Core Java 8 and Development Tools

Lesson 04 : Classes and Objects

Capgemini

Lesson Objectives

After completing this lesson, participants will be able to:

- Define classes and objects
- Create Packages
- Work with Access Specifiers
- Define Constructors
- understand **this** reference
- Understand memory management in java
- use **static** keyword
- Declaring and using Enum
- Best Practices

© 2017 Capgemini. All rights reserved.

4.1 : Classes and Objects
Classes and Objects

Class:

- A template for multiple objects with similar features
- A blueprint or the definition of objects

Object:

- Instance of a class
- Concrete representation of class

```
class < class_name>
{
    type var1; …
    Type method_name(arguments )
     {
        body
      } …
} //class ends
```


© 2017 Capgemini. All rights reserved.

4.1 : Classes and Objects
Introduction to Classes

A class may consist the following elements:

- Fields
- Methods
- Constructors
- Initializers

© 2017 Capgemini. All rights reserved.

4.1 : Classes and Objects
Introduction to Classes

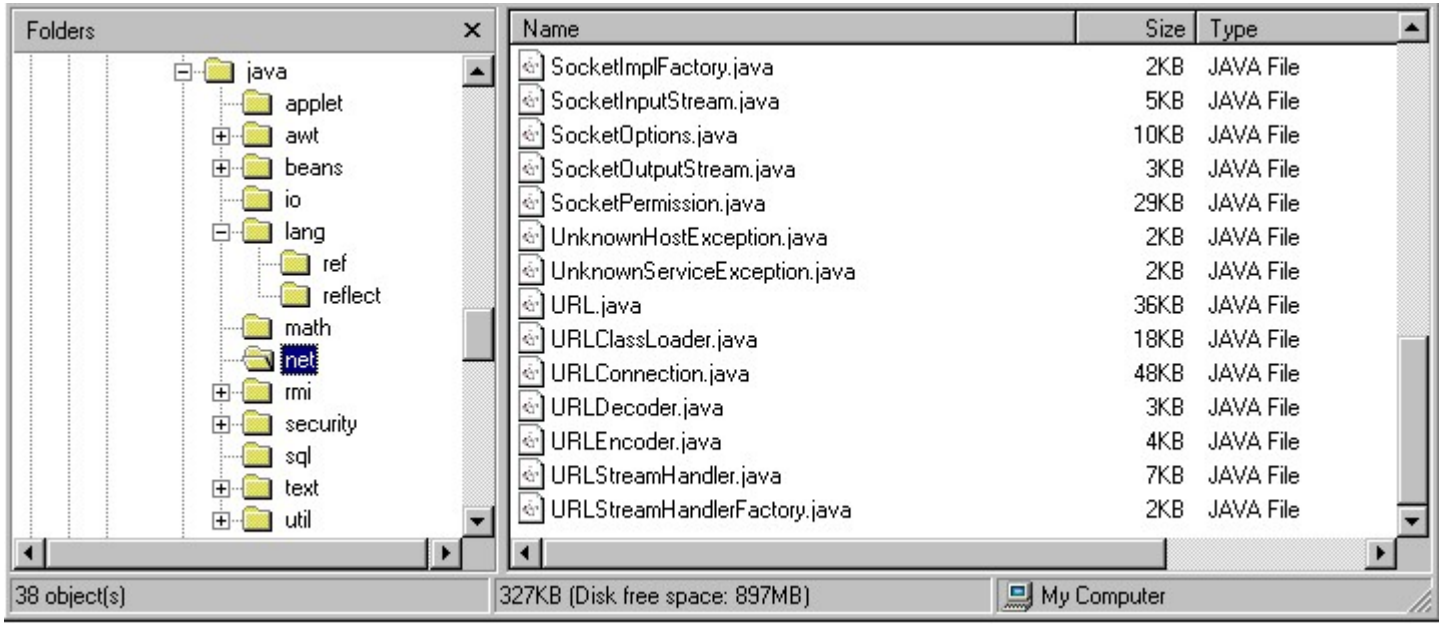
```
class Box{
        double dblWidth;
        double dblHeight;
        double dblDepth;
        double calcVolume(){
                return dblWidth * dblHeight * dblDepth;
        } //method calcVolume ends.
}//class Box ends.
```

```
class BoxDemo{
    public static void main(String a[])
    {
        Box box;                        //declare a reference to object
        box = new Box();            //allocate a memory for box object.
        box.calcVolume();           // call a method on that object.
} }
```


© 2017 Capgemini. All rights reserved.

4.2: Packages
Packages

In Java, by the use of packages, you can group a number of related classes and/or interfaces together into a single unit.



© 2017 Capgemini. All rights reserved.

4.2: Packages
Benefits of Packages

These are the benefits of organising classes into packages:

- It prevents name-space collision.
- It indicates that the classes and interfaces in the package are related.
- You know where to find the classes you want if they're in a specific package.
- It is convenient for organizing your work and separating your work from code libraries provided by others.

© 2017 Capgemini. All rights reserved.

4.2: Packages
Creating Your Own Package

```
package com.igate.trg.demo;
public class Balance {
    String name;
    public Balance(String n) {
        name = n;
    }
    public void show() {
      ......
       if( bal < 0)
         System.out.println(name + ": $" + bal);
}
```

Package should be the first statement

© 2017 Capgemini. All rights reserved.

4.2: Packages
Packages and Name Space Collision

Namespace collision can be avoided by accessing classes with the same name in multiple packages by their fully qualified name.

package pack1;

- class Teacher
- class Student

package pack2;

- class Student
- class Courses

```
import pack1.*;
import pack2.*;
pack1.Student stud1;
pack2.Student stud2;
Teacher teacher1;
Courses course1;
```


© 2017 Capgemini. All rights reserved.

4.2: Packages
Using Packages

Use fully qualified name.

```
java.util.Date = new java.util.Date();
```

You can use import to instruct Java where to look for things defined outside your program.

```
import java.util.Scanner;
Scanner sc = new Scanner (System.in);
```

You can use multiple import statements

You can use ∗ to import all classes in package:

```
import java.util.*;
Scanner sc = new Scanner (System.in);
```

Use * carefully; you may overwrite definitions

© 2017 Capgemini. All rights reserved.

4.2: Packages
Static Import

Static import enables programmers to import static members.

Class name and a dot (.) are not required to use an imported static member.

```
import static java.lang.Math.*;
public class StaticImportTest
{
 public static void main( String args[] )
 {
 System.out.printf( "sqrt( 900.0 ) = %.1f\n", sqrt( 900.0 ) );
 } // end main
}
```

Note: It's not Math.sqrt

© 2017 Capgemini. All rights reserved.

4.2: Packages
Some Java Packages

Package Name	Description
java.lang	Classes that apply to the language itself, which includes the Object class, the String class, and the System class. It also contains the Wrapper classes. "Classes belonging to java.lang package need not be explicitly imported".
java.util	Utility classes, such as Date, as well as collection classes, such as Vector and Hashtable
java.io	Input & output classes for writing to & reading from streams (such as standard input and output) & for handling files
java.net	Classes for networking support, including Socket and URL (a class to represent references to documents on the WWW)
java.applet	Classes to implement Java applets, including the Applet class itself, as well as the AudioClip interface

© 2017 Capgemini. All rights reserved.

4.2: Packages
Demo : Package

Execute the following programs:

- Balance.java
- AccountBalance.java
- StaticImportDemo.java
- StaticImportNotUsed.java

© 2017 Capgemini. All rights reserved.

4.3: Access Modifiers
Types of Access Modifiers

Default

Private

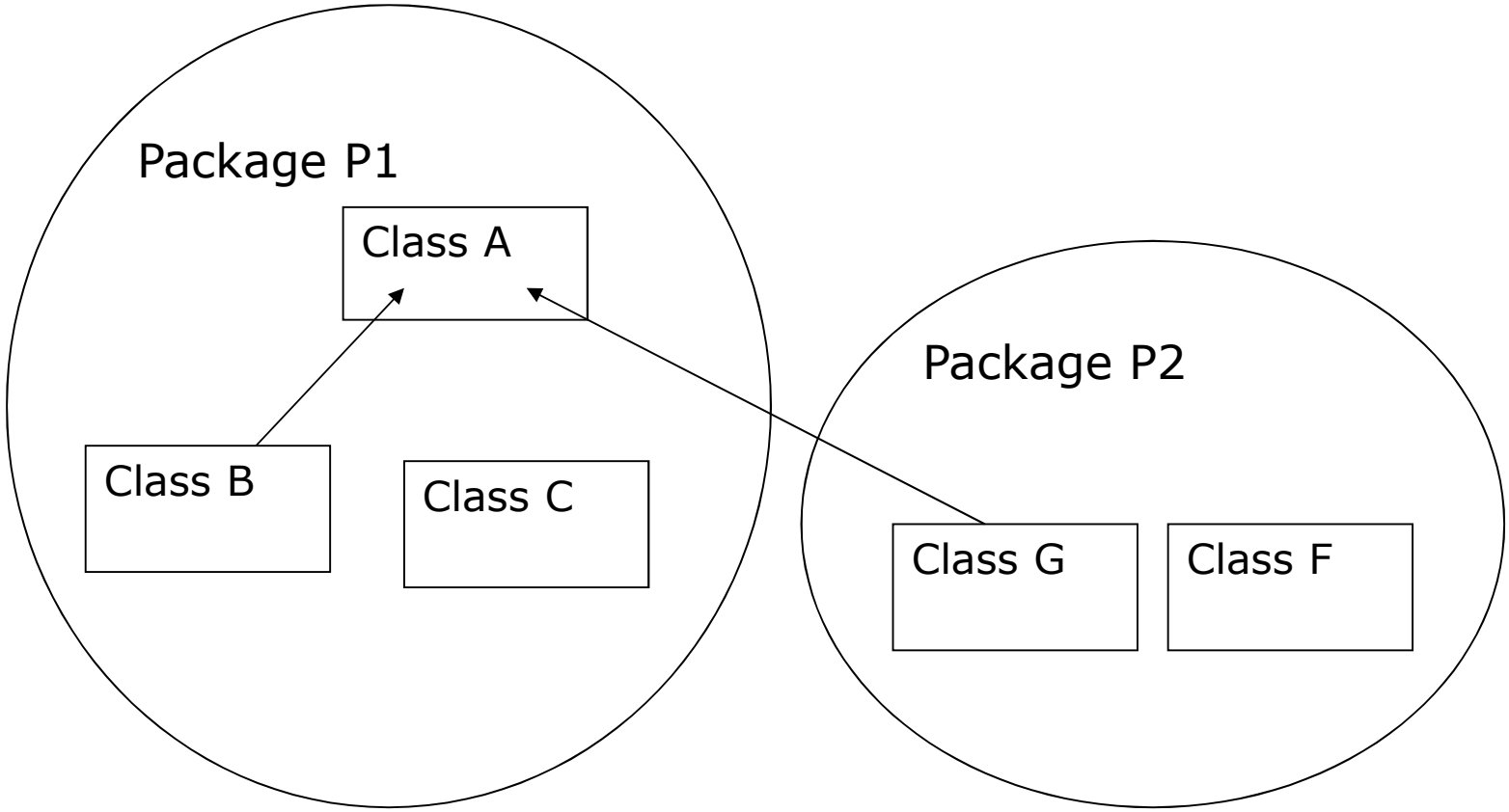
Public

Protected

Location/Access Modifier	Private	Default	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package non-subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

© 2017 Capgemini. All rights reserved.

4.3: Access Specifiers and Modifiers
What is access protection?



© 2017 Capgemini. All rights reserved.

4.4: Constructors
Default Constructors

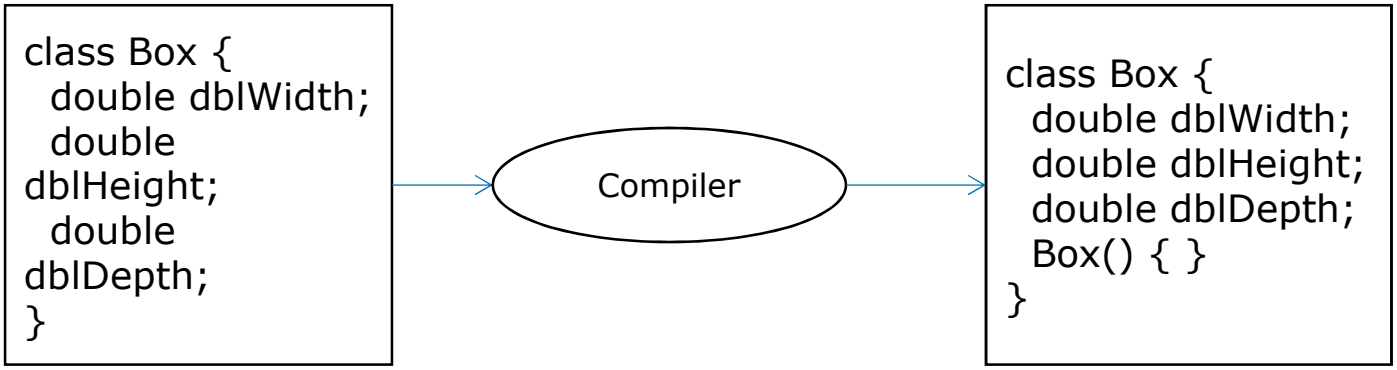
All Java classes have *constructors*

- Constructors initialize a new object of that type

Default no-argument constructor is provided if program has no constructors

Constructors:

- Same name as the class
- No return type, not even void



© 2017 Capgemini. All rights reserved.

4.4: Constructors
Demo

Execute the BoxDemo.java program.

- This uses the Box.java

© 2017 Capgemini. All rights reserved.

4.5: this reference
this reference

The this keyword is used to refer to the current object from any method or constructor.

There are mainly two uses of this keyword:

- Refer the class level fields
- Chaining constructors

```
// Field reference using this
class Point {
        int xCord;// instance variable
        int yCord;

        Point(int xCord, int yCord) {
            this.xCord = xCord;
            this.yCord = yCord;
        }
}
```


© 2017 Capgemini. All rights reserved.

4.6: Memory Management
Memory Management

Dynamic and Automatic

No *Delete* operator

Java Virtual Machine (JVM) de-allocates memory allocated to unreferenced objects during the garbage collection process

© 2017 Capgemini. All rights reserved.

4.6: Memory Management Enhancement in Garbage Collector

Garbage Collector:

- Lowest Priority Daemon Thread
- Runs in the background when JVM starts
- Collects all the unreferenced objects
- Frees the space occupied by these objects
- Call *System.gc()* method to "hint" the JVM to invoke the garbage collector
  - There is no guarantee that it would be invoked. It is implementation dependent

© 2017 Capgemini. All rights reserved.

4.6: Memory Management
Finalize() Method

Memory is automatically de-allocated in Java

Invoke *finalize()* to perform some housekeeping tasks before an object is garbage collected

Invoked just before the garbage collector runs:

- protected void finalize()

© 2017 Capgemini. All rights reserved.

4.7: using static keyword
Static modifier

Static modifier can be used in conjunction with:

- A variable
- A method

Static members can be accessed before an object of a class is created, by using the class name

Static variable :

- Is shared by all the class members
- Used independently of objects of that class
- Example: static int intMinBalance = 500;

© 2017 Capgemini. All rights reserved.

4.7: using static keyword
Static modifier

Static methods:

- Can only call other static methods
- Must only access other static data
- Cannot refer to this or super in any way
- Cannot access non-static variables and methods

Static constructor:

- used to initialize static variables

Method main() is a static method. It is called by JVM.

© 2017 Capgemini. All rights reserved.

4.7: using static keyword
Static modifier

```
// Demonstrate static variables, methods, and blocks.
public class UseStatic {
    static int intNum1 = 3;                    // static variable
    static int intNum2;
static {                                  //static constructor
        System.out.println("Static block initialized.");
        intNum2 = intNum1 * 4;
    }
    static void myMethod(int intNum3) {     // static method
        System.out.println("Number3 = " + intNum3);
        System.out.println("Number1 = " + intNum1);
        System.out.println("Number2 = " + int
    }
public static void main(String args[]) {
    myMethod(42);
  }}
```

Output:
Static block initialized.
Number3 = 42
Number1 = 3
Number2 = 12

© 2017 Capgemini. All rights reserved.

4.8: Enums
Enums

ENUM representation

pre-J2SE 5.0

```
public static final int SEASON_WINTER = 0;
public static final int SEASON_SUMMER = 1;
public static final int SEASON_SUMMER = 2;
```

Problem?

- Not type safe (any integer will pass)
- No namespace (SEASON_*)
- Brittleness (how do add value in-between?)
- Printed values uninformative (prints just int values)

Solution: New type of class declaration

- enum type has public, self-typed members for each enum constant

© 2017 Capgemini. All rights reserved.

4.8: Enums
Declaring Type Safe Enums

Permits variable to have only a few pre-defined values from a given list

Helps reduce bugs in the code

- Example:

```
enum CoffeeSize { BIG, HUGE, OVERWHELMING };
CoffeeSize cs = CoffeeSize.BIG;
```

cs can have values *BIG, HUGE* and *OVERWHELMING* only

© 2017 Capgemini. All rights reserved.

4.8: Enums

Enums with Constructors, Methods and Variables

Add constructors, instance variables, methods, and a constant specific class body

- Example:

```
enum CoffeeSize {
      BIG(8), HUGE(10), OVERWHELMING(16);
      // the arguments after the enum value are "passed"
     // as values to the constructor
    CoffeeSize(int ounces) {
              this.ounces = ounces;
             // assign the value to an instance variable
}
```


© 2017 Capgemini. All rights reserved.

4.8: Enums
Demo

Demo: EnumMonths.java

© 2017 Capgemini. All rights reserved.

4.9: Best Practices
Constructor

Initializing fields to default values is redundant

Constructors should not call *overridables*

Beware of mistaken field *redeclares*

```
public final class Quark {
//private String fName;
//private double fMass;
 public Quark(String aName, double aMass){
     fName = aName;
     fMass = aMass;
   }
//WITH redundant initialization to default values
private String fName = null;
private double fMass = 0;
}


>javap -c -classpath . Quark
```


© 2017 Capgemini. All rights reserved.

4.9: Best Practices
Static and Constants

Declare constants as static and final

Static, final and private methods are faster

If possible, use constants in *if* conditions

© 2017 Capgemini. All rights reserved.

Lab

Lab 2: Language Fundamentals , Classes and Objects

© 2017 Capgemini. All rights reserved.

Summary

In this lesson you have learnt:

- Classes and Objects
- Packages
- Access Specifiers
- Constructors - Default and Parameterized
- this reference
- Memory management
- Using static keyword
- Enums
- Best Practices

© 2017 Capgemini. All rights reserved.

Review Questions

Question 1: Which of the following are the benefits of using Package?

- **Option1:** prevents name-space collision.
- **Option2:** To implement security of contained classes.
- **Option3:** Better code library management.
- **Option4:** To increase performance of your class.

Question 2: Which of the following is true regarding enum?

- **Option1:** enum cannot be used inside methods.
- **Option2:** enum need not have a semicolon at the end.
- **Option3:** enum can be only declared with public or default access specifier.
- **Option4:** All the above are true.

© 2017 Capgemini. All rights reserved.