More About Classes

CS 1025 Computer Science Fundamentals I

Stephen M. Watt

University of Western Ontario
The Story So Far....

• Classes as collections of *fields* and *methods*.
• Methods can access fields, and each other.
• Used to model *objects*.
• *Constructors* are used to allocate new objects, each of which has *its own copy of the fields*.

```java
class Rectangle {
    private int height, width;
    public Rectangle(int h, int w) {
        height = h; width = w;
    }
    public int area() {
        return height*width;
    }
    public void sayArea() {
        System.out.println(“The area is “ + area());
    }
}
```
The Story So Far...

• **Visibility:** Constructors, fields and methods may either be *public* or *private*.

• In the implementation of a class, fields and methods of the same object are used by giving their name. E.g. `height*width`, `area()`

• From outside, only *public* items can be used.

• These are accessed using *object*. *name*  
  E.g. `r1.area()`, `(new Rectangle(3,4)).sayArea()`
Shared Fields

• **Shared items:** static

• A static field is used in common by *all* objects of a class.

• A field that is not static is sometimes called an *instance variable*.

• A method that uses only static fields may be declared static.
Using Shared Items

• From within the implementation of a class, static fields and methods are used just like any other.

```java
class Circle {
    public static double pi = 3.1415926535897932;
    private double radius;

    public Circle(double r) { radius = r; }
    public double area() { return pi*radius*radius; }
    public static void sayPi(){System.out.println("Pi= " + pi);}
}
```

• From outside, these are accessed using `class . name`

```java
class Client {
    public static void main(String[] args) {
        System.out.println("Pi is ", Circle.pi);
        Circle.sayPi();
    }
}
```
Subclasses

- Sometimes a subset of elements have additional guaranteed properties.

- These additional properties may allow additional operations, or even fields, to be defined. E.g.
  - Even Integers can have a “half” operation.
  - Graduate Students can have a “thesisTitle” method.
  - Button Areas can have an “onClick” method.

- For this situation we can construct a subclass that extends a base class.
class Shape {
    private double area = 0;
    public void sayArea() { System.out.println("Area = " + area); }
}
class Triangle extends Shape {
    private double base, height;
    public Triangle(double b, double h) {
        base = b; height = h; area = b*h/2.0;
    }
}
class Rectangle extends Shape {
    private double width, height;
    public Rectangle(double w, double h) {
        width = w; height = h; area = w*h;
    }
    public double diagonal() {
        return Math.sqrt(width*width + height*height);
    }
}
Subclass / Superclass

• The base class might itself be a subclass of another class.
• When we are thinking about a class what it extends, we talk about the class and its superclass or superclasses.

• Values belonging to a subclass have all the fields and methods of the superclass (plus whatever they define). These fields and methods are said to be inherited.

• In Java, if you all classes have Object as their ultimate superclass.
Subclass Values

• An object that belongs to a **subclass** also belongs to the superclass(es). E.g.

```java
Triangle   t = new Triangle(3.0, 4.0);
Rectangle  r = new Rectangle(4.0, 5.0);
Shape      s1 = t, s2 = r;
```

The object referred to by `t` is both a **Triangle** and a **Shape**.

• This allows us to write programs that operate on elements of the base class and then we can use them on elements of any subclass.

```java
private void sayIt(Shape s) { s.sayArea(); }
public static void main(String[] args) {
    sayIt(t);
    sayIt(r);
}
```
Subclass Polymorphism

• Being able to write programs for a common base class (like Shape) that are then used on values in particular subclasses (like Triangle or Rectangle) is the *central idea of object-oriented programming*.

• This is known as *subclass polymorphism*.
Null

• Variables of type `Object` or `[]` (array) may be assigned the value `null`.

• Using this judiciously allows you to write polymorphic programs.
  – E.g. use as “not yet initialized” or “value not found”

• Using this injudiciously will lead to bugs and drive you crazy.
  – E.g. needing to test for null before calling a method.
“Protected” Visibility

• Sometimes we want fields, methods or constructors to be available to implement subclasses, but not to be used by anyone else.

For this, use protected instead of public or private.
“Super” Constructors

• You can use the constructor of the base class to construct that part of the object of the superclass.
• To do this, you call the superclass constructor, using the name `super`, as the first thing you do in the new constructor.

```java
class Rectangle extends Shape {
    protected double width, height;
    public Rectangle(double w, double h) {
        width = w; height = h; area = w*h;
    }
    ...
}
class Square extends Rectangle {
    public Square(double length) {
        super(length, length);
        ...
    }
}
```
Over-Riding

• Sometimes there is a *better way* to compute a quantity in a subclass.
• This may be because there is *more information* or because of the *specialized values*.
• A subclass can over-ride an implementation of a method in a base class in this situation.
• The over-riding (new) method should have the *same meaning* as the over-ridden (old) method.
Example of Over-Riding

class Rectangle extends Shape {
    protected double width, height, area;
    public Rectangle(double w, double h) {
        width = w; height = h; area = w*h;
    }
    public double diagonal() {
        return Math.sqrt(width*width + height*height);
    }
}

class Square extends Rectangle {
    protected static double root2 = 1.4142135623730950;
    public Square(double len) { super(len, len); }
    public double diagonal() { return root2*width; }
}
When You Don’t Want Over-Riding

- Over-riding has a cost:
  - It means the compiler can not readily know at compile time which method definition will be used.
  - An inconsistent over-riding can cause bugs.
- So sometimes you want to make it impossible to over-ride an item.
- You can do this by saying that a field or method is final.
When You Still Need the Old Method

• Sometimes you *do* want to over-ride a method, but you *also* want to make use of the old one.

• In this case, you can access it in the sub-class by calling it from `super`.

Example:

```java
class ObjectWithBorder {
    ...
    public void redraw() {
        drawBorder();
        super.redraw();
    }
    ...
}
```
Example of “final”

class Rectangle extends Shape {
    protected double width, height, area;
    public Rectangle(double w, double h) {
        width = w; height = h; area = w*h;
    }
    public double diagonal() {
        return Math.sqrt(width*width + height*height);
    }
}

class Square extends Rectangle {
    protected final static double root2 = 1.4142135623730950;
    public Square(double len) { super(len, len); }
    public final double diagonal() { return root2*width; }
}
Review

- Objects and classes.
- Fields, methods, constructors.
- Instance vs static fields and methods.
- Visibility: public, private, protected.
- Base class, subclass, superclass.
- Subclass polymorphism.
- Null.
- Over-riding.
- Super constructor and methods.
- Final.