Mobility Programming

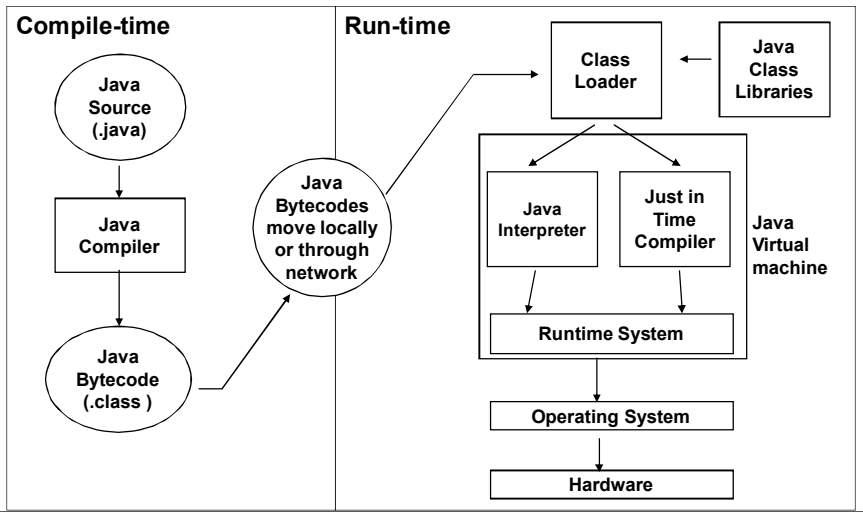
Lecture 1: Java Review

DR. RAMEZ ALKHATIB

Java

- Java is an *object-oriented* language, with a syntax similar to C
  - Structured around *objects* and *methods*
  - A method is an action or something you do with the object
- Avoid those overly complicated features of C++:
  - Operator overloading, pointer, templates, friend class, etc.

How it works…!



Getting and using java

- JDK freely download from http://www.oracle.com
- All text editors support java
  - Vi/vim, emacs, notepad, wordpad
  - Just save to .java file
- Eclipse IDE
  - Eclipse
  - http://www.eclipse.org
  - Android Development Tools (ADT) is a plugin for Eclipse

Compile and run an application

- Write java class `HolaWorld` containing a `main()` method and save in file "*`HolaWorld.java`*"
  - The file name *MUST* be the same as class name
- Compile with: `javac HolaWorld.java`
- Creates compiled .class file: *`HolaWorld.class`*
- Run the program: *`java HolaWorld`*
  - Notice: use the class name directly, no .class!

Hola World!

File name: *HolaWorld.java*

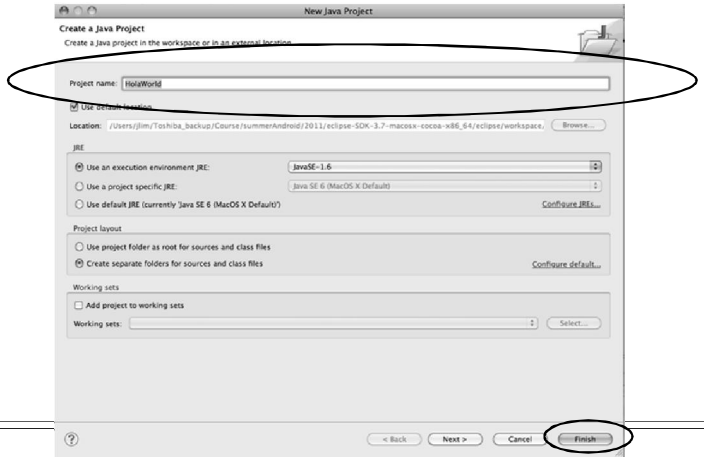
```
/* Our first Java program – HolaWorld.java */

public class HolaWorld {
      //main()
      public static void main ( String[] args )
{
            System.out.println( "Hola world!" );
      }
}
```

Command line arguments

Standard output, print with new line

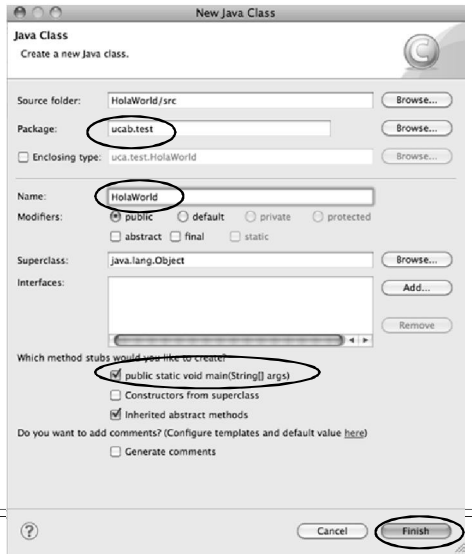
HolaWorld in Eclipse - create a new project

- File > New > Java Project
- Project Name : HolaWorld



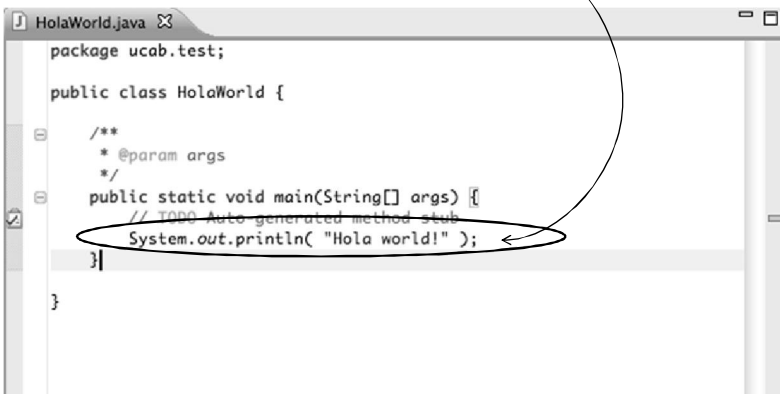
HolaWorld in Eclipse – add a new class

- File > New > Class
- source folder : HolaWorld/src
- Package : ucab.test
- Name : HolaWorld
- check "public static void main (String[] args)



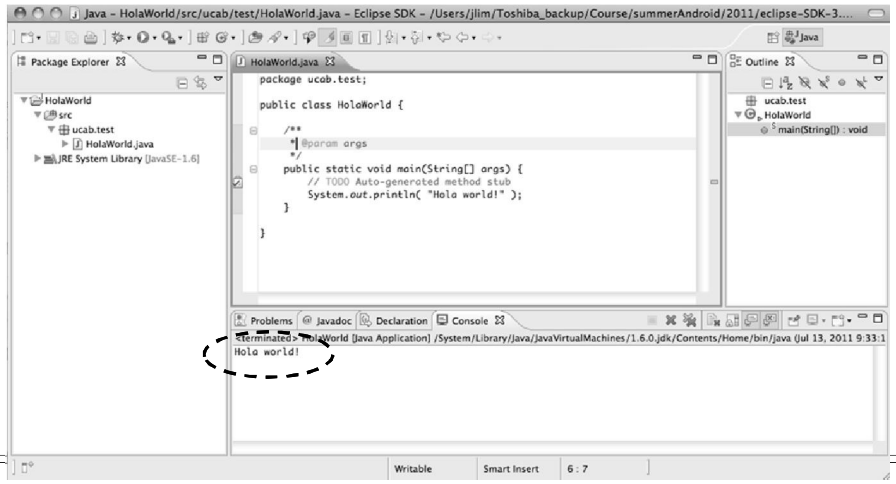
HolaWorld in Eclipse – write your code

- Add your code
  System.out.println("Hola world!");



HolaWorld in Eclipse – run your program

- Run > Run As > Java Application

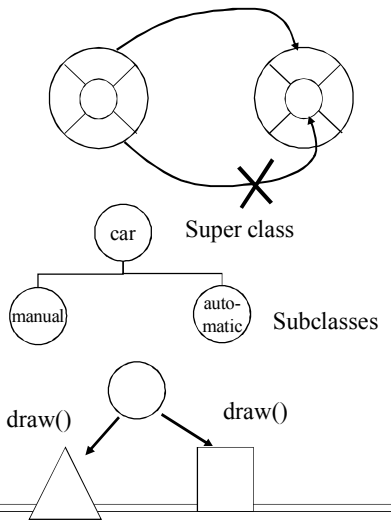


Object-Oriented

- Java supports OOP
  - Polymorphism
  - Inheritance
  - Encapsulation
- Java programs contain nothing but definitions and instantiations of classes
  - Everything is encapsulated in a class!

The three principles of OOP

- Encapsulation
  - Objects hide their functions (**methods**) and data (**instance variables**)
- Inheritance
  - Each **subclass** inherits all variables of its **superclass**
- Polymorphism
  - Interface same despite different data types



About class

- Fundamental unit of Java program
- All java programs are classes
- A class is a template or blueprint for objects
- A class describes a set of objects that have identical characteristics (data elements) and behaviors (methods).
  - Existing classes provided by JRE
  - User defined classes
- Each class defines a set of fields (variables), methods or other classes

What is an object?

- An object is an instance of a class
- An object has state, behavior and identity
  - Internal variable: store state
  - Method: produce behavior
  - Unique address in memory: identity

What does it mean to create an object?

- An object is a chunk of memory:
  - holds field values
  - holds an associated object type
- All objects of the same type share code
  - they all have same object type, but can have different field values.

Class Person: definition

```
class Person {
     String name;
     int height; //in inches
     int weight; //in pounds
     public void printInfo(){
       System.out.println(name+" with height="+height+",
weight="+weight);
     }
}
```

Variables

Method

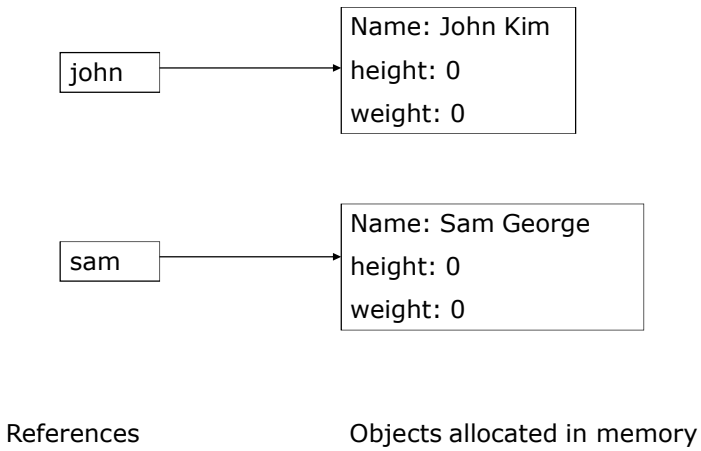
```
class ClassName{
        /* class body goes here */
}
```

class: keyword

Class Person: usage

```
Person john;        //declaration
john = new Person();//create an object of Person
john.name= "John Kim";//access its field
Person sam = new Person();
sam.name="Sam George";
john.printInfo();     // call method
sam.printInfo();
```

Class Person: reference



Reference

```
Person john;
```
//only created the reference, not an object. It points to nothing now (null).

```
john = new Person();
```
//create the object (allocate storage in memory), and john is initialized.

```
john.name="John";
```
//access the object through the reference

Primitive types

Primitive type	Size	Minimum	Maximum	Wrapper type
boolean	1-bit	—	—	**Boolean**
char	16-bit	Unicode 0	Unicode 2^{16}- 1	**Character**
byte	8-bit	-128	+127	**Byte**
short	16-bit	-2^{15}	$+2^{15}-1$	**Short**
int	32-bit	-2^{31}	$+2^{31}-1$	**Integer**
long	64-bit	-2^{63}	$+2^{63}-1$	**Long**
float	32-bit	IEEE754	IEEE754	**Float**
double	64-bit	IEEE754	IEEE754	**Double**

Reference vs. primitive

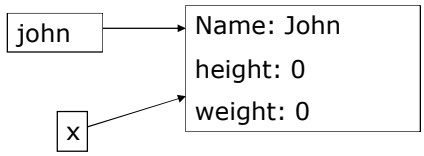
Java handle objects and arrays always by reference.
- classes and arrays are known as reference types.
- Class and array are composite type, don't have standard size

Java always handle values of the primitive types directly
- Primitive types have standard size, can be stored in a fixed amount of memory

Because of how the primitive types and objects are handles, they behave different in two areas: copy value and compare for equality

Copy

- Primitive types get copied directly by =
  - int x= 10; int y=x;
- Objects and arrays just copy the reference, still only one copy of the object existing.




```
Person john = new Person();
john.name="John";
Person x=john;
x.name="Sam";
System.out.println(john.name);  // print Sam!
```

Scoping: in a class

```
public class VisibilityDemo {
    private int classVar1;
    private int classVar2;
    public int method1(int x) {
        int local = 0;
        for (int i = 0; i < x; i++){
            local += i;
        }
        return local;
    }

    public void method2 ( int x) {
        classVar1 = x + 10;
        classVar2 = method1(classVar2);
    }
}
```

The red identifiers denote class varialbes and methods. They have visibility anywhere inside the outermost pair of red curly brackets

The blue identifiers are local to a single block (identified by blue brackets). They are not visible to anything outside of their block, but are visible inside blocks nested inside of the blue bracketed block.

The gray identifiers are found inside the for-loop. The gray variable *i* is visible only inside the loop.

Parameters are denoted by green. They are visible everywhere inside the method in which they appear, but only in that method

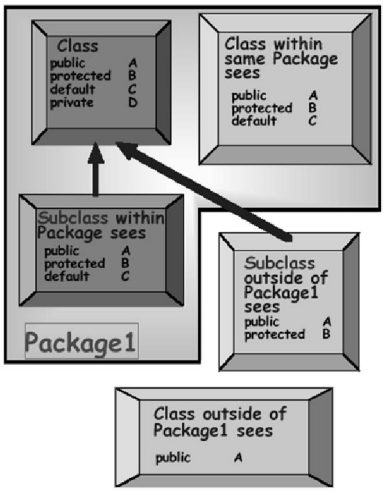
Access control

Access to packages
- Java offers no control mechanisms for packages.
- If you can find and read the package you can access it

Access to classes
- All top level classes in package P are accessible anywhere in P
- All public top-level classes in P are accessible anywhere

Access to class members (in class C in package P)
- Public: accessible anywhere C is accessible
- Protected: accessible in P and to any of C′s subclasses
- Private: only accessible within class C
- Package: only accessible in P (the default)

Scoping: visibility between classes



A summary of Java scoping visibility

The static keyword

Java methods and variables can be declared static

These exist **independent of any object**

This means that a Class's

- static methods can be called even if no objects of that class have been created and
- static data is "shared" by all instances (i.e., one rvalue per class instead of one per instance

```
class StaticTest {static int i = 47;}
StaticTest st1 = new StaticTest();
StaticTest st2 = new StaticTest();
// st1.i == st2.I == 47
StaticTest.i++;    // or st1.I++ or st2.I++
// st1.i == st2.I == 48
```

XML Review

XML

- eXtensible Markup Language
- Simple text (Unicode) underneath
- Tags (like in HTML) are used to provide information about the data
- Similar to HTML, but:
  - HTML is used to describe how to display the data
  - XML is used to describe what is the data
- Often used to store and transfer data

HTML Example

```
<html>
 <head><title>Here goes the
 title</title></head.
<body>
 <h1>This is a header</h1>
   Here goes the text of the page
</body>
</html>
```

- Tags mean something specific to the browser
- They are used for display

XML Example

```
<?xml version="1.0"/>
<person>
 <name>
      <first>Jose</first>
      <last>Barrios</last>
  </name>
 <email>jb@ucab.edu</email>
 <phone 555-456-1234 />
</person>
```

- Tags mean whatever the user wants them to mean
- They are used to describe the data

XML Rules

Tags are enclosed in angle brackets.

Tags come in pairs with start-tags and end-tags.

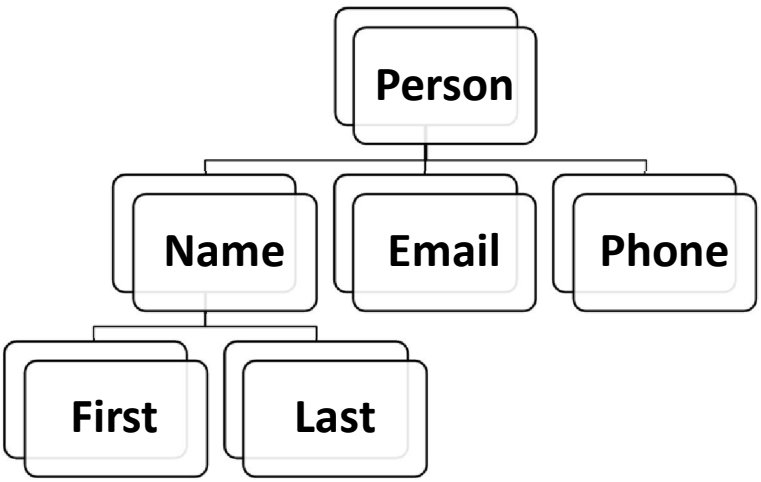
Tags must be properly nested.

- **<name><email>…</name></email>** is not allowed.
- **<name><email>…</email><name>** is.

Tags that do not have end-tags must be terminated by a '/'.

Document has a single root element

XML Documents are Trees



Android Manifest

```
❑<?xml version="1.0" encoding="utf-8"?>
❑<manifest xmlns:android="http://schemas.android.com/apk/res/android"
❑     package="com.example.helloandroid"
❑     android:versionCode="1"
❑     android:versionName="1.0">
❑    <application android:icon="@drawable/icon"
 android:label="@string/app_name">
❑        <activity android:name=".HelloAndroid"
❑                  android:label="@string/app_name">
❑            <intent-filter>
❑                <action android:name="android.intent.action.MAIN" />
❑                <category android:name="android.intent.category.LAUNCHER" />
❑            </intent-filter>
❑        </activity>
❑    </application>
❑</manifest>
```

Attributes

Questions?