Inheritance, Polymorphism, and Overriding

Weiss, from the beginning of chapter 4 through section 4.1.8 (pages 109–121)
Inheritance

• You want to create a new class.
• This class requires some commonly used code that another class already has it.
• You can derive the new class from the existing class.
  – Reuse fields and methods of the existing class.
• Derived class: A class that is derived from another class.
• Base class: A class that from which the derived class inherits.
Example of Inheritance

```java
class Bicycle {
    private int gear;
    protected int speed;
    private int cadence;

    public Bicycle(int startCadence, int startSpeed, int startGear) {
        gear = startGear;
        speed = startSpeed;
        cadence = startCadence;
    }
    public void setGear(int newValue) {
        gear = newValue;
    }
    public void applyBrake(int decrement) {
        speed -= decrement;
    }
    public void setCadence(int newValue) {
        cadence = newValue;
    }
}
```
Example of Inheritance

```java
public class MountainBike extends Bicycle {
    private int seatHeight;

    public MountainBike (int startHeight, int startSpeed, int startGear) {
        super(startSpeed, startGear);
        seatHeight = startHeight;
    }

    public void setHeight(int newValue) {
        seatHeight = newValue;
    }
}
```

Derived class inherits the public and protected members of the base class. Derived class inherits the package-visible members of the base class if the derived and base class are in the same package.
Private Members of Base Class

- Access **private members** through the **public** or **protected** methods of the base class.
- These methods are called **accessors** and **mutators**.

```java
// Bicycle class
public void setCadence(int newValue) {
    cadence = newValue;
}

// Bicycle class
public void getCadence() {
    return cadence;
}
```

```java
// MountainBikeClass
public void adjustSeatWithCadence() {
    seatHeight = seatHeight + getCadence();
}
```
Method Overriding

• Derived class inherits a method from base class and modifies its functionality.
• Both methods have the same signature (name, type and number of arguments)
• The return type of method in the derived class is either the same type of the subtype of the base class. This subtype is called covariant return type.
• Visibility modifier of an overridden method in derived class can be the same or more than base class.
Method Overriding (example)

```java
public class Animal {

    private int age;

    public Animal(int age) {
        this.age = age;
    }

    public int getAge() {
        return age;
    }

    public void makeSound() {
        System.out.println("the animal makes sound");
    }

    public static void recordSound(Animal obj) {
    }
}
```
Method Overriding (example)

```java
public class Dog extends Animal {

    public Dog(int age) {
        super(age);
    }

    @Override
    public void makeSound() {
        System.out.println("the Dog barks");
    }
}
```
Method Overriding (example)

```java
public class TestInheritance {
    public static void main(String[] args) {
        Animal animal = new Animal(5);
        Dog dog = new Dog(20);

        animal.makeSound(); // the animal makes sound
        dog.makeSound();     // the Dog barks
    }
}
```
Type Compatibility

• Every object of derived class is also an object of base class.
  – Every dog is an animal (IS-A relationship)
  – The reverse is not correct.
• Derived class calls inherited methods (code reuse).
• Type compatibility applies to parameter passing.

```
static type  → Animal myAnimal = dog;
             myAnimal.getAge();
```

• Type compatibility applies to parameter passing.

```
Animal.recordSound(animal);
Animal.recordSound(dog);
```
Polymorphism

• An organism can have many different forms.
• In Java, derived classes of a base class define their own behaviors for the inherited methods.
Polymorphism

```java
public class Cat extends Animal {
    public Cat(int age) {
        super(age);
    }

    @Override
    public void makeSound(){
        System.out.println("the Cat meows");
    }
}
```
Polymorphism

```java
Animal animal = new Dog(20);
animal.makeSound();

animal = new Cat(10);
animal.makeSound();
```

the Dog barks

the Cat meows
Super Keyword

• A derived class uses `super` when
  ▪ Calling an overridden method of its super class (base class)

```java
@Override
public void makeSound(){
    super.makeSound();
    System.out.println("the Dog barks");
}
```

• Calling the constructor of base class (see slide 12)
  ➢ `super()` needs to be the first line inside the constructor of derived class.
  ➢ If base class has a constructor with `parameters`, derived class needs to call `super(parameters)`. 

the animal makes sound
the Dog barks
final keyword

• Declare a class as final.
• The final class is not inherited.
• The String class is final.

• The final method is not overridden.
  – You do not want to let others change the default implementation of methods.
Multiple Inheritance

• A class inherits from multiple classes.
• Java does not allow multiple inheritance.
• Suppose class A inherits classes B and C. B and C have a field with the identical name.
  – Does A get two copies of the field? It causes conflicting names.
• Suppose B and C have a method with the same signature?
  – Which method is called? It causes conflicting implementations.
Downcast

```
public void meow() {
    System.out.println("Meow!");
}
```

`meow()` method is defined in `Cat` class

```
Animal myCat = new Cat();
myCat.meow();
```

Compilation error:
The `static` type of `myCat` is `Animal`.

Change the static type from base class to one of its derived classes.
Java Platform Class Hierarchy

• The **Object** class defines and implements behavior that are common to all classes.
• The Object class is the most general class. Classes near the **bottom** of hierarchy provide specific behavior.