Lecture 21

Inheritance and Polymorphism continued
• Bit of review:
  • What’s superclass? What’s a subclass?
  • What is inherited in a subclass?
  • How do we call superclass constructors?
  • What’s the difference between overloading and overriding methods?
  • Do all classes inherit from a superclass?
Polymorphism

- A class defines a type
- Subtype vs supertype

- A subclass is a specialization of a superclass, so every instance of a class is an instance of the supertype.

- This isn’t the same the other way around

- Think of our Fish and Animal classes.

- Every Fish is an Animal, but every Animal is not a Fish
• So… this means that you can always pass an instance of a subclass to a parameter of its superclass type.

• ???
Polymorphism

Listing 11.5  PolymorphismDemo.java

```java
public class PolymorphismDemo {
    /** Main method */
    public static void main(String[] args) {
        // Display circle and rectangle properties
        displayObject(new CircleFromSimpleGeometricObject(1, "red", false));
        displayObject(new RectangleFromSimpleGeometricObject(1, 1, "black", true));
    }

    /** Display geometric object properties */
    public static void displayObject(SimpleGeometricObject object) {
        System.out.println("Created on " + object.getDateCreated() + ". Color is " + object.getColor());
    }
}
```

Created on Mon Mar 09 19:25:20 EDT 2011. Color is red
Created on Mon Mar 09 19:25:20 EDT 2011. Color is black
public class Polymorphic {

    public static void main(String[] args) {
        Animal myAnimal = new Animal("Bigfoot", "GRUNT");
        Fish myFish = new Fish("Wanda", "glub glub", true);
        Bird myBird = new Bird("Poly", "Poly wants a cracker", true, 3.0);

        printOut(myAnimal);
        printOut(myFish);
        printOut(myBird);
    }

    public static void printOut(Animal animal) {
        System.out.println(animal.getName() + " says " + animal.getSound());
    }
}
Polymorphism

- An object of a subclass can be used wherever its superclass object is used
Dynamic binding

- You can declare objects by their super type, like:
  
  - Animal myFish = new Fish();

- The *declared* type is Animal, the *actual* type is Bird

- So what happens when I run myFish.getSound()?
Dynamic binding

- Dynamic binding means that at runtime, the JVM will pick the method of the *actual* type to run.
Polymorphism

• So why would we use this?!!

• We can use subtypes as if they were the supertypes

• When we override methods, we can write fewer, more generic methods that can handle all the specific classes
Polymorphism

• What happens if we try to run the `myFish.getLivesInOcean()` method?

  • Can’t do it! Because the Animal class doesn’t know about that method.

  • BUT myFish is not only an Animal, its a Fish too.
Casting

• We can cast an object to its *correct* subclass

```java
((Fish) myFish).getLivesInOcean();
```

• But be careful! If you tried this on myBird, we’d get an error
• To check if an instance is the right class, you can use the `instanceof` keyword
if (animal instanceof Fish){
    if (((Fish)animal).getLivesInOcean() ){
        System.out.println(animal.getName() + " lives is the ocean" );
    } else {
        System.out.println(animal.getName() + " does not live in the ocean" );
    }
}
instanceof

• BUT… we should probably use override methods instead (cleaner, extensible)
Object’s equals

- Ripe for overriding!

- In our example, you could return true if the name is the same, or all the data fields are the same, for example
• Don’t be mad....
ArrayList

- Arrays are great for storing data, but they are a bit rigid, inflexible.
  - The size is fixed
  - You can only store items of the same type
- ArrayLists solve this problem
import java.util.ArrayList;

class ArrayListExample {
    public static void main(String[] args) {
        ArrayList<String> streets = new ArrayList<>();
        streets.add("main");
        streets.add("water");
        streets.add("86th");
        streets.add("Bleecker");

        System.out.println(streets.size());
        System.out.println(streets.contains("main"));
        System.out.println(streets.indexOf("main"));
        System.out.println(streets.get(3));

        System.out.println();
        System.out.println(streets.remove("main"));
        System.out.println(streets.size());
        System.out.println(streets.contains("main"));
        System.out.println(streets.indexOf("main"));
        System.out.println(streets.get(2));
    }
}
ArrayList

• They’re great.

• **Note:** you can only store reference types, so no primitive data types to store an ArrayList of ints, for example, you have to do:

```java
ArrayList<Integer> list = new ArrayList<>();
```

• Use a foreach loop to go through a list

```java
for (String s: streets){
    System.out.print(s + " ");
}
```
11.12 Useful Methods of lists

• Copy to and from arrays, sort shuffle

• Sorting and shuffling come from java.util.Collections
Protected Visibility
Modifier

• Remember that private data fields and methods can only be accessed from within their own class. That excludes subclasses!

• Protected data fields and methods can be seen and accessed in subclasses, but not outside of them.
## Protected Visibility Modifier

<table>
<thead>
<tr>
<th>Modifier on members in a class</th>
<th>Accessed from the same class</th>
<th>Accessed from the same package</th>
<th>Accessed from a subclass in a different package</th>
<th>Accessed from a different package</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>protected</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>default (no modifier)</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>private</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
• Remember that a final data field is a constant, meaning it can’t change

• You can do the same for classes and methods so they cannot be extended or overridden!

    public final class A {
    }

• Examples of final classes are Math, String, StringBuilder, etc