

# Java Programming Language SE - 6

## Module 2 : Object-Oriented Programming

Team Emertxe

**ORACLE®**

**Certified Professional**

Java SE 6 Programmer



# Objectives

- Define object modeling concepts: abstraction, encapsulation and packages
- Discuss why you can reuse Java technology application code
- Define class, member, attribute, method, constructor, and package
- Use the access modifiers private and public as appropriate for the guidelines of encapsulation
- Invoke a method on a particular object
- Use the Java technology application programming interface (API) online documentation



# Relevance

- What is your understanding of software analysis and design?
- What is your understanding of design and code reuse?
- What features does the Java programming language possess that make it an object-oriented language?
- Define the term object-oriented.

Software Engineering

# The Analysis and Design Phase



- Analysis describes what the system needs to do:
  - Modeling the real-world, including actors and activities, objects, and behaviors
- Design describes how the system does it:
  - Modeling the relationships and interactions between objects and actors in the system
  - Finding useful abstractions to help simplify the problem or solution

# Abstraction



- Functions - Write an algorithm once to be used in many situations
- Objects - Group a related set of attributes and behaviors into a class
- Frameworks and APIs - Large groups of objects that support a complex activity; Frameworks can be used as is or be modified to extend the basic behavior

# Classes as Blueprints for Objects



- In manufacturing, a blueprint describes a device from which many physical devices are constructed.
- In software, a class is a description of an object:
  - A class describes the data that each object includes.
  - A class describes the behaviors that each object exhibits.

# Classes as Blueprints for Objects



- In Java technology, classes support three key features of object-oriented programming (OOP):
  - Encapsulation
  - Inheritance
  - Polymorphism



# Declaring Java Technology Classes

- Basic syntax of a Java class:

```
<modifier>* class <class_name> {  
  <attribute_declaration>*  
  <constructor_declaration>*  
  <method_declaration>*  
}
```

# Declaring Java Technology Classes

```
public class Vehicle {  
    private double maxLoad;  
    public void setMaxLoad(double value) {  
        maxLoad = value;  
    }  
}
```

# Declaring Attributes

- Basic syntax of an attribute:

`<modifier>* <type> <name> [ = <initial_value>];`

- Examples:

```
public class Foo {  
    private int x;  
    private float y = 10000.0F;  
    private String name = "Bates Motel";  
}
```

# Declaring Methods

Basic syntax of a method:

```
<modifier>* <return_type> <name> ( <argument>* ) {  
<statement>*  
}
```

# Declaring Methods

```
public class Dog {  
    private int weight;  
    public int getWeight() {  
        return weight;  
    }  
    public void setWeight(int newWeight) {  
        if ( newWeight > 0 ) {  
            weight = newWeight;  
        }  
    }  
}
```

# Accessing Object Members

- The dot notation is: <object>.<member>
- This is used to access object members, including attributes and methods.
- Examples of dot notation are:

`d.setWeight(42);`

`d.weight = 42; // only permissible if weight is public`

# Information Hiding

The problem:

MyDate
+day : int
+month : int
+year : int

The solution:

MyDate
-day : int
-month : int
-year : int
+getDay() : int
+getMonth() : int
+getYear() : int
+setDay(int) : boolean
+setMonth(int) : boolean
+setYear(int) : boolean

Verify days in month

# Encapsulation

- Hides the implementation details of a class
- Forces the user to use an interface to access data
- Makes the code more maintainable

<b>MyDate</b>
<b>-date : long</b>
<b>+getDay() : int</b> <b>+getMonth() : int</b> <b>+getYear() : int</b> <b>+setDay(int) : boolean</b> <b>+setMonth(int) : boolean</b> <b>+setYear(int) : boolean</b> <b>-isDayValid(int) : boolean</b>



# Declaring Constructors

- Basic syntax of a constructor:

```
[<modifier>] <class_name> ( <argument>* ) {  
  <statement>*  
}
```

- Example:

```
public class Dog {  
  private int weight;  
  public Dog() {  
    weight = 42;  
  }  
}
```

# The Default Constructor

- There is always at least one constructor in every class.
- If the writer does not supply any constructors, the default constructor is present automatically:
  - The default constructor takes no arguments
  - The default constructor body is empty
- The default enables you to create object instances with `new Xxx()` without having to write a constructor.

# Source File Layout

- Basic syntax of a Java source file is:

[<package\_declaration>]

<import\_declaration>\*

<class\_declaration>+

# Source File Layout

```
package shipping.reports;

import shipping.domain.*;
import java.util.List;
import java.io.*;

public class VehicleCapacityReport {
    private List vehicles;
    public void generateReport(Writer output) {...}
}
```

# Software Packages

- Packages help manage large software systems.
- Packages can contain classes and sub-packages.

# The package Statement

- Basic syntax of the package statement is: package
  - `<top_pkg_name>[.<sub_pkg_name>]*;`
- Examples of the statement are:
  - `package shipping.gui.reportscreens;`
- Specify the package declaration at the beginning of the source file.
- Only one package declaration per source file.
- If no package is declared, then the class is placed into the default package.
- Package names must be hierarchical and separated by dots.

# The import Statement

- Basic syntax of the import statement is:

```
Import<pkg_name>[.<sub_pkg_name>]*.<class_name>;
```

OR

```
import<pkg_name>[.<sub_pkg_name>]*.*;
```

- Examples of the statement are:

```
import java.util.List;
```

```
import java.io.*;
```

```
import shipping.gui.reportscreens.*;
```

# The import Statement

The import statement does the following:

- Precedes all class declarations
- Tells the compiler where to find classes



# Compiling Using the -d Option

```
cd JavaProjects/ShippingPrj/src
```

```
javac -d ../classes shipping/domain/*.java
```

# Recap

- Class - The source-code blueprint for a run-time object
- Object - An instance of a class;  
also known as instance
- Attribute - A data element of an object;  
also known as data member, instance variable, and data field
- Method - A behavioral element of an object;  
also known as algorithm, function, and procedure



# Recap



- Constructor - A method-like construct used to initialize a new object
- Package - A grouping of classes and sub-packages



- A set of Hypertext Markup Language (HTML) files provides information about the API.
- A frame describes a package and contains hyperlinks to information describing each class in that package.
- A class document includes the class hierarchy, a description of the class, a list of member variables, a list of constructors, and so on.

# Java Technology API Documentation

The screenshot shows a web browser window titled "Object (Java 2 Platform SE 5.0) – Web Browser". The address bar displays "file:Wopt/java/docs/api/index.html". The browser's menu bar includes "File", "Edit", "View", "Go", "Bookmarks", "Tools", "Window", and "Help". The bookmarks bar shows "Java Store", "Apple Training", "Admin", "SES", "Java", "Tomcat", "Java Web Tech", and "Personal".

The main content area is divided into a left sidebar and a main pane. The sidebar, titled "Java™ 2 Platform Standard Ed. 5.0", contains a list of packages and classes. The main pane displays the "Class Object" page for the "java.lang" package. The page includes a navigation bar with links for "Overview", "Package", "Class", "Use", "Tree", "Deprecated", "Index", and "Help". Below this, there are links for "PREV CLASS", "NEXT CLASS", "FRAMES", "NO FRAMES", "SUMMARY: NESTED | FIELD | CONSTRUCTOR | METHOD", and "DETAIL: FIELD | CONSTRUCTOR | METHOD".

The main content of the "Class Object" page is as follows:

**Class Object**  
java.lang.Object

public class Object

Class Object is the root of the class hierarchy. Every class has Object as a superclass. All objects, including arrays, implement the methods of this class.

Since: JDK1.0  
See Also: [Class](#)

**Constructor Summary**

	<a href="#">Object()</a>
--	--------------------------

**Method Summary**

protected <a href="#">Object</a>	<a href="#">clone()</a> Creates and returns a copy of this object.
boolean	<a href="#">equals(Object obj)</a> Indicates whether some other object is "equal to" this one.
protected void	<a href="#">finalize()</a> Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.

# Stay connected

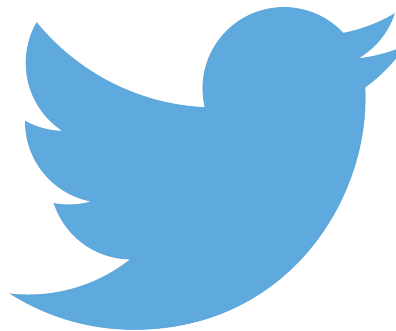


**About us:** Emertxe is India's one of the top IT finishing schools & self learning kits provider. Our primary focus is on Embedded with diversification focus on Java, Oracle and Android areas

Emertxe Information Technologies,  
No-1, 9th Cross, 5th Main,  
Jayamahal Extension,  
Bangalore, Karnataka 560046  
T: +91 80 6562 9666  
E: [training@emertxe.com](mailto:training@emertxe.com)



<https://www.facebook.com/Emertxe>



<https://twitter.com/EmertxeTweet>



**slideshare**  
Present Yourself

<https://www.slideshare.net/EmertxeSlides>



Thank You