FUNDAMENTALS

CS062

DATA STRUCTURES AND ADVANCED PROGRAMMING

3: Inheritance and Interfaces

Alexandra Papoutsaki
LECTURES

Mark Kampe
LABS

Lecture 3: Inheritance and Interfaces

- Inheritance
- Interfaces

Some slides adopted from Algorithms, 4th Edition and Oracle tutorials

Inheritance

- When you want to create a new class and there is already a class that includes some of the code that you want, you can derive your new class from the existing class. → reuse code!
- Central concept in OOP.

- A class that is derived from another is called a subclass or child class.
- The class from which the subclass is derived is called a superclass or parent class.
- Single inheritance: A class can only extend ONE AND ONLY one parent class.
- Multilevel inheritance: A class can extend a class which extends another class etc.

Remember our `Bicycle` class?

```
/**
 * Represents a bicycle
 * @author https://docs.oracle.com/javase/tutorial/java/concepts/class.html
 *
 */
public class Bicycle {

    //instance variables
    private int cadence = 0;
    private int speed = 0;
    private int gear = 1;

    // the Bicycle class has one constructor
    public Bicycle(int startCadence, int startSpeed, int startGear) {
        gear = startGear;
        cadence = startCadence;
        speed = startSpeed;
    }


    public void changeCadence(int newValue) {
         cadence = newValue;
    }

    public void changeGear(int newValue) {
         gear = newValue;
    }

    public void changeSpeed(int change) {
         speed = speed + change;
    }

    public int getCadence() {
         return cadence;
    }

    public void printGear() {
       System.out.println("Gear:" + gear);
    }

    public String toString() {
         return "cadence:" + cadence + " speed:" + speed + " gear:" + gear;
    }
}
```

A MountainBike is specialized type of Bicycle

```
/**
 * Demonstrates concept of inheritance
 * @author https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html
 *
 */
public class MountainBike extends Bicycle {

    // the MountainBike subclass adds one field
    public int seatHeight;

    // the MountainBike subclass has one constructor
    public MountainBike(int startHeight,
                        int startCadence,
                        int startSpeed,
                        int startGear) {
        super(startCadence, startSpeed, startGear);
        seatHeight = startHeight;
    }

    // the MountainBike subclass adds one method
    public void setHeight(int newValue) {
        seatHeight = newValue;
    }
}
```

Inheritance

- The subclass inherits all the public and protected members.
- The inherited fields can be used directly, just like any other fields.
- You can declare a field in the subclass with the same name as one in the superclass, thus hiding it.
  - AVOID
- You can write a new instance method in the subclass that has the same signature as the one in the superclass, thus overriding it.
- You can write a new static method in the subclass that has the same signature as the one in the superclass, thus hiding it.
- You can write a subclass constructor that invokes either implicitly the default constructor of the superclass or by directly invoking it using the keyword `super()`.

Polymorphism

- The ability of an object to take many forms.
- Static Polymorphism: Happens during method overloading, that is more than one method have the same name but different signatures.
  - Also known as Compile-Time Polymorphism, Static binding, Compile-Time binding, Early binding
- Dynamic Polymorphism: Happens during method overriding, that is a method with the same signature exists both in parent and child class. When a parent reference is used to refer to a child object, the method that will be executed with be defined at run-time, therefore will be the child's overridden method.
  - ```
    Student student = new Student();
    Person person = new Student();
    ```
  - Also known as Run-Time Polymorphism, Dynamic binding, Run-Time binding, Late binding

https://medium.com/@shanikae/polymorphism-explained-simply-7294c8deeef7
    ```

Example: Animal

```
public class Animal {
    public int legs = 2;
    public static String species = "Animal";
    public static void testClassMethod() {
        System.out.println("The static method in Animal");
    }
    public void testInstanceMethod() {
        System.out.println("The instance method in Animal");
    }
}
```

Example: Cat

```
public class Cat extends Animal {
    public int legs = 4;
    public static String species = "Cat";
    public static void testClassMethod() {
        System.out.println("The static method in Cat");
    }
    public void testInstanceMethod() {
        System.out.println("The instance method in Cat");
    }
}
```

Hiding vs overriding

```
public static void main(String[] args) {
    Cat myCat = new Cat();
    myCat.testClassMethod(); //invoking a hidden method
    myCat.testInstanceMethod(); //invoking an overridden method
    System.out.println(myCat.legs); //accessing a hidden field
    System.out.println(myCat.species); //accessing a hidden field
}
```

- Output:

```
The static method in Cat
The instance method in Cat
4
Cat
```

WHAT YOU WERE EXPECTING, RIGHT?

Hiding vs overriding

```
public static void main(String[] args) {
    Animal yourCat = new Cat();
    yourCat.testClassMethod(); //invoking a hidden method
    yourCat.testInstanceMethod(); //invoking an overridden method
    System.out.println(yourCat.legs); //accessing a hidden field
    System.out.println(yourCat.species); //accessing a hidden field
}
```

- Output:

```
The static method in Animal
The instance method in Cat
2
Animal
```

???

Hiding vs overriding

- Hiding: For fields (instance+static) and methods (static) the class is determined at compile-time. Here, the compiler sees that `yourCat` is declared as `Animal`.
- Overriding: For instance methods this is determined at run-time. At this point, we know that `yourCat` is of type `Cat`.
- One form of polymorphism (dynamic).
- You will get a compile-time error if you attempt to change an instance method in the superclass to an static method in the subclass and vice-versa.

`super` keyword

- Refers to the direct parent of the subclass.
- `super.variable`: for hidden fields, avoid altogether.
- `super.instanceMethod()`: for overridden methods.
- `super(args)`: to call the constructor of the super class. First line in constructor of subclass.

All classes inherit class `Object`

- Directly if they do not extend any other class, or indirectly as descendants.
- `Object` class has built-in methods that are inherited.
- `public boolean equals (Object other)`
  - Default behavior returns true only if same object.
- `public String toString()`
  - Returns string representation of object – default is hexadecimal.
  - Does not print the string.
  - Typically needs to be overridden to be useful.
- `public int hashCode()`
  - Unique identifier defined so that if `a.equals(b)` then `a`, `b` have same hashCode.

`final` keyword

- Variable: only assigned once in its declaration or in constructor – its value cannot be changed after initialization.
  - E.g., `static final PI = 3.14;`
- Method: cannot be overridden by subclass.
- Class: cannot be extended.

Nested classes

- Java allows us to define a class (nested) within another class (outer).

```
class OuterClass {
   class NestedClass {
   }
}
```

- Nested classes are divided into two categories: static nested class and non-static which are called inner classes.

```
class OuterClass {
   static class StaticNestedClass {
   }
   class InnerClass {
   }
}
```

Example

```
/**
 * Demonstrates concept of inner class
 * @author https://docs.oracle.com/javase/tutorial/java/javaOO/examples/DataStructure.java
 *
 */

public class DataStructure {

    private final static int SIZE = 15;
    private int[] arrayOfInts = new int[SIZE];

    public DataStructure() {
      for (int i = 0; i < SIZE; i++) {
            arrayOfInts[i] = i;
       }
    }

    public void printEven() {
        DataStructureIterator iterator = this.new EvenIterator();
        while (iterator.hasNext()) {
            System.out.print(iterator.next() + " ");
        }
   }

    interface DataStructureIterator extends java.util.Iterator<Integer> { }

    // Inner class implements the DataStructureIterator interface, which extends the Iterator<Integer> interface
    private class EvenIterator implements DataStructureIterator {

       private int nextIndex = 0;

        public boolean hasNext() {
            return (nextIndex <= SIZE - 1);
        }

        public Integer next() {
            Integer retValue = Integer.valueOf(arrayOfInts[nextIndex]);
            nextIndex += 2;
            return retValue;
        }
    }

    public static void main(String s[]) {
        DataStructure ds = new DataStructure();
        ds.printEven();
    }
}
```

Practice Time

```
public class ClassA {
    public void methodOne(int i) {
    }
    public void methodTwo(int i) {
    }
    public static void methodThree(int i) {
    }
    public static void methodFour(int i) {
    }
}

public class ClassB extends ClassA {
    public static void methodOne(int i) {
    }
    public void methodTwo(int i) {
    }
    public void methodThree(int i) {
    }
    public static void methodFour(int i) {
    }
}
```

1. Which method *overrides* a method in the superclass?

2. Which method *hides* a method in the superclass?

3. What do the other methods do?

Answers

1. `methodTwo`.

2. `methodFour`.

3. They cause compile-time errors.
   `methodOne`: "`This static method cannot hide the instance method from ClassA`".
   `methodThree`: "`This instance method cannot override the static method from ClassA`".

Lecture 3: Inheritance and Interfaces

- Inheritance
- Interfaces

Interfaces

- Contracts of what a class must do, not how to do it, abstracting from implementation.
- Central concept in OOP.
- In Java, an interface is a reference type (like a class), that contains only constants, method signatures, default methods, and static methods.
- A class that implements an interface is obliged to implement its methods.
- Method bodies exist only for default methods and static methods.
- Interfaces cannot be instantiated (no new keyword). They can only be implemented by classes or extended by other interfaces.

Example

```
public interface Moveable{
  int turn(Direction direction, double radius, double speed);

  default int stop(){
    speed=0;
  }
}
```

```
public class Car implements Moveable{
  int turn(Direction direction, double radius, double speed){
    //code goes here
  }
}

public class Bicycle implements Moveable{
  int turn(Direction direction, double radius, double speed){
    //code goes here
  }
}
```

Interfaces

- A class can implement multiple interfaces.
  - `class A implements Interface1, Interface2{…}`
- An interface can extend multiple interfaces.
  - `public interface GroupedInterface extends Interface1,Interface2{…}`

Lecture 3: Inheritance and Interfaces

- Inheritance
- Interfaces

Readings:

- Oracle's guides:
  - Interfaces and Inheritance: https://docs.oracle.com/javase/tutorial/java/IandI/index.html
- Textbook:
  - Chapter 1.2 (Pages 100-104)

Practice Problems:

- If you want more practice with hiding vs overriding: http://javabypatel.blogspot.com/2016/04/java-interview-questions.html