



Coding-bootcamps.com

Introduction to Java Programming Language

By Jim Sullivan
from [Coding Bootcamps](https://coding-bootcamps.com)



Coding Bootcamps

Defining Classes Using Inheritance

Session 8



Brief Recap

- Re-hash what we've covered in our last class

Objective - Discuss The Following

- Superclasses and Subclasses
- Advantages of Using Inheritance
- protected Class Members
- Constructors in Subclasses

Materials

- These Powerpoint Slides

OOP - Inheritance

- Allows programmers to create new classes based on existing classes
- Methods and attributes from the parent class are inherited by the newly-created class
- New methods and attributes can be created in the new class, but doesn't affect the parent class.

OOP - Inheritance

- Example

```
public class Bicycle {  
  
    public int cadence;  
    public int gear;  
    public int speed;  
  
    public Bicycle(int startCadence, int startSpeed, int startGear) {  
        gear = startGear;  
        cadence = startCadence;  
        speed = startSpeed;  
    }  
  
    public void setCadence(int newValue) {  
        cadence = newValue;  
    }  
  
    public void setGear(int newValue) {  
        gear = newValue;  
    }  
  
    public void applyBrake(int decrement) {  
        speed -= decrement;  
    }  
  
    public void speedUp(int increment) {  
        speed += increment;  
    }  
  
}
```

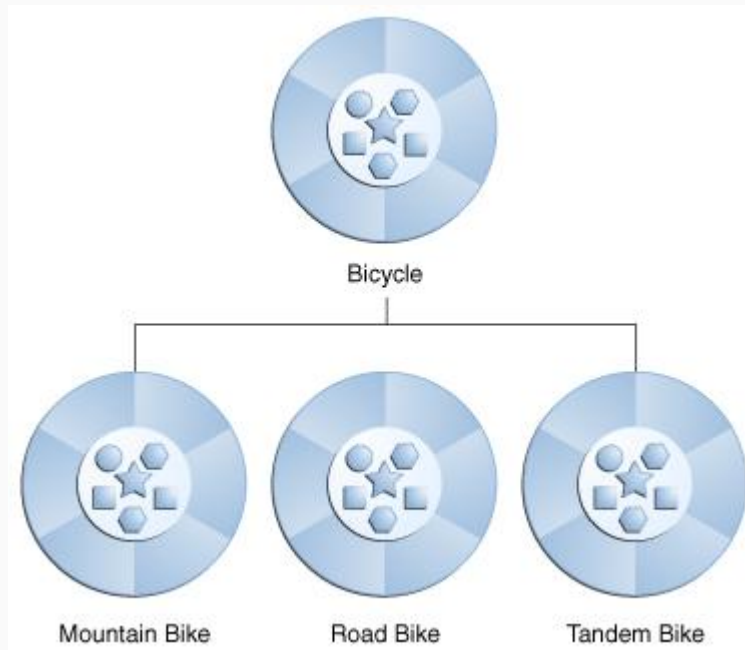
OOP - Inheritance - Example

```
public class MountainBike extends Bicycle {  
  
    // the MountainBike subclass has  
    // 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 has  
    // one method  
    public void setHeight(int newValue) {  
        seatHeight = newValue;  
    }  
}
```


OOP - Inheritance

- Mountain Bike inherited all the fields and methods of Bicycle
- It also added the field 'seatHeight'
- It also added the method to adjust the seat height

OOP - Inheritance



OOP - Abstract Classes

- Abstract classes are classes that contain one or more abstract methods.
- An abstract method is a method that is declared, but contains no implementation.
- Abstract classes may not be instantiated, and require subclasses to provide implementations for the abstract methods.

OOP - Abstract Classes (Example)

```
public abstract Animal
{
    public void eat(Food food)
    {
        // do something with food....
    }

    public void sleep(int hours)
    {
        try
        {
            // 1000 milliseconds * 60 seconds * 60 minutes * hours
            Thread.sleep ( 1000 * 60 * 60 * hours);
        }
        catch (InterruptedException ie) { /* ignore */ }
    }

    public abstract void makeNoise();
}
```

OOP - Abstract Classes (Continued)

```
public Dog extends Animal
{
    public void makeNoise() { System.out.println ("Bark! Bark!"); }
}

public Cow extends Animal
{
    public void makeNoise() { System.out.println ("Moo! Moo!"); }
}
```

OOP - protected Class Members

- Java provides a number of access modifiers to set access levels for classes, variables, methods, and constructors.
 - Visible to the package, the default. No modifiers are needed
 - Visible to the class only (private).
 - Visible to the world (public).
 - Visible to the package and all subclasses (protected).

OOP - protected Class Members

- **Default Access Modifier - No Keyword**
 - Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc
 - A variable or method declared without any access control modifier is available to any other class in the same package. The fields in an interface are implicitly public static final and the methods in an interface are by default public.

```
String version = "1.5.1";  
  
boolean processOrder() {  
    return true;  
}
```

OOP - protected Class Members

- Private Access Modifier - Private

- Methods, variables, and constructors that are declared private can only be accessed within the declared class itself.
- Methods, variables, and constructors that are declared private can only be accessed within the declared class itself.
- Variables that are declared private can be accessed outside the class, if public getter methods are present in the class.
- Using the private modifier is the main way that an object encapsulates itself and hides data from the outside world.

```
public class Logger {  
    private String format;  
  
    public String getFormat() {  
        return this.format;  
    }  
  
    public void setFormat(String  
format) {  
        this.format = format;  
    }  
}
```


OOP - protected Class Members

- **Public Access Modifier - Public**

- A class, method, constructor, interface, etc. declared public can be accessed from any other class. Therefore, fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe.
- However, if the public class we are trying to access is in a different package, then the public class still needs to be imported. Because of class inheritance, all public methods and variables of a class are inherited by its subclasses.

```
public static void main(String[]  
arguments) {  
    // ...  
}
```

OOP - protected Class Members

- Protected Access Modifier - Protected
 - Variables, methods, and constructors, which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class.
 - The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected
 - Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.

```
class AudioPlayer {  
    protected boolean openSpeaker(Speaker sp)  
    {  
        // implementation details  
    }  
}  
  
class StreamingAudioPlayer {  
    boolean openSpeaker(Speaker sp) {  
        // implementation details  
    }  
}
```

Assignment

For this homework assignment, you will be writing software in support of a Dessert Shoppe which sells candy by the pound, cookies by the dozen, ice cream, and sundaes (ice cream with a topping). Your software will be used for the checkout system.

To do this, you will implement an inheritance hierarchy of classes derived from a **DessertItem** *abstract* superclass.

The **Candy**, **Cookie**, and **IceCream** classes will be derived from the **DessertItem** class.

The **Sundae** class will be derived from the **IceCream** class.

You will also write a **Checkout** class which maintains a list (Set) of **DessertItem**'s

The **Checkout** class, provides methods to enter dessert items into the cash register, clear the cash register, get the number of items, get the total cost of the items (before tax), get the total tax for the items, and get a String representing a receipt for the dessert items.

Summary

Live private coaching sessions for Java

- [Private tutoring sessions for software design and engineering- Weekly and monthly plans](#)
- [Java programming language- Private tutoring sessions](#)



Coding-bootcamps.com

Thank You



Coding
Bootcamps