Mobile Application Development
Aileen Pierce

JAVA FOR ANDROID

Java

- 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 a compiled language that is platform independent
  - The compiler creates bytecode that the Java Virtual Machine(JVM) then interprets into machine language
  - 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

Primitive 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

- Variables

```
int result;
int i = 5;
```

- Constants use the keyword **`final`** and are usually capitalized
  - Constant variables can’t have their value changed

    ```
    final int SIZE = 3;
    ```

  - A final method means you can’t override it
  - A final class means you can’t create a subclass

Conditionals

```
if(test expression)
{
     // Execute this code if test is true
} else {
     // Execute this code if test is false
}
```

Conditionals

- Java has all the typical logical and relational operators (==, !=, <, <=, >, >=, &&, ||)
- Don’t use == to test for String value equality, use **`.equals()`** instead
- == tests for reference equality (whether they are the same object)
- **`.equals()`** tests for value equality (whether they are logically equivalent).

Conditionals

- A switch statement checks many options called cases
- A selector variable is compared against a series of case values
- If a match is found the corresponding case statements are run
- Each case can be terminated with a break statement
- If a break statement is not supplied, the code will continue executing into the next case statement
- You can also supply a default option to execute if none other cases apply

Conditionals

```
switch(selector variable)
{
     case value 1 : case 1 statements
                         break;
     case value 2 : case 2 statements
                         break;
              •
              •
              •
     case value n : case n statements
     default : default statements
}
```

Loops

- Java supports the common loops, for, while, and do while

```
while(test expression)
{
        statement 1;
        statement 2;
                •
                •    //LOOP BODY
                •
        statement n;
}
```

Loops

```
for(int count=initial value; test
expression; increment/decrement count)
 {
   statement 1;
   statement 2;
       •
       •     //LOOP BODY
       •
   statement n;
 }
```

Arrays

- In Java arrays are a fixed size collection of elements all of the same data type
- Declare an array

```
dataType[] myarray;
```

- Create an array

```
myarray = new dataType[size];

double[] myarray = new double[10];
myarray[0]=5.6;
myarray[1]=4.2;
```

Classes

- A class is an Abstract Data Type because it describes a real-world abstraction
- Classes provide the template for their objects
  - Data members (variables)
  - Method members (methods)
- You can control the visibility of a class as well as its variables and methods by specifying the access level

Classes

- The access levels are
  - – public: accessible outside the class
  - – private: only accessible inside the class
  - – protected: accessible by the class and its subclasses
- Variables are usually private because they are accessible only by the method members declared for the same class
- Methods are usually public because they can be accessed from outside the class
  - – Provide the public interface to the class

Methods

```
access return_type method_name(parameters)
{
     method body
}
```

- Return type
  - void methods don’t return a value

```
public void start_game()
{statements}


public float calc_temp(int tempF)
{return (tempF-32)*.56;}
```

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

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

Classes

```
public class Animal {
//variables
    protected int weight;
    protected String name;
//constructor methods
    public Animal() {
    }
    public Animal(int animalWeight){
        weight= animalWeight;
    }
```

Classes

```
//methods
  public void setName(String
  animalName){
     name = animalName;
  }

  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

Objects

- 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

```
ClassName objectName = new ClassName();
```

Objects

```
Animal animal1 = new Animal();
Animal animal2 = new Animal(50);
animal1.setName(“Fred”);
String name1 = animal1.getName();

animal2.setName(“Wilma”);
String name2 = animal2.getName();
```

Inheritance

- Inheritance enables classes to form a hierarchy like a family tree.
- Allows subclasses to share the structure and behavior of its superclass.
  - Superclass is the parent class
  - A subclass extends a class
    - Inherits from the superclass
    - Can add properties and methods

Inheritance

```
public class Dog extends Animal {
//variables
     private String breed;
//methods
  public void setBreed(String dogBreed){
     breed= dogBreed;
  }

  public String getBreed(){
     return breed;
  }
```

Inheritance

```
Dog myDog = new Dog();
myDog.setName(“Cole”);
String dog_name = myDog.getName();

mydog.setBreed(“Black Lab”);
String dog_breed = myDog.getBreed();
```

Casting

- Casting is taking an object of a certain class and turning it into an object of another 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

```
Dog yourDog = (Dog)animal1; //downcast

Animal animal3 = (Animal)myDog; //upcast
```

Java Objects

- 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

```
String str="Hello World";
String str=new String("Hello World");
```

- The android.app package contains high-level classes encapsulating the overall Android application model

Java

- To log messages to the console in Java use **`System.out.println("string");`**
- Java has the standard comments
  - – // single line
  - – /* multi-line */