Chapter 3- Class & Objects

By Biruk M.

Outline

All that is to know on class & objects

- Introduction
- Class variables
- Access specifiers
- This vs this()
- Mutable & immutable classes
- packages
- Exercise

3.1. Introduction

Classes

- A set of similar objects is called a class.
- Model objects that have attributes (data members) and behaviors (member functions)
- Defined using keyword **class**
- Have a body delineated with braces (**{** and **}**)
  - Example: public Student{….}
- Each **.java** source file may contain only one public class. A source file may contain any number of default visible classes.
- Finally, the source file name must match the public class name and it must have a .java suffix.

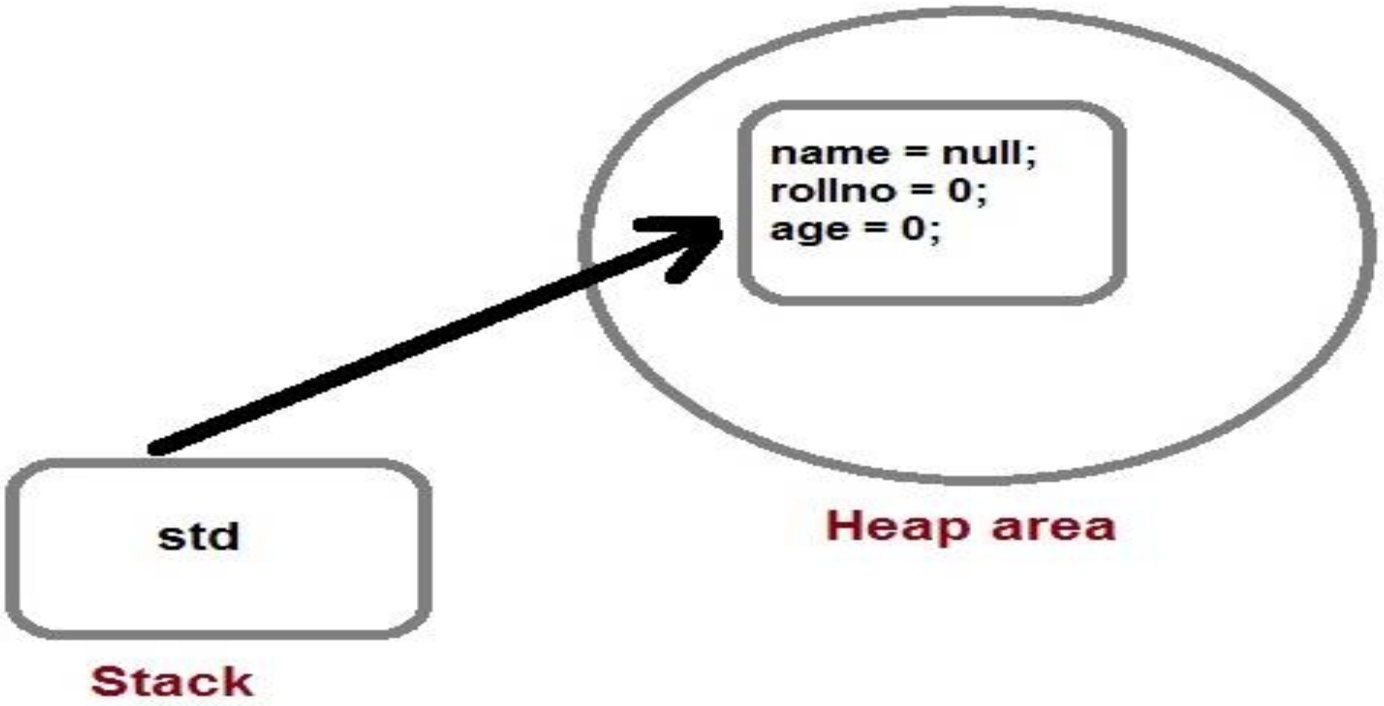
Object

- An Object represents a real world entity. It is complete and self contained with distinct characteristics and behaviors.
- Is a unique instance of one class

Cont…

Eg:

- class Student. { String name; int rollno; int age; }
- Student std=new Student();
  - **The new operator dynamically allocates memory for an object**



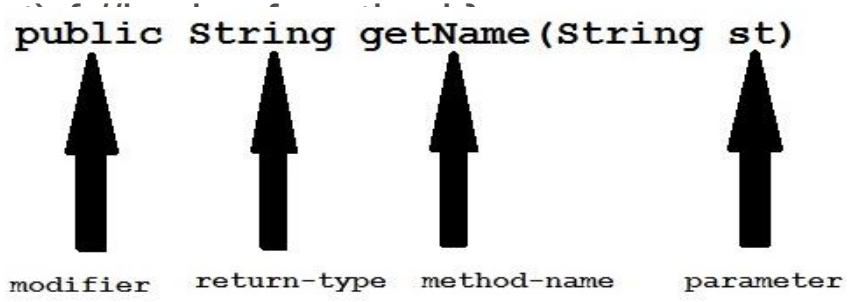
Cont…

Methods

- Method describe behavior of an object.
- A method is a collection of statements that are group together to perform an operation.
- **Syntax :**
  - return-type methodName(parameter-l
- **Example**
  - public String getName(String st)

  { String name="StudyTonight";

  name=name+st; return name; }



- Member Methods
  - Methods that are defined in the class and invoked using the class object
  - Eg: public int getAge():-is a member function
- **Static methods:**
  - Methods that can be defined by using keyword **static** and can be called

Cont…

Constructor

- A class contains constructors that are invoked to create objects from the class blueprint.
- Constructor declarations look like method declarations—except that they use the name of the class and have no return type.
- For example, Student constructor:
- private String name; private int age;
- public Student(String n, int a) {
- name=n;
- age=a;
- }
- To create a new Student object called s1, a constructor is called by the new operator:
- Student s1=new Student (“Abebe”,20);//creates space in memory for the object

Cont…

- We can also create a no-argument constructor/default-const:
  - public Student() {name=”abebe”; age=12;}
- **Difference b/n Constructors & Methods???**
  - **constructors** can not return a value and are only called once (return current instant of a class)
  - while regular methods could be called many times and it can return a value or can be void.
- Student s1=new Student (); invokes the no-argument constructor to create a new Student object (s1)
- As with methods, the Java platform differentiates constructors on the basis of **the number of arguments in the list and their types**.(constructor overloading)
- You cannot write two constructors that have the same number and type of arguments for the same class, because the platform would not be able to tell them apart. Doing so causes a compile-time error.
- By default, the compiler automatically provides a no-argument,default constructor for any class without constructors.

3.2 Class Variables

The variable of any class are classified into two types;

- Static or class variable
- Non-static or instance variable

Static variable

- Memory for static variable is created only one in the program at the time of loading of class.
- These variables are preceded by **static keyword**. static variable can access **with class reference**.
- static variable not only can be access with class reference but also some time it can be accessed with object reference.

Non-static variable

- Memory for non-static variable is created at the time of create an object of class. These variable should not be preceded by any static keyword Example: **These variables can access with object**

Cont…

Non-static variable

No static keyword-EG:

```
class  A
{
int a;
}
```

They are specific to an object

They can access with object reference.

Syntax

- obj_ref.variable_name

Static-variable

preceded by static keyword.

```
class  A
{
static int b;
}
```

They are common for every object /there memory location can be **sharable** by every object reference or same class.

Static variable can access with class reference.

- **Syntax:**class_name.variable_nam

cnt'd…

Non-static variable

Memory is allocated for these variable whenever an object is created

Memory is allocated multiple time whenever a new object is created.

Non-static var/instance var because memory is allocated whenever instance is created.

Static-variable

Memory is allocated for these variable at the time of loading of the class.

Memory is allocated for these variable only once in the program.

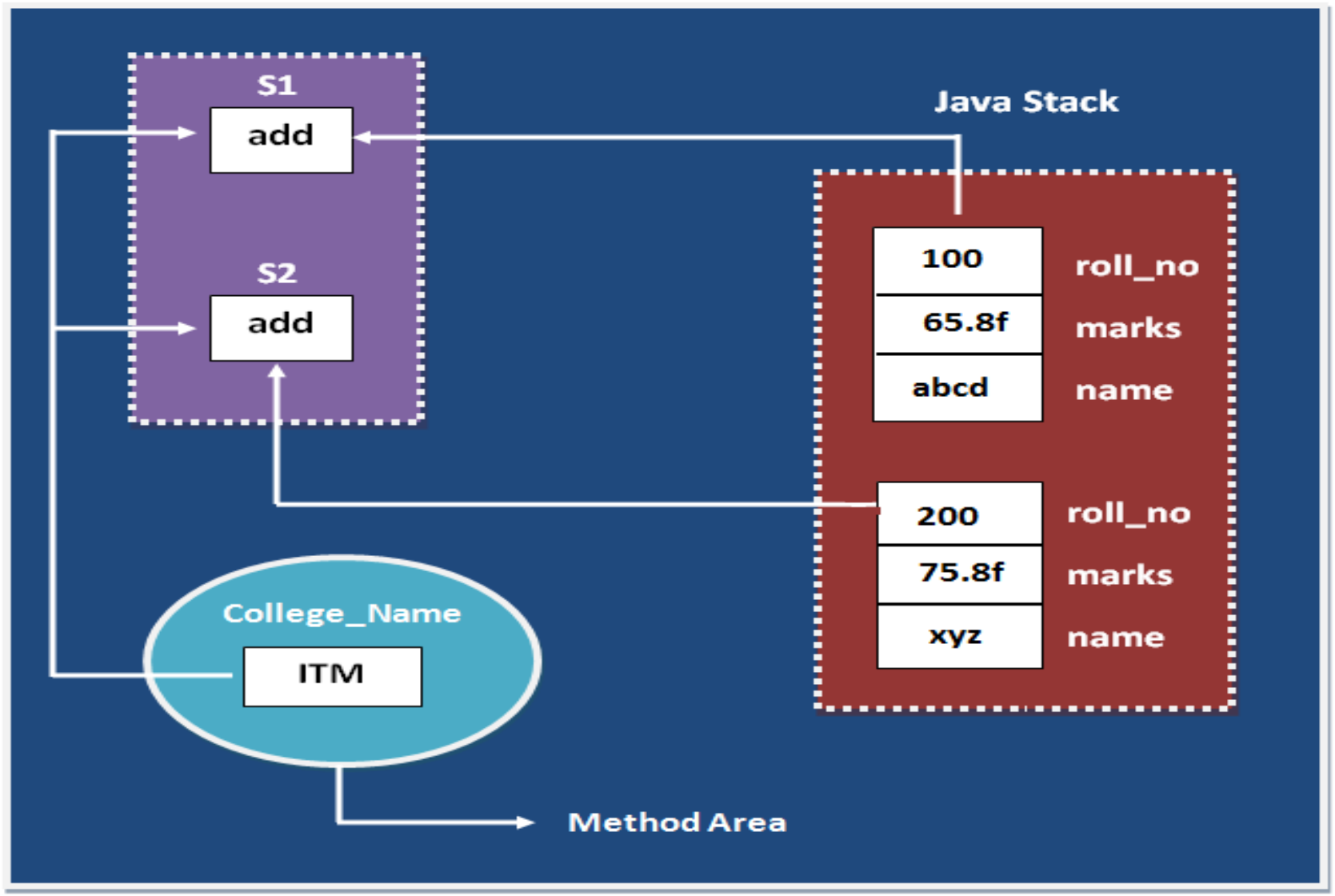
Static var/class-var Because Memory is allocated at the time of loading of class

Example-1

```
class Student
{  int roll_no;
float marks;
String name;
static String College_Name="ITM"; }
class StaticDemo {
public static void main(String args[])
{
Student s1=new Student();
s1.roll_no=100;
s1.marks=65.8f;
s1.name="abcd";
System.out.println(s1.roll_no);
System.out.println(s1.marks);
Student  s2=new  Student();
s2.roll_no=200;
s2.marks=75.8f;
s2.name="zyx";
System.out.println(s2.roll_no);
System.out.println(s2.marks);
System.out.println(s2.name);
System.out.println(Student.College_Name);
}
}
```

What will be the output?

Pictorially:



Example-2 (predict output for both???)

```
class Counter
{
int count=0;// what if it is static int count=0;
Counter()
{
count++;
System.out.println(count);
}
public static void main(String args[])
{
Counter c1=new Counter();
Counter c2=new Counter();
Counter c3=new Counter();
```

3.3 Access Specifiers

The four access levels are −

- 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).**

Note:

- a class cannot be associated with the access modifier **private**, but if we have the class as **a member of other class**, then the inner class can be made private.
- outer classes can only be declared public or *package private*.
- Use the most restrictive access level that makes sense for a particular member. (private)unless you have a good reason not to.
- Avoid public fields except for constants.It is not recommended for production code. (Public fields tend to link you to a particular

Cont…

1. Default Access Modifier - No Keyword

- 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.**

Example:

- Variables and methods can be declared without any modifiers, –

```
String version = "1.5.1";

 boolean processOrder() {   return
   true;}
```

2. Private Access Modifier

- Methods, variables, and constructors that are declared private can only be accessed within the declared class itself.
- the most restrictive access level. Class and interfaces cannot be private.
- Using the private modifier is the main way that an object **encapsulates** itself and hides data from the outside world.
- **Example:**

```
public class Eg1 {

 private String name;

public String getName() {      return
   this.name;   }
```

Cont…

3. Public Access Modifier

- 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.
  - **Example**

```
public static void main(String[]
```

4. Protected Access Modifier

- 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, however methods and fields in an interface cannot be declared protected.

Cont…

- Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.

Example

- **its child class override *openSpeaker()* method −**

```
class AudioPlayer {

  protected boolean openSpeaker(Speaker sp) {      // implementation
    details   }}

class StreamingAudioPlayer

{//sub-class

boolean openSpeaker(Speaker sp)

 {      // implementation details   }}
```

- **What if it (**openSpeaker**) is declared as private? Or public?**

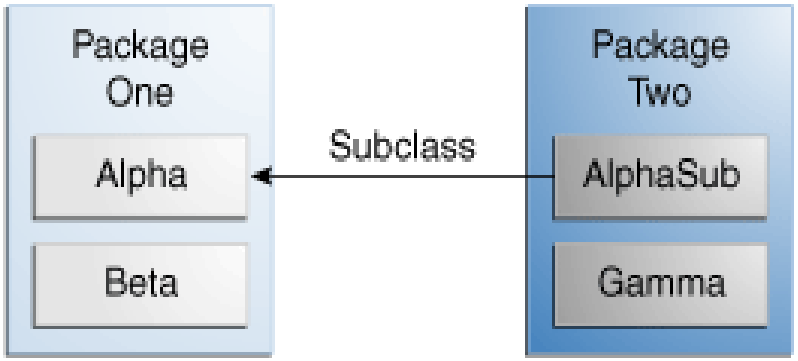
Summary

how access levels affect visibility in class relationships

The following table shows the access to members permitted by each modifier E

Access Levels

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N



Visibility

Modifier	Alpha	Beta	Alphasub	Gamma
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Example-1(class & Objects)

```
public class simpleClass {
    private String name;
    private int age;
    public simpleClass(String n,int a)
    {  this.name=n; //what if there is setter methods?
        this.age=a;  }
    public String getName()
    { return name; }
    public int getAge()
    { return age; }
    public static void main(String args[])
    {        simpleClass s1=new simpleClass("abe",12);
        String n=s1.getName();
        int a=s1.getAge();
        System.out.println(n+"\t"+a); }
```

Example-2(static-methods)

```
import java.util.Scanner;
public class c1 {
    public static void main(String[] args)
      {    method1();
           c1.method2() ;  }
     public static void method1()
     {  System.out.println("hello");}
     public  static void method2()
     {    int x,y,r;
          Scanner s=new Scanner (System.in);
          System.out.println ("enter...");
          x=s.nextInt();  y=s.nextInt();
          r=x+y;
               System.out.println("res="+r);}
     }
```

Exercises:

- Define a **class product** with members(name, cost, qty & calcCost(),calcTax(),calcProf())
  - Input values
    - Using constructor
    - Using input from keyboard(Scanner/JOptionPane)
    - What if for N-products
- Define **four static methods** of simple calculator(Arithemetics)
  - Input values
    - Using constructor
    - Using input from keyboard(Scanner/JOptionPane)

3.4 This and this()

this keyword

- is used to refer always current object/current instance of a class,
- this is a non static and can not be used in static context, which means you can not use this keyword inside main method in Java(compilation error)

this() method

- is used to access one constructor from another where both constructors belong to the same class
  - With the both, **inheritance** is not involved; everything happens within the same class.
- OR it can be used in constructor chaining to call another constructor
  - e.g. this() calls **no argument constructor** of child and parent class.

Note:

- Another use of this in Java is for *accessing instance variables of a class and it's parent (if it is the same instance-to **remove ambiguity**)*
- *You can not reassign the (this variable) because you can not assign a new value to final variable this"*.-they are special variables and they are final

Examples-predict?

Eg-1

```
public class Number
{   int num;
  public void favorite(int num)
  {  this.num = num;
  }
  public static void main(String args[])
  {
    Number n1 = new Number();
    n1.favorite(8);
    System.out.println("Your favorite
    number is " + n1.num);
  }
}
```

Eg-2

```
public class Officer
{  public Officer()  {
     this("Second");
    System.out.println("I am First");
  }
  public Officer(String name)  {
    System.out.println("Officer name is " +
name);   }
public Officer(int salary) {
                    this();
 System.out.println("Officer salary is Rs."
+ salary);  }
public static void main(String args[])
  {    Officer o1 = new Officer(9000);
                }
```

Cont…

The this is also used to call Method of that class.

- public void getName()
- {
- System.out.println("Studytonight"); }
- public void display()
- {
- this.getName();
- System.out.println(); }

this is used to return current Object

- public Car getCar()
- { return this; }

3.5 Mutable & Immutable Class

Mutable classes

- We can change the state of object after initiation in case of mutable classes.
- These classes are not thread safe so we should use proper synchronize block to access its member data(instance variable).
- Its recommended to create getter and setter methods in muttable class to embrace Encapsulation.
- With mutable objects you can change the state anytime during its lifetime (via the setters),
- whereas with immutable objects you change only set its state when you create it (via the constructor).

Cont…

Immutable Classes

- are those classes whose state can not be changed after initiation(object creation).
- Immutable classes have only one state for whole life; So we can use these classes in multi-threading code without any data inconstancy.
  - because the effect of state change is often dependent on the order of a sequence of events, which you can't guarantee during multithreaded operation.
- Immutable classes do not required setter methods :
- Immutable classes are final no one can extend them(if you need to extend one class not make/call this immutable)
  - Java itself provide many immutable classes - String, Integer, Boolean, Float, other Wrapper classes.

For the JVM, there is no notion of "mutable" or "immutable" classes, and mutable objects are not treated differently from immutable objects.

Examples:

```
class Mutable{
private int value;
 public Mutable(int value) {
      this.value = value;
}
 getter and setter for value
}
class Immutable {
  private final int value;
 public Immutable(int value) {
   this.value = value;  }
```

Eg-2

- class Person {

  public int age;}.

  The age variable can be changed, hence mutability achieved. whereas

- class Person {

  public final int fingerNo = 10;}.

  Immutable objects can't be changed:

3.6 Packages

- **Packages** in Java are a way of grouping similar types of classes / interfaces together.(acts as a **container** for group of related classes).
- It is a great way to achieve **reusability.**
- We can simply **import** a class providing the required functionality from an existing package and use it in our program.
- it avoids name conflicts and controls access of class, interface and enumeration etc
- It is easier to locate the related classes
- The concept of package can be considered as means to achieve **data encapsulation**.
- consists of a lot of classes but only few needs to be exposed as most of them are required internally. Thus, we can **hide** the classes and prevent programs or other packages from accessing classes which are meant for internal usage only

Cont…

- The Packages are categorized as :
  - **Built-in packages** (standard packages which come as a part of Java Runtime Environment)
    - These packages consists of a large number of classes which are a part of Java **API**
  - **Accessing classes in a package: Eg:-**

    1) import java.util.Random; *// import the Random class from util package*

    2) import java.util.*; *// import all the class from util package*
  - **User-defined packages** (packages defined by programmers to bundle group of related classes)
    - Java is a friendly language and permits to create our own packages and use in programming.
    - We know packages avoid **name collision** problems.
    - Creating packages are indispensable in project development where number of developers are involved doing different modules and tasks.

Built-in packages

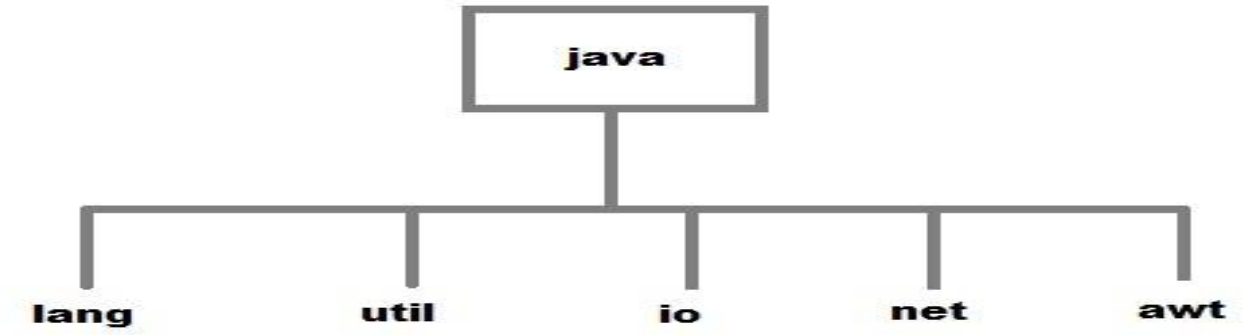
Examples:

Package Name	Description
java.lang	Contains language support classes (for e.g classes which defines primitive data types, math operations, etc.) . This package is automatically imported.
java.io	Contains classes for supporting input / output operations.
java.util	Contains utility classes which implement data structures like Linked List, Hash Table, Dictionary, etc and support for Date / Time operations.
java.applet	Contains classes for creating Applets.
java.awt	Contains classes for implementing the components of graphical user interface (like buttons, menus, etc.).
java.net	Contains classes for supporting networking operations.

Cont..

Subpackage

- A package created inside another package is known as a **subpackage**.
- When we import a package, subpackages are not imported by default. They have to be imported explicitly.
- Eg:
  - **import** java.util.*;(**util** is a subpackage inside **java** package)
  - **import javax.swing.Jbutton;?**
  - **import java.io.*;?**



Import and Static Import

Import :

- It facilitates accessing any static member of an imported class using the class name.

```
import java.lang.Math;
public class Eg1 {
  public static void main(String args[]) {
    double val = 64.0;
    double sqroot = Math.sqrt(val);
    System.out.println("Sq. root of " + val + " is " + sqroot);
  }
}
```

Cont…

Static Import

- It facilitates accessing any static member of an imported class directly i.e without using the class name.

```
import static java.lang.Math.*;
public class Eg2 {
   public static void main(String args[]) {
      double val = 64.0;
      double sqroot = sqrt(val);
      System.out.println("Sq. root of " + val + " is " + sqroot
);
   }
}
```

User defined packages

Creating a package in java is quite easy.

Simply include a package command followed by name of the package as the first statement in java source file.

- package RelationEg;

However, because of new editors we can simply create using GUI wizard

- R->click on ur project
- New java-package
- Then you can create many-related classes in it

example

```
package REL1;
public class Comp1 {
    public int getMax(int x, int y) {
        if ( x > y ) {
            return x;
        }
        else {
            return y;
        }
    }
}
```

Example cont…

```
package packageeg;
import REL1.Comp1;
public class EgComp {
   public static void main(String args[]) {
      int val1 = 7, val2 = 9;
      Comp1 comp = new Comp1();
      int max = comp.getMax(val1, val2); // get the max value
      System.out.println("Maximum value is " + max);
   }
}
```

Exercise

1.Create user defined package calculator and define four methods in one class (simple calc)

- Define a new class in different package and call the methods to do the four operations
  - Show simple import and static import

2. define **your own** packages and show userdefined package import?

3. define a class Student {name, age, ht,wt,st-type}

- Methods –insert(), display(),filter(), Undergraduate(),calcBMI()

Questions?

That concludes this chapter

Thank You

Biruk M.