic sized arrays in Java that implement List interface. ArrayList is part of collection framework in Java.

- ArrayList only supports object entries, not the primitive data types.
- ArrayList has a set of methods to access elements and modify them, can’t use []

(add at the very top of the repl)

```
import java.util.ArrayList;

  ArrayList<String> size_list = new ArrayList<>();
  size_list.add("small");
  System.out.println(size_list);
```

You can add as many items as you want to an ArrayList

```
size_list.add("medium");
System.out.println(size_list);
```

There are other data structures and collection types in Java that we'll get to next semester.

Loops

Java supports the common loops, for, while, and do while as well as the for each loop.

```
for(int i=0; i<sizes.length; i++){
  System.out.println(sizes[i]);
}

for(String mysize: more_sizes){
  System.out.println(mysize);
}

for(String mysize: size_list){
  System.out.println(mysize);
}
```

OOP

Java is an object-oriented programming language.

Classes

- A class is a template/blueprint that describes the behavior/state that the object of its type supports.
  - Data members for state (variables)
  - Method members tor behavior (methods)
- In Java it's an Abstract Data Type as it describes a real-world abstraction
- You can control the visibility of a class as well as its variables and methods by specifying the access level
- The access levels are
  - public: accessible outside the class
  - private: only accessible inside the class
  - protected: accessible by the class and its subclasses
- Variables should be private and be accessible only by the method members declared for the same class
  - No methods outside a given class can access the private data members of the class.
- Class names should start with a upper case letter

Methods

A method is where the logic is written to define a specific behavior. Data is manipulated and all the actions are executed.

- Public methods can be accessed from outside the class and provide the public interface to the class
- Private methods are helper methods used inside the class
- Method names should start with a lower case letter

- Getter methods are used to get data
- Setter methods are used to set data

Java uses pass by value (not reference) for everything.

```
class Animal {
  //instance variables
  private int weight;
  private String name;

  //methods
  //setter
  public void setName(String animalName){
    name = animalName;
  }
  //getter
  public String getName(){
    return name;
  }
}
```

Objects

An object is an instance, or occurrence, of a given class. An object of a given class has the structure and behavior defined by the class that is common to all objects of the same class.
Many different objects can be defined for a given class with each object made up of the data and methods defined by the class
It's only when an object is instantiated that memory is allocated
Objects are instantiated by calling the class's constructor method
In Java, objects store the reference to the memory where its data is stored

```
Animal animal1 = new Animal();
animal1.setName("Fred");
System.out.println(animal1.getName());
```

Constructors

Constructor methods are initializer methods
Classes have constructor methods to initialize objects of that class

- Constructors have the same name as the class
- A class can have multiple constructors that each take a different number and/or type of parameters
- Constructors don't have a return type
- If you don't provide a constructor method Java will create one

```
public Animal(){
}
```

```
  public Animal(String animalName){
    name = animalName;
  }

  public Animal(int animalWeight, String animalName){
    weight = animalWeight;
    name = animalName;
  }

Animal animal2 = new Animal("Wilma");
Animal animal3 = new Animal(20, "BamBam");
System.out.println(animal2.getName());
System.out.println(animal3.getName());
```

Variable types

There are three kinds of variables in Java −

- Local Variables
  - Local variables are declared in methods, constructors, or blocks.
  - Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.
  - Their scope is where they are declared -- method, constructor, or block.
  - There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.
  - Access modifiers cannot be used for local variables.
- Instance Variables
  - Instance variables are declared in a class, but outside a method, constructor or any block as they store essential data for an object's state.
  - The instance variables are visible for all methods, constructors and block in the class and can be accessed directly by calling the variable name inside the class.
  - Instance variables are created, and space allocated, when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
  - Instance variables have default values. For numbers, the default value is 0, for Booleans it is false, and for object references it is null. Values can be assigned during the declaration or within the constructor.
  - Access modifiers can be given for instance variables.
- Class/Static Variables
  - Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
  - There is only one copy of each class variable per class, regardless of how many objects are created from it.
  - Static variables are usually used for declaring constants.
  - Static variables are created when the program starts and destroyed when the program stops.
  - Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class.

- Default values are same as instance variables. For numbers, the default value is 0; for Booleans, it is false; and for object references, it is null. Values can be assigned during the declaration or within the constructor.
- Static variables can be accessed by calling with the class name ClassName.VariableName.

Encapsulation

Encapsulation means data and operations are packaged into a single well-defined programming unit
In Java, the class provides for encapsulation

- The getter and setter methods provide the public class interface that is accessible from outside the class
- The private data members provide the information that is accessible only from within the class, thus providing the information hiding

We've already used some getter and setter methods
getText() EditText
setText() TextView

Inheritance

Inheritance allows you to define a hierarchy of classes with subclasses acquiring the properties of the superclass.

```
class Dog extends Animal {
  private String breed;

  public void setBreed(String dogBreed){
    breed = dogBreed;
  }

  public String getBreed(){
    return breed;
  }
}

Dog pet = new Dog();
pet.setName("Marmaduke");
pet.setBreed("Great Dane");
System.out.println(pet.getName() + " is a " + pet.getBreed());
```

If you tried to call getBreed() on an object of the Animal class you'd get an error.

```
animal1.getBreed();
```

Casting

Casting is taking an object of a certain class and changing its type to be of a different class

- Downcasting turns an object into a more specific type of object so you can have access to the methods and properties of that class
- Upcasting turns an object into a more generic type of object
- You will get a ClassCastException error if it's an illegal cast

- Downcasting is the more common direction for casting

<u>Java String class</u>

- The String class is immutable, so that once a String object is created cannot be changed.
- When you assign it a new value it's actually creating a new object.

http://developer.android.com/reference/packages.html

<u>Java Packages</u>

A package groups related classes together in one directory whose name is the same as the package name

- The java.lang package provides a number of helpful classes such as the String class
- The android.app package contains high-level classes encapsulating the overall Android application model

<u>Comments</u>

- Java has the standard comments
  - // single line
  - /* multi-line */

Interfaces

In Java an interface can be defined as a contract between objects on how to communicate with each other.
An interface defines the methods, a deriving class (subclass) should use. But the implementation of the methods is totally up to the subclass.
A subclass that implements an interface must implement the required methods and has the choice to implement any optional ones.

Debugging

Debug messages

- To log messages to the console in Java use **System.out.println("string");**
- Print messages to the LogCat
  - DEBUG: Log.d(tag, message);
  - ERROR: Log.e(tag, message);
  - INFO: Log.i(tag, message);
  - VERBOSE: Log.v(tag, message);
  - WARN: Log.w(tag, message);
- Log.i("TAG", "in start of count method");
- Filter to just see the logs for your tags

Debugging https://developer.android.com/studio/debug/index.html
More info on Logcat https://developer.android.com/studio/debug/am-logcat.html