Core Java 8 and Development Tools

Lesson 04 : Classes and Objects

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

Lesson Objectives:
This lesson introduces to the fundamentals of the Java Programming Language.

Lesson 4: Classes and Objects
  4.1: Classes and Objects
  4.2: Packages
  4.3: Access Specifiers
  4.4: Constructors
  4.5: this reference
  4.6: Memory Management
  4.7: using static keyword
  4.8: Enums
  4.9: Best Practices

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
```

Classes describe objects that share characteristics, methods, relationships, and semantics. Each class has a name, attributes (its values determine state of an object), and operations (which provides the behavior for the object).

What is the relationship between objects and classes? What exists in real world is objects. When we classify these objects on the basis of commonality of structure and behavior, the result that we get are the classes.

Classes are "logical". They don't really exist in real world. When writing software programs, it is the classes that get defined first. These classes serve as a blueprint from which objects are created.

Reference: Refer to OOP material for a detailed discussion on Object-Oriented Programming Concept.
Note: Java does not have functions defined outside classes (as C++ does).

4.1 : Classes and Objects
Introduction to Classes

- A class may consist the following elements:
  - Fields
  - Methods
  - Constructors
  - Initializers

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved   4

   A Java class may consist of the components as listed in the above slide. Fields and methods are primary components of any class and are termed as members of class. A class can have zero or more class members.

A class methods are categorized as business method or getters/setters. A business method of a class usually contains the business logic for the given problem/requirement.

A getter/setter usually written to access the private properties/attributes of a class. For example, consider the property given below:

```
private String name;
//getter
public String getName() { return name;}
//setter
public void setName(String name) {this.name=name;}
```
For the boolean properties, the getter format isXXX()

Constructors are special methods (having same name as the class name and with no value return) which are used to create object of class. A class must have at least one constructor.
   Initializers are special blocks in class, used to initialize members of class. A class can have zero or more initializers.

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.
}}
```

The above slide shows an example of Box class with 3 fields and 1 method.

The new operator followed by the call to constructor of the class is used to create object. The above slide shows how to create an object of Box class.

        new <<call to constructor>>;

Even though the Box class doesn't add constructor, Java compiler internally does. This constructor is called as default constructor and added by Java compiler only when class don't have any constructor declared.
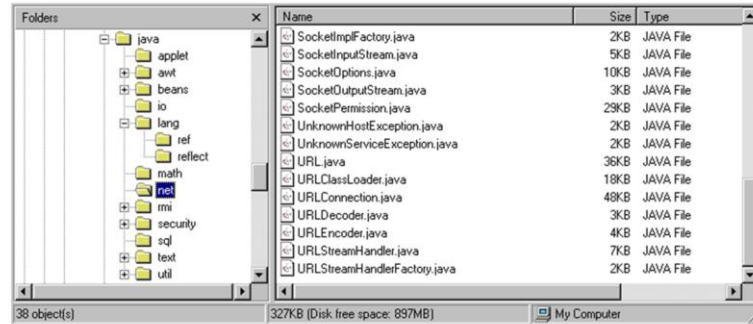
        Box() {   }

Once object created, we can access class members using dot operator. The contents of main() method shows how objects are instantiated and how methods are invoked.

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.

Folders	×	Name	Size	Type
⊟ 📁 java		SocketImplFactory.java	2KB	JAVA File
📁 applet		SocketInputStream.java	5KB	JAVA File
⊞ 📁 awt		SocketOptions.java	10KB	JAVA File
⊞ 📁 beans		SocketOutputStream.java	3KB	JAVA File
📁 io		SocketPermission.java	29KB	JAVA File
⊟ 📁 lang		UnknownHostException.java	2KB	JAVA File
📁 ref		UnknownServiceException.java	2KB	JAVA File
📁 reflect		URL.java	36KB	JAVA File
📁 math		URLClassLoader.java	18KB	JAVA File
📁 net		URLConnection.java	48KB	JAVA File
⊞ 📁 rmi		URLDecoder.java	3KB	JAVA File
⊞ 📁 security		URLEncoder.java	4KB	JAVA File
📁 sql		URLStreamHandler.java	7KB	JAVA File
⊞ 📁 text		URLStreamHandlerFactory.java	2KB	JAVA File
⊞ 📁 util				

38 object(s)   327KB (Disk free space: 897MB)   💻 My Computer

Packages:

When you work on some small projects you intend to put all java files into one single directory. It is quick, easy and harmless. However, if your project gets bigger, and the number of files increase, putting all these files into the same directory would be tedious for you. In java, you can avoid this sort of problem by using packages.

Basically, files in one directory (or package) have different functionality from those of another directory. For example, files in java.io package carry out functions related to I/O, but files in java.net package provide you the way to deal with the Network.

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.

Benefits of Packages:
Prevents name-space collision: One of the many concerns that programmers face today is, how to ensure that their source code does not conflict with the source code of other programmers. A typical example is the case in which two programmers define two distinct classes that have the same name. Suppose, you decide to write a List class that keeps a sorted list of objects. This would inherently conflict with the List class in the Java API that is used for displaying a list of items in a Graphical User Interface. To this problem, Java has a simple solution, which is known as namespace management. In namespace management, each programmer defines their own namespace and places their code within that namespace, thus two classes that have the exactly same name are now distinguishable since they occur in different namespaces. Namespaces are called packages in Java.

Provides greater control over source codes: Another important reason for using packages is that it provides programmers with greater control over their source code. It is typical to have a few thousand source files in medium to large scale applications. Trying to maintain them would be difficult, if not impossible. However, separating these source files into packages makes it much easier to manage the source code. Usually, related classes are grouped into a single package, for example, all the user interface classes of an application are grouped into a package.

Makes it easy to find a class: If you are looking for a specific class and you know about the functionality it provides, you will naturally be able to find it in the right package. For example, if you are looking for an InputStreamReader, you will find it in the input and output package, that is, java.io.

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);
    }
}
```

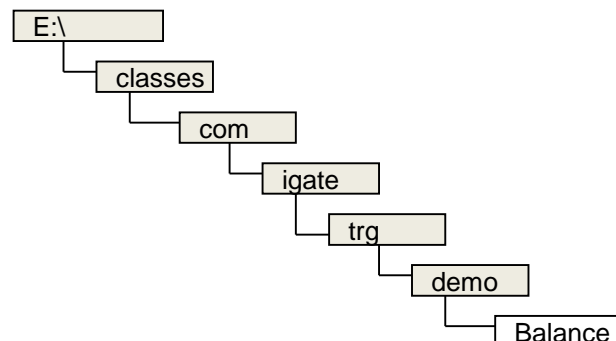Package should be the first statement

Creating Your Own Package:
A package is a directory structure mapped on the operating system, hence compilation of Balance.java file results in the creation of directory structure com\igate\trg\demo and under that directory, Balance.class file is created. A package can contain more than one class and interface.

Compiling into a Package:
While compiling the file, we need to specify the URL where directory, com is to be created. For example, if you give the following command at command prompt.
    E:\yourdirectory> javac   –d   E:\classes   Balance.java
It is expected that you have a directory called classes under E:\. the result of this command is this directory structure. Here, all the grayed boxes are directories.

```
E:\
  └── classes
        └── com
              └── igate
                    └── trg
                          └── demo
                                └── Balance
```

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;
```

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

java.lang package is automatically imported by every Java program.

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[] )          Note: It's not
  {                                                  Math.sqrt
  System.out.printf( "sqrt( 900.0 ) = %.1f\n", sqrt( 900.0 ) );
  } // end main
}
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING   Copyright © Capgemini 2015. All Rights Reserved   11

Static Import:

A static import declaration has two forms:
1. The form that imports a particular static member (which is known as single static import):
The following syntax imports only the sqrt method of the Math class:

```
import static java.lang.Math.sqrt;
```

2. The form that imports all static members of a class:
The following syntax imports all static members of the Math class:

```
import static java.lang.Math.*
```

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

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Refer to the java documentation for more details on other important packages.

4.2: Packages
Demo : Package

- Execute the following programs:
  - Balance.java
  - AccountBalance.java
  - StaticImportDemo.java
  - StaticImportNotUsed.java

Demo

Capgemini

```java
package com.igate.lesson4.demo;
public class Balance {
    String name;
    double bal;
    public Balance(String n, double b) {
        name = n;
        bal = b;
    }
    public void show() {
      if(bal<0)
          System.out.println(name + ": $" + bal);
    }}
```

```java
package com.igate.lesson4;
import com.igate.lesson4.demo.Balance;
class AccountBalance {
  public static void main(String args[]) {
     Balance object= new Balance("K. J. Fielding", 123.23);
    object.show();
  }
}
```

4.3: Access Specifier
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

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

public access modifier
Fields, methods and constructors declared public (least restrictive) within a public class
are visible to any class in the Java program, whether these classes are in the same
package or in another package.

private access modifier
Fields, methods or constructors declared private (most restrictive) cannot be accessed
outside an enclosing class. This modifier cannot be used for classes. It also cannot be
used for fields and methods within an interface. A standard design strategy is to make all
fields private and provide public getter methods for them.

protected access modifier
Fields, methods and constructors declared protected in a superclass can be accessed
only by subclasses in other packages. Classes in the same package can also access
protected fields, methods and constructors, even if they are not a subclass of the
protected member's class. This modifier cannot be used for classes. It also cannot be
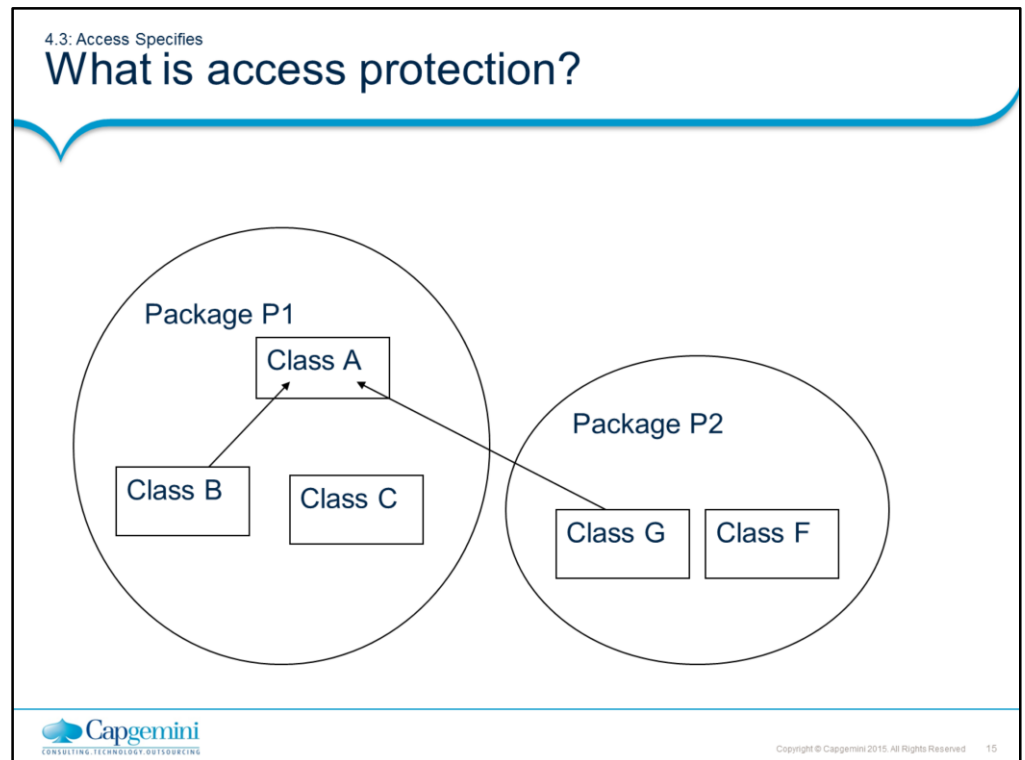used for fields and methods within an interface.

default access modifier
Default specifier is used when "no access modifier is present". Any class, field, method or
constructor that has no declared access modifier is accessible only by classes in the
same package. The default modifier is not used for fields and methods within an
interface.

The table shown above is applicable only to members of classes. A class has only two
possible access levels: default and public.
When a class is declared as public, it is accessible by any other code.
If a class has default access, then it can only be accessed by other code within its same
package

4.3: Access Specifies
What is access protection?

Package P1

Class A

Class B   Class C

Package P2

Class G   Class F

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING   Copyright © Capgemini 2015. All Rights Reserved   15

Access Protection:

Classes and packages are both means of encapsulating and containing the name
space and scope of variables and methods. Packages act as containers for
classes. Classes act as containers for data and code. Accessibility in java is
specified with respect to packages. Because of the interplay between classes and
packages, Java addresses five categories of visibility for class members.

Same Class
Subclass in the same package
Non-subclasses in the same package
Subclasses in different package
Classes that are neither in the same package nor in subclasses.
See the diagram given in the slide, to understand the five categories.
There are two packages P1 and P2. Package P1 contains Class A, B and C. B
inherits from A. Package P2 contains Class G and F. G inherits from A. Which
class comes under which category with respect to class A, is described below:

Same Class (Class A)
Subclass in the same package (Class B)
Non-subclasses in the same package (Class C)
Subclasses in different package (Class G)
Classes that are in neither the same package nor subclasses. (Class F)

4.4: Constructors - Default and Parameterized
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

```
class Box {
  double dblWidth;
  double
dblHeight;
  double dblDepth;
}
```
→ Compiler →
```
class Box {
  double dblWidth;
  double dblHeight;
  double dblDepth;
  Box() { }
}
```

Class basically holds blueprint of Objects. In order to create and instantiate objects of class and also to provide initial values for the object, constructors are used. The structure of a constructor looks similar to a method. Constructors are used to create objects of a class. A Java class must have at least one constructor.

A class can have many constructors in order to facilitate the object creation in different ways.

Default Constructor: If class doesn't declare any constructor, compiler adds a constructor to the class. This constructor is called as default constructor. The default constructor accepts no arguments. Above slide shows an example of default constructor.

4.4: Constructors- Default and Parameterized

Demo

- Execute the BoxDemo.java program.
  - This uses the Box.java

This demo example contains default (no-arg) constructor as well as a constructor that takes 3 parameters. Three box objects are created with initialization done using constructors and setter methods.

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;
        }
}
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved   18

this keyword:

this keyword is used to refer the current object. As shown in the above example, the constructor parameters are shadowing the instance variables. Therefore we can use this keyword to make difference between the local variables/parameters and instance variables.

The other use of this keyword is to invoke constructor of same class.

```
class Point {
        int xCord;
        int yCord;
        Point() {
            this(0, 0);        //chaining constructors using this
        }
        Point(int xCord, int yCord) {
            this.xCord = xCord;
            this.yCord = yCord;
        }
}
```

Note: this keyword cannot be used to refer static variables.

4.6: 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;

Variables and methods marked with static modifier belong to the class rather than any particular instance. Ie, you do not have to instantiate the class to invoke a static method or access a static variable.

4.6: 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.
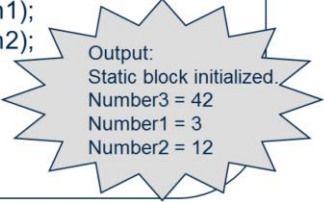
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

main() is called by JVM. Making this method static means the JVM does not have to create an instance of your class to start running code.

Static constructor is also known as static initialization block. This is a normal block of code enclosed in braces { } and preceded by the static keyword. This can appear anywhere in the class body. It is normally used to initialize static variables.

4.6: 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 = " + intNum2);
    }
public static void main(String args[]) {
    myMethod(42);
    }}
```

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

4.7: 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
```

Initializing fields to default values is redundant.
In the declaration of a field, setting it explicitly to its default initial value is always redundant, and may even cause the same operation to be performed twice (depending on your compiler). Declaring and initializing object fields as null, for instance, is simply not necessary in Java.

(It is worth recalling here that Java defines default initial values only for fields, and not for local variables).

Constructors should not call overridables.
That is, they should only call methods that are private, static, or final.

Beware of mistaken field re-declares.
Beware of this simple mistake, which can be hard to track down : if a field is mistakenly redeclared within the body of a method, then the field is not being referenced, but rather a temporary local variable with the same name. A common symptom of this problem is that a stack trace indicates that a field is null, but a cursory examination of the code makes it seem as if the field has been correctly initialized.

4.7: 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

Declare constants as static and final.
Use static, if variable is not going to change. This reduces the memory space for the object. A single copy is shared among all the objects.
Static, final and private methods are faster.
Static, final and private functions are treated as inline functions. These types of methods are resolved faster than other types because resolutions are done only within the class. So, if the method is not to be overridden by the sub class then define it as final, static and private.
If possible, use constants in if conditions.
Conditional Compilation may be useful. Conditional compilation gets rid of the run time resolution of variables and their values.

```
class A {
    public static final constA = 1;
}
class B {
  public static void main(String arg[])  {
        if (A.constA == 1)  System.out.println("inside if ");
  }
}
```

In the above example, during the compilation of Class B the if condition is checked. Since the condition involves the final (constant), the time required for resolving the if condition and variables are not done at runtime, which can save some time. But if the value of const "constA" is changed ensure that both classes are compiled.

Lab

- Lab 1: Language Fundamentals , Classes and Objects

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

Summary

Add the notes here.

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 static variable?
  - **Option1:** static variable cannot be used inside instance methods.
  - **Option2:** static variable can be used in static methods only.
  - **Option3:** static variable can be used in static methods as well as instance methods.
  - **Option4:**. static variable cann't be used in constructor .

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING