



Coding-bootcamps.com

Introduction to Java Programming Language

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



Deeper Into Classes and Objects

Session 7



Brief Recap

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

Objective - Discuss The Following

- Controlling Access to Class Members
- Referencing the Current Object Using this
- Overloading Constructors
- Default and No-Argument Constructors
- Composition of Classes
- Garbage Collection and Destructors
- The finalize Method
- Static Class Members

Materials

- These Powerpoint Slides

OOP - Encapsulation

- Encapsulation is used for controlling access to class members
- Get's and Set's are the actual access points of the instance of this class. Therefore, any class that wants to get this data needs to use the 'getter'
- Likewise, if an attribute was private and no 'getter' was present, this data would be hidden

OOP - Encapsulation

- Example

```
public class EncapTest {  
    private String name;  
    private String idNum;  
    private int age;  
  
    public int getAge() {  
        return age;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public String getIdNum() {  
        return idNum;  
    }  
  
    public void setAge( int newAge) {  
        age = newAge;  
    }  
  
    public void setName(String newName) {  
        name = newName;  
    }  
  
    public void setIdNum( String newId) {  
        idNum = newId;  
    }  
}
```

OOP - Encapsulation - Example

```
/* File name : RunEncap.java */
public class RunEncap {

    public static void main(String args[]) {
        EncapTest encap = new EncapTest();
        encap.setName("James");
        encap.setAge(20);
        encap.setIdNum("12343ms");

        System.out.print("Name : " + encap.getName() + " Age : " +
            encap.getAge());
    }
}
```


The 'this' keyword

- Within an instance method or a constructor, this is a reference to the current object
 - The object whose method or constructor is being called.
 - You can refer to any member of the current object from within an instance method or a constructor by using this.
- The most common reason for using the this keyword is because a field is shadowed by a method or constructor parameter.

The 'this' keyword

```
public class Point {  
    public int x = 0;  
    public int y = 0;  
  
    //constructor  
    public Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
}
```

Using 'this' with a Constructor

- From within a constructor, you can also use the `this` keyword to call another constructor in the same class.
 - Doing so is called an explicit constructor invocation.
 - Here's another `Rectangle` class, with a different implementation from the one in the `Objects` section.

Using 'this' with a Constructor

```
public class Rectangle {  
    private int x, y;  
    private int width, height;  
  
    public Rectangle() {  
        this(0, 0, 1, 1);  
    }  
    public Rectangle(int width, int height) {  
        this(0, 0, width, height);  
    }  
    public Rectangle(int x, int y, int width, int height) {  
        this.x = x;  
        this.y = y;  
        this.width = width;  
        this.height = height;  
    }  
    ...  
}
```

Overloading Constructors

- It's common to overload constructors -
 - Define multiple constructors which differ in number and/or types of parameters.

Overloading Constructors

```
public class Perimeter
{
    public Perimeter(int
x)                                // II
    {
        System.out.println("Circle perimeter: " + 2*Math.PI*x);
    }
    public Perimeter(int x, int
y)                                // III
    {
        System.out.println("Rectangle perimeter: " + 2*(x+y));
    }
    public static void main(String args[])
    {
        Perimeter p1 = new Perimeter(10);
        Perimeter p2 = new Perimeter(10, 20);    // III
    }
}
```

Default And No Argument

- A constructor without parameters is called as "default constructor" or "no-args constructor".
- It is called default because if the programmer does not write one, the compiler will create and supply one.
- The default constructor supplied does not have any functionality (output).

Overloading Constructors

```
public class Demo
{
    public Demo()
    {
        System.out.println("From default
constructor");
    }
    public static void main(String
args[])
    {
        Demo d1 = new Demo();
        Demo d2 = new Demo();
    }
}
```


Composition of Classes

- Composition is the design technique to implement has-a relationship in classes.
- We can use java inheritance or Object composition for code reuse.
- Composition in java is achieved by using instance variables that refers to other objects. For example, a Person has a Job.

Composition

- Example 1

```
public class Employee {
    private String name;
    private String address;
    private int number;

    public Employee(String name, String address, int number) {
        System.out.println("Constructing an Employee");
        this.name = name;
        this.address = address;
        this.number = number;
    }

    public void mailCheck() {
        System.out.println("Mailing a check to " + this.name + " " +
this.address);
    }

    public String toString() {
        return name + " " + address + " " + number;
    }

    public String getName() {
        return name;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String newAddress) {
        address = newAddress;
    }

    public int getNumber() {
        return number;
    }
}
```

Composition

- Example 2

```
public class Salary extends Employee {
    private double salary; // Annual salary

    public Salary(String name, String address, int number, double
salary) {
        super(name, address, number);
        setSalary(salary);
    }

    public void mailCheck() {
        System.out.println("Within mailCheck of Salary class ");
        System.out.println("Mailing check to " + getName()
+ " with salary " + salary);
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double newSalary) {
        if(newSalary >= 0.0) {
            salary = newSalary;
        }
    }

    public double computePay() {
        System.out.println("Computing salary pay for " + getName());
        return salary/52;
    }
}
```

Memory Management

- You all know that an object is created in the memory using new operator.
- The constructor is used to initialize the properties of that object.
- When an object is no more required, it must be removed from the memory so that that memory can be reused for other objects.
- Removing unwanted objects or abandoned objects from the memory is called garbage collection (GC)

Garbage Collection and Destructors

- Java Memory Management, with its built-in garbage collection, is one of the language's finest achievements.
- It allows developers to create new objects without worrying explicitly about memory allocation and deallocation, because the garbage collector automatically reclaims memory for reuse.

The finalize Method

- The `java.lang.Object.finalize()` is called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
- A subclass overrides the finalize method to dispose of system resources or to perform other cleanup.

Static Class Members

- When a number of objects are created from the same class blueprint, they each have their own distinct copies of instance variables.
- In an example, consider a Bicycle class.
 - The instance variables could be things like cadence, gear, and speed.
 - Each Bicycle object has its own values for these variables, stored in different memory locations.

Static Class Members

- Sometimes, you want to have variables that are common to all objects.
 - This is accomplished with the static modifier.
- Fields that have the static modifier in their declaration are called static fields or class variables.
 - They are associated with the class, rather than with any object.
 - Every instance of the class shares a class variable, which is in one fixed location in memory.
 - Any object can change the value of a class variable, but class variables can also be manipulated without creating an instance of the class.

Static Class Members

```
public class Bicycle {  
  
    private int cadence;  
    private int gear;  
    private int speed;  
  
    // add an instance variable for the object ID  
    private int id;  
  
    // add a class variable for the  
    // number of Bicycle objects instantiated  
    private static int numberOfBicycles = 0;  
    ...  
}
```

Static Class Members

- Class variables are referenced by the class name itself, as in

`Bicycle.numberOfBicycles`

Homework

- Write a simple Vehicle class that has fields for (at least) current speed, current direction in degrees, and owner name.
 - Add a static field to your Vehicle class for the highest Vehicle Identification Number issued, and a non-static field that holds each vehicle's ID number.
 - Write a main method for your Vehicle class that creates a few vehicles and prints out their field values. Note that you can also write a separate tester program as well.
 - Add two constructors to Vehicle. A no-arg constructor and one that takes an initial owner's name. Modify the tester program from the previous step and test your design.
 - Make the fields in your Vehicle class private, and add accessor methods for the fields. Which fields should have methods to change them and which should not?
 - Add a static method to Vehicle that returns the highest identification number used so far.

Summary

- What we learned in this session

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