Inheritance and Polymorphism

Implementation

Game plan

- Inheritance, Subclassing, Polymorphism
- Yesterday: Basic concepts
- Today: How it’s done in Java

But first…

- Questions from last time?

Inheritance and Java

- Class hierarchy for theatre app

```
public class Performer {
    ...
}
public class Musician extends Performer {
    ...
}
public class Guitarist extends Musician {
    ...
}
```

Inheritance and Java

- To define a class as a subclass, in Java we say that the subclass extends the superclass
Inheritance, Java, and Constructors

- When in a subclass, to call the constructor of your superclass use the `super()` function.

- `super()` must be called if all of the constructors for a superclass have arguments.

- `super()`, if used, must be the first statement in the constructor of the subclass.

```java
public class Musician extends Performer {
    private Instrument myInstrument;
    public Musician (String name, Instrument I) {
        super(name);
        myInstrument = I;
    }
}
```

Inheritance, Java, and Constructors

```java
public class Guitarist extends Musician {
    public Guitarist(String name) {
        super (name, new Guitar());
    }
}
```

The Java Object class

- All classes in Java, whether explicitly stated or not, are subclasses of the java Object class (defined in package java.lang).
- `String Object.toString()` method

```java
toString() example
```

```java
toString example
```

```java
public class Musician extends Performer {
    // constructor
    public Musician (String name) {
    }
}
```

```java
public class Performer {
    private String myName;
    // constructor
    public Performer (String name) {
    }
}
```

```java
public class Guitarist extends Musician {
    public Guitarist(String name) {
        super (name, new Guitar());
    }
}
```

```java
public class Drummer extends Musician {
    public Drummer (String name) {
    }
}
```

```java
public String toString() {
    return "I am a musician";
}
```

```java
Object P = new Performer("foo");
Object M = new Musician("fred", new Guitar());
Object D = new Drummer("keith");
Object G = new Guitarist("barney");
System.out.println(M.toString());
System.out.println(D.toString());
System.out.println(G.toString());
System.out.println(P.toString());
```

Output:
- I am a musician
- I am a drummer
- I am a musician
- Performer@77d163
Accessing superclass methods using super

• You can access any public or protected member of your superclass explicitly using super.

```java
class Drummer extends Musician {
    public String toString()
    {
        return super.toString() + " that plays the drums";
    }
}
```

Accessing superclass methods using super

• You can only specify one level up the hierarchy.

```java
class Drummer extends Musician {
    public String toString()
    {
        // This will cause a compiler error
        return Performer.toString() + " that plays the drums";
    }
}
```

Just a side note

• You cannot override/redeclare methods/variables declared as final in the superclass.

Declaring abstract classes

• To declare a class as abstract, use the abstract keyword when defining the class.

```java
class Performer {
    
}
```

Declaring abstract methods

• To declare a method to be abstract, add the abstract keyword to the method.
• The abstract method in the superclass will have no body defined for it.

```java
class Performer {
    
    public abstract double calculatePay();
}
```

Declaring abstract methods

• If a class has an abstract class, it must be declared as abstract

```java
class Performer {
    
    public abstract double calculatePay();
}
```

• Will fail at compile time
Why use abstract classes

- Abstract classes
  - provide a set of methods that a subclass must implement
  - Subclass Implementations may be vastly different but set of methods are the same

Declaring abstract methods

- Abstract classes
  - A class that has abstract method must be declared as abstract
  - However, you can declare an class that does not have abstract methods to be abstract.

Abstract classes

- Why would one do this?
  - Design purposes – If, in your logical design of your app, a class is so general that it doesn’t make sense to instantiate an object of the class directly
  - Reserve the right to add abstract methods later.

Abstract classes

- FileSystem Object

  - FileSystem
  - Abstract char read()
  - PCFileSystem
  - MacFileSystem
  - SunFileSystem

Abstract classes

- FileSystem Object

  ```java
  public abstract class Instrument {
      private double rentalCost;
      protected Instrument (double cost) {
          rentalCost = cost;
      }
      public double getWeeklyRental () {
          return rentalCost;
      }
  }
  ```

More on Polymorphism

- When you declare a variable with type of a superclass, it can hold an object of type superclass or any class inherited from superclass.
- When sending messages to that object via a method call, you must use the method set available by the superclass.
More on Polymorphism

```java
public abstract class Performer {
    public abstract double calculatePay();
}

public class Musician extends Performer {
    public double calculatePay() {
    }
    public Instrument getInstrument() {
    }
}

Performer P = new Musician("fred");
double pay = P.calculatePay(); // this call is okay
Instrument I = P.getInstrument(); // this call is bad
```

Multiple Inheritance

- In some languages (like C++), it is possible for class to inherit from more than one class (i.e. have more than one superclass).
- This is called multiple inheritance.
- Java does not support multiple inheritance.
- Instead, Java provides a different mechanism: interfaces.

Interfaces

- To define an interface:
  ```java
  public interface InterfaceName {
      // constants defined by the interface
      public static int constant1 = 12;
      public static int constant2 = 20;
      // set of methods that need to be defined by
      // classes implementing this interface
      public void method1();
      public float method2(int arg1);
  }
  ```

- A class can have definitions for all methods of an interface, we declare that class to implement the interface:
  ```java
  public class Foo implements InterfaceName {
      ...
  }
  ```

- A class can have only one superclass, however, it can implement as many interfaces as it likes.
  ```java
  public class Foo2 extends Foo implements InterfaceName2, InterfaceName3 {
      ...
  }
  ```

More on Polymorphism

- Polymorphism works on method arguments as well.
```java
public class Payroll {
    public void addPerformer(Performer P) {
    }
    public static void main(String args[]) {
        Payroll P = new Payroll();
        Guitarist G = new Guitarist("fred");
        P.addPerformer(G);
        ...
    }
}
```
Interface

- **Example:** `java.lang.Comparable`
  - Interface for objects that can compare itself with other objects
  - Used by sorting methods in the `java.util` package.

```java
public interface Comparable {
    public int compareTo(Object O);
}
```

Summary

- `extends`
- `super()`
- `abstract`
- `super again`
- `interface/implements`

- Questions?