CS260 Intro to Java & Android
03.Java Language Basics

http://www.tutorialspoint.com/java/index.htm

Winter 2017

What is the distinction between “fields” and “variables?”

Java has the following kinds of variables:

- Instance Variables (Non-Static Fields) – variables whose values are unique to each instance of a class
- Class Variables (Static Fields) – variables declared with the static modifier which tells the compiler exactly one copy of this variable exists
- Local Variables – a variable declared between the opening and closing brace of a method
- Parameters – variables that are arguments to a method

fields – fields in general exclude local variables and parameters

variables – in general, variables are all of the above

P1: List the instance variables, class variables, local variables, and parameters for the class Rational.

```
1  package edu.pacificu.cs.Rational;
2
3  public class Rational
4  {
5    private int mNumerator;
6    private int mDenominator;
7
8    public Rational ()
9    {
10     mNumerator = 0;
11     mDenominator = 1;
12   }
13
14   public Rational (int numerator, int denominator)
15   {
16     this.mNumerator = numerator;
17     this.mDenominator = denominator;
18   }
19   public Rational multiply (Rational fraction)
20   {
21     Rational tempFraction = new Rational ();
22
23     tempFraction.setNumerator (this.mNumerator * fraction.getNumerator());
24     tempFraction.setDenominator (this.mDenominator * fraction.getDenominator());
25
26     return tempFraction;
27   }
28
29   @Override
30   public String toString ()
31   {
32     return Integer.toString (mNumerator) + "/" +
33            Integer.toString (mDenominator);
34   }
```

Primitive Data Types

byte	8-bit signed number
short	16-bit signed number
int	32-bit signed number
long	64-bit signed number
float	32-bit IEEE 754 floating-point number
double	64-bit IEEE 754 floating-point number
boolean	either true or false
char	16-bit Unicode character
String	String is an object and NOT a primitive data type

Primitive Datatypes versus Objects

Explain the specifics of what is going on in the following Java code:

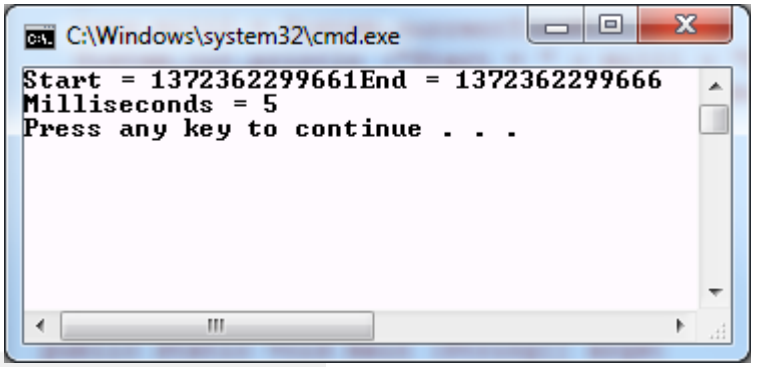
```
1   public class DataTypes
2   {
3     public static void main (String[] args)
4     {
5       int i = 5;
6       Integer j = new Integer (5);
7
8       System.out.println ("i = " + i + " j = " + j);
9     }
10  }
```

What is the output? Draw the activation record (AR).

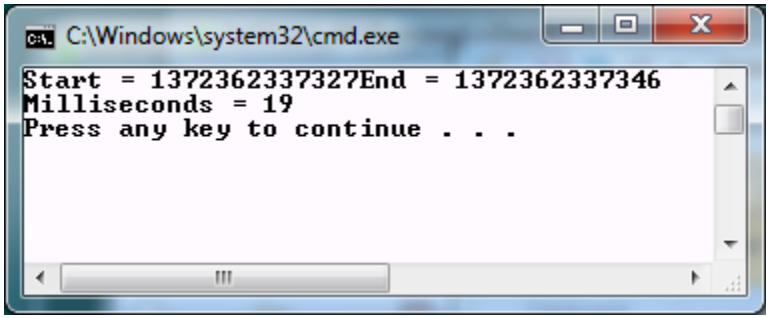
```
1   public class DataTypes
2   {
3     public static void main (String[] args)
4     {
5       int i = 5;
6       Integer j = new Integer (5);
7       change (j);
8       System.out.println ("i = " + i + " j = " + j);
9     }
10
11    public static void change (Integer k)
12    {
13      System.out.println (k);
14      k = 6;
15    }
16  }
17
```

What must be happening?

```
1  public class DataTypes
2  {
3    public static void main (String[] args)
4    {
5      long start = System.currentTimeMillis ();
6      for (int i = 0; i < 1000000; ++i)
7      {
8      }
9      long end = System.currentTimeMillis ();
10     System.out.println ("Start = " + start + "End = " + end);
11     System.out.println ("Milliseconds = " + (end - start));
12   }
13 }
```




```
1  public class DataTypes
2  {
3    public static void main (String[] args)
4    {
5      long start = System.currentTimeMillis
6      for (Integer i = 0; i < 1000000; ++i)
7      {
8      }
9      long end = System.currentTimeMillis ();
10     System.out.println ("Start = " + start + "End = " + end);
11     System.out.println ("Milliseconds = " + (end - start));
12   }
13 }
```



Default Values

Fields that are declared but not initialized are set to a reasonable default value by the compiler.

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	FALSE

Note: The compiler never assigns a default value to an uninitialized local variable. Accessing an uninitialized local variable is a compiler error.

Literals

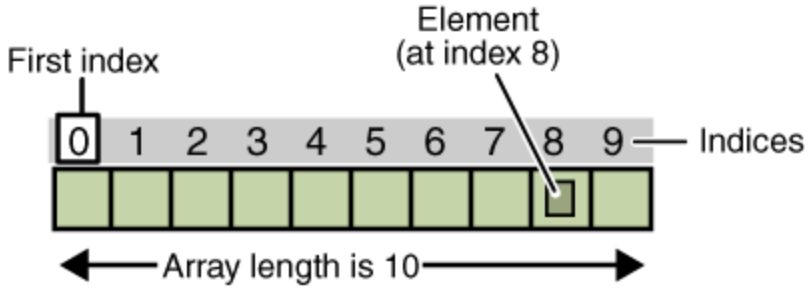
```
boolean bIsFound = false;
char ch = 'C';
byte b = 100;
short s = 10000;
int i = 100000;
long l = 100L;
int decVal = 26;      // The number 26, in decimal
int octVal = 032;     // The number 26, in octal
int hexVal = 0x1a;    // The number 26, in hexadecimal
double d1 = 123.4;
double d2 = 1.234e2;
float f1 = 123.4f;    // the default value 123.4 is a double
```

You should be noticing many similarities to C++ yet enough subtle differences that will make coding somewhat frustrating at first.

array – is a “container object” that holds a fixed number of values of a single type

container – a class used to contain other Java objects. A container class has methods for adding and deleting objects as well as aiding in iteration.

```
class Foo
{
  public static void main (String[] args)
  {
    int[] intArray; // int intArray[]; OK, don't use
    intArray = new int[10];
    intArray[5] = 99;
    System.out.println ("Value = " + intArray[5]);
```



Arrays

- are objects
- are dynamically created
- can be assigned to variables of type Object
- can use all methods of Object

```
int[] intArray; // int intArray[]; OK, don't use
intArray = new int[10];

for (int i = 0; i < intArray.length; ++i)
{
  intArray[i] = 0;
}
```

Operator Precedence	
Operators	Precedence
postfix	expr++ expr--
unary	++expr --expr +expr -expr ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Note1: All binary operators are left associative except the assignment operator is right associative

Note2: Watch out for things like:

```
byte b1 = 0, b2 = 1, b3;
b3 = b1 + b2;   // not allowed …
                // must type cast b3 = (byte) (b1 + b2);
```

There are too many of these to mention so read the tutorials closely!!!!

I would suggest that we use int and double for most of our code to avoid mixed mode arithmetic.

Declaring Classes

```
// simple class declaration
class MyClass1
{ // class body begins
     // field(s)
     // constructor(s)
     // method(s)
} // class body ends

// more complicated class declaration
class MyClass2 extends MySuperClass implements MyInterface
{ // class body begins
     // field(s)
     // constructor(s)
     // method(s)
} // class body ends
```

class declaration

In general, class declarations can include the following components in order:

1. Modifiers such as *public*, *private*, and a number of others that you will encounter later.

2. The class name, with the initial letter capitalized by convention.

3. The name of the class's parent (superclass), if any, preceded by the keyword *extends*. A class can only *extend* (subclass) one parent.

4. A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword *implements*. A class can *implement* more than one interface.

5. The class body, surrounded by braces, {}.

Access Modifiers

- public modifier—the field is accessible from all classes.
- private modifier—the field is accessible only within its own class.

```
public class Bicycle
{
  private int mCadence;
  private int mGear;
  private int mSpeed;

  public Bicycle(int startCadence, int startSpeed,
                 int startGear)
  {
    mGear = startGear;
    mCadence = startCadence;
    mSpeed = startSpeed;
  }
  …
```

Defining Methods

The only required elements of a method declaration are:

- the method's return type
- the method’s name
- a pair of parentheses, ()
- a body between braces, {}

Defining Methods

Method declarations have six components, in order:

1. Modifiers—such as public, private, and others you will learn about later.
2. The return type—the data type of the value returned by the method, or void if the method does not return a value.
3. The method name—the rules for field names apply to method names as well, but the convention is a little different.
4. The parameter list in parenthesis—a comma-delimited list of input parameters, preceded by their data types, enclosed by parentheses, (). If there are no parameters, you must use empty parentheses.
5. An exception list—to be discussed later.
6. The method body, enclosed between braces—the method's code, including the declaration of local variables, goes here.

Overloading Methods

```
public class DataArtist
{ ...
  public void draw(String s) { ... }
  public void draw(int i) { ... }
  public void draw(double f) { ... }
  public void draw(int i, double f) { ... }
}
```

1. Overloaded methods are differentiated by the number and the type of the arguments passed into the method

2. The compiler does not consider return type when differentiating methods, so you cannot declare two methods with the same signature even if they have a different return type.

Garbage Collecting

- Java platform allows you to create as many objects as you want and you don't have to worry about destroying them

- The Java runtime environment deletes objects when it determines that they are no longer being used. This process is called *garbage collection*

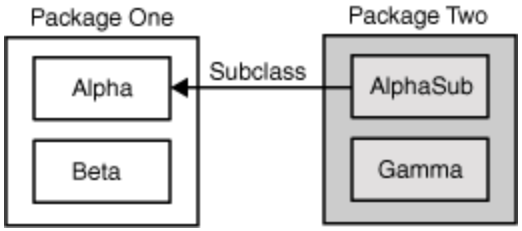
- An object is eligible for garbage collection when there are no more references to that object.

Controlling Access

class - if declared public, the class is visible to all classes. If no modifier exists, the class is visible to all classes within the same package

class “member” access by other classes. A class member can have a modifier of: public, protected, no modifier, or private

Consider the following figure:



The table below shows where the "members" of the class Alpha are visible for each of the access modifiers that can be applied

		Visibility		
Modifier	Alpha	Beta	AlphaSub	Gamma
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Inheritance

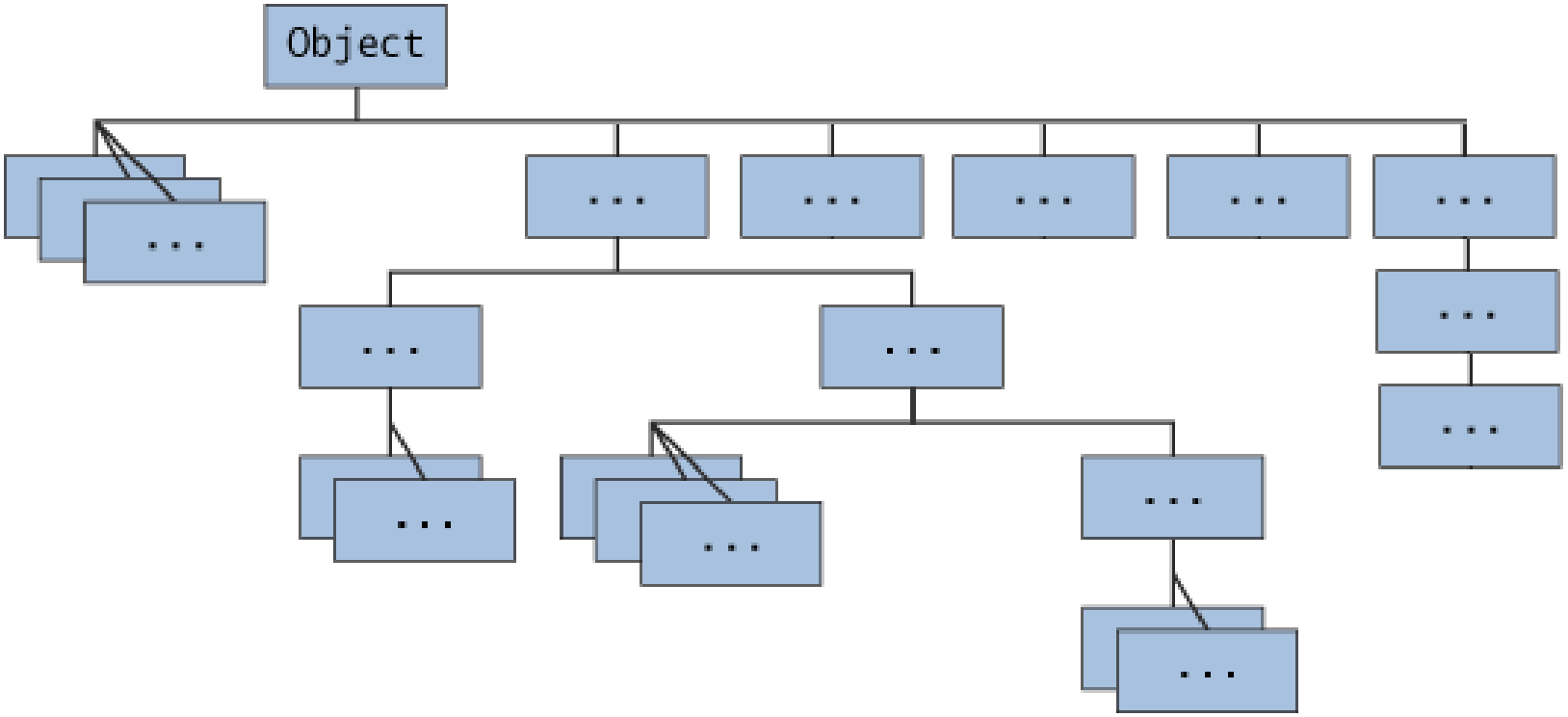
1. A class that is derived from another class is called a *subclass* (also a *derived class*, *extended class*, or *child class*).

2. The class from which the subclass is derived is called a *superclass* (also a *base class* or a *parent class*).

3. Excepting class Object, which has no superclass, every class has one and only one direct superclass (single inheritance).

4. In the absence of any other explicit superclass, every class is implicitly a subclass of Object.

The Object class, defined in the java.lang package, defines and implements behavior common to all classes—including the ones that you write.



Subclass Facts

1. The inherited fields can be used directly (depending on the controlling access), just like any other fields.

2. You can declare a field in the subclass with the same name as the one in the superclass, thus *hiding* it (not recommended).

3. You can declare new fields in the subclass that are not in the superclass.

4. The inherited methods can be used directly as they are.

Subclass Facts

5. You can write a new *instance* method in the subclass that has the same signature as the one in the superclass, thus *overriding* it.

6. You can write a new *static* method in the subclass that has the same signature as the one in the superclass, thus *hiding* it.

7. You can declare new methods in the subclass that are not in the superclass.

8. You can write a subclass constructor that invokes the constructor of the superclass, either implicitly or by using the keyword super.

Polymorphism – is the capability of a method to perform different actions based on the object being acted upon

Q1: Overloading and overriding are two different types of polymorphism. What is the difference between overloading and overriding?

With polymorphism, the programmer can deal in generalities while the execution environment handles the specifics.

Note: If a superclass reference is aimed at a subclass object, it is a compiler error attempting to reference a subclass-only member.

Java Project: Polymorphism

Abstract Class – declares the common functionality (state and behavior)of all classes in a class hierarchy

An abstract class usually contains one or more abstract methods that must be overridden by a subclass

One cannot instantiate an object of an abstract class

```
public abstract class Employee

public abstract double computePay ();
```

Note1: If a class contains an abstract method, the class must be declared abstract

Note2: Any child class must either override the abstract method or declare itself abstract

More important tidbits:

Important packages include:

- java.util
- java.math
- java.io
- java.lang

If you actually want to check if one object is equal to another object, visit these sites:

- http://www.ensta.fr/~diam/java/online/notes-java/data/expressions/22compareobjects.html
- http://www.javaworld.com/community/node/1006
- http://www.technofundo.com/tech/java/equalhash.html
- http://technologiquepanorama.wordpress.com/2009/02/12/use-of-hashcode-and-equals/