Java Class Definitions

CS151

Chris Pollett

Sept. 19, 2005.

Outline

- Creating and Initializing Objects
- Access Fields and Methods
- Method Invocation and Parameter Passing
- Static Fields and Methods
- Constants and Enumerated Types
- Singletons

Creating and Initializing Objects

- There are four different ways of initializing a field of a class:
  - By an explicit initializer:

    public double x=0.0, y= 0.0;
  - Default initial values - if don't initialize get value as described earlier
  - Constructor
  - Initialization Blocks

Will describe these two on next slides

Constructors

- Are special methods with the same name as the class.
- Their return type can be omitted.

```
class Point
{
 public double x,y;
 public Point() { x=0.0; y=0.0;}/* no-arg constructor  */
 public Point(double init_x, double init_y){x=init_x; y=init_y;}
    /* other constructor */
public void move(double dx, double dy){x+=dx; y+=dy;}
}
```

- To create instances of a class one uses the new operator: Point p1 = new Point(); Point p2=new Point(5.0, 3.0);
- If no constructor is given a default empty-bodied one is provided implicitly.
- However, is any constructor is given the default one vanishes.

Initialization Blocks

- You can use a statement block within the class declaration to initialize fields:

```
class MyClass
{
    public int myArray[]= new int[20];
    {
      for(int i=0; i< 20; i++)
      {
                myArray[i] = i;
      }
    }
}
```

- Initialization blocks are executed before any constructor is executed.
- They can be used for code fragments common to all the constructors.

Accessing Fields and Methods

- After an instance has been created fields and methods can be accessed as follows:

  object.method(Parameters)

  object.field

- For example,

  Point p1=new Point();

  double x =p1.x; //access field

  p1.move(5.0,2.0); // access method

Method Invocation

- The body of a method is simply a block statement.
- If the return type of a method is **void** the return statement in the body may not return values and you don't always need a return statement
- If the return type is not void all path-ways through the method body must return a value of the return type. So for instance the following will give a compile error:

  public double product(double x, double y)

  { if (y > 0) return x*y;}
- To fix this need to add a return line for the case when y<=0
- Note one also has to be careful about: double foo(){ double a; return a;} since local variables declared in method bodies are not automatically initialized to their default values.

Parameter Passing

- In Java all parameter methods are passed by value. So modifications to parameters of primitive types inside a method do not affect calling variable values.
- For example:

```
class C{void inc(int i){i++;}
C c =new C();
int k =1;
c.inc(k); //k is unchanged
```

- For parameters of reference type, state of an object can be affected inside a method:

```
class D{void pointInc(Point p){p.x++; p.y++;}
D d =new D();
Point p = new Point(10.0, 5.0);
d.pointInc(p); // p is now (11.0 ,6.0)
```

More on Parameter Passing

- To achieve a similar result for primitive types as we did for reference types, one could use a wrapper class:

```
class IntRef {public int val; public IntRef(int i){val=i;}}
class E {void inc(IntRef i){i.val++;}}
E e = new E();
IntRef k = new IntRef(1);
e.inc(k); // now k.val is 2
```

- These kind of in-out parameters can be useful if we want to pass and return multiple values from a method.

Class Static Fields and Methods

- By default the fields declared in a class are called *instance fields*.
- This means each instance of the class gets a separate copy of these fields. Modification of field values in one instance will not affect those values of other instance
- In contrast class fields are shared by all instances of a class.
- There are also *instance methods* and *class methods*.
- The keyword **static** is used to declare class fields and class methods.
- Class fields are initialized before any instance of the class is created. They live until the program ends.
- One can use default values, explicit initializers, and static initializer blocks to set the value of a class field.
- Class fields should not be initialized in constructors as they affect the value in other instances.

Constants and Enumerated Types

- In Java constants can be defined using final class fields:

```
public MyClass
{
    public final static int MY_CONSTANT_INT = 4;
    //Notice our counvetion on constants being all-caps
}
```

- Java 5 also supports enumerated types:

```
public enum SchoolYear {FROSH, SOPHOMORE,
   JUNIOR, SENIOR};
for (Rank r : Rank.values())
{ System.out.println("rank:" + r);}
```

Singletons

- Some classes are not supposed to have more than one instance at a time. For example, the top-level window of an application.
- One design pattern using static methods which guarantees this is:

```
public class Singleton
{
        static public Singleton getInstance(){return theInstance;}
        protected Singleton() {/* init stuff */}
        //… more code
        private static Singleton theInstance = new Singleton();
}
```

To get the one object use: myInstance = Singleton.getInstance();

this reference

- Instance methods operate on a specific instance of a class.
- This object instance is often referred to as the *receiving instance*.
- Inside instance methods this receiving instance is called by the name **this**.