Marwadi
education foundation

Unit – 4
Objects and Classes

Prepared By

Prof. Ravikumar Natarajan

Assistant Professor, CE Dept.

Defining classes for objects

- A class defines the properties and behaviors for objects.
- An object represents an entity in the real world that can be distinctly identified.
- For example, a student, a desk, a circle, a button, and even a loan can all be viewed as objects.
- An object has a unique identity, state, and behavior.

Properties for Objects

- The state of an object (also known as its properties or attributes) is represented by data fields with their current values.
- A circle object, for example, has a data field ‘radius’, which is the property that characterizes a circle.
- A rectangle object has the data fields ‘width’ and ‘height’, which are the properties that characterize a rectangle.

Characteristics of Objects

- An object has three characteristics:
- State: represents the data (value) of an object.
- Behavior: represents the behavior (functionality) of an object such as deposit, withdraw, etc.
- Identity: An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

Behaviors for Objects

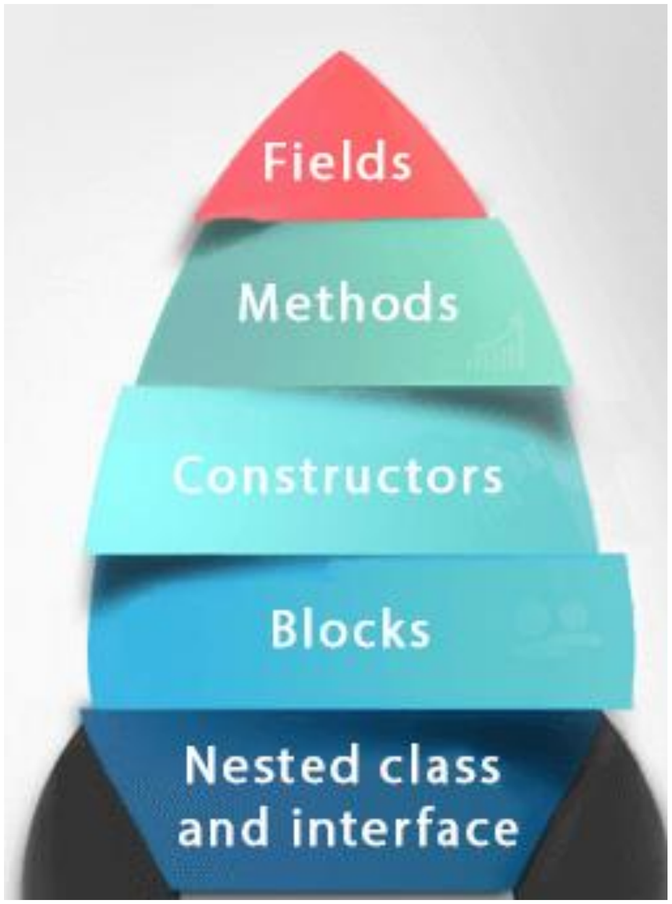
- The behavior of an object (also known as its actions) is defined by methods.
- To invoke a method on an object is to ask the object to perform an action.
- For example, you may define methods named getArea() and
- getPerimeter() for circle objects.
- A circle object may invoke getArea() to return its area and
- getPerimeter() to return its perimeter.
- You may also define the setRadius(radius) method.
- A circle object can invoke this method to change its radius.

Defining Classes for Objects

Marwadi education foundation

- Objects of the same type are defined using a common class.
- A class is a template, blue-print, or contract that defines what an object's data fields and methods will be.
- An object is an instance of a class.
- You can create many instances of a class.
- Creating an instance is referred to as instantiation.
- The terms object and instance are often interchangeable.

Defining Classes for Objects



Defining Classes for Objects

```
class Circle {
double radius = 1; /** The radius of this circle */
        double getArea(){ /** Return the area of this circle */
        return Math.PI * radius * radius;
        }

        double getPerimeter(){ /** Return the perimeter */
        return 2 * Math.PI * radius;
        }

        void setRadius(double newRadius){ /** Set new radius */
        radius = newRadius;
}
}
```

Accessing Objects via reference variable

- An object's data and methods can be accessed through the dot (.) operator via the object's reference variable.
- A class is essentially a programmer-defined type. A class is a reference type, which means that a variable of the class type can reference an instance of the class.
- The following statement declares the variable myCircle to be of the Circle type:
- Circle myCircle;
- The variable myCircle can reference a Circle object.

Accessing Objects via reference variable

- The next statement creates an object and assigns its reference to myCircle:
- myCircle = new Circle();
- You can write a single statement that combines the declaration of an object reference variable, the creation of an object, and the assigning of an object reference to the variable with the following
- syntax:
- Circle myCircle = new Circle();
- The variable myCircle holds a reference to a Circle object.

Accessing Object

```
public class TestObject{
public static void main(String[] args){

/** Create an object of the Circle class */
Circle myCircle = new Circle();

/** Invoke a method using object reference variable */
System.out.println(myCircle.getArea());
}
}
```

Constructors

- A Java class uses variables to define data fields and methods to define actions.
- Additionally, a class provides methods of a special type, known as constructors, which are invoked to create a new object.
- A constructor can perform any action, but constructors are designed to perform initializing actions, such as initializing the data fields of objects.

Constructors

- Two types
    - Default constructor
    - Parameterized constructor

Constructing Objects Using Constructors

- A constructor is invoked to create an object using the new operator.
- Constructors are a special kind of method. They have following properties:
- A constructor must have the same name as the class itself.
- Constructors do not have a return type, not even void.
- Constructors are invoked using the new operator when an object is created. Constructors play the role of initializing objects.
- Cannot be inherited and cannot be virtual

Default Constructor

```
class Main{

//creating a default constructor
Main()
{System.out.println("Default constructor created");}

//main method
public static void main(String args[]){

//calling a default constructor
Main b=new Main();
}
}
```

Default Constructor

```
class box
{
          double w, h, d;
          box() //Default constructor
          {
          System.out.println("Constructing box");
                    w=10;
                    h=10;
                    d=10;
          }
          double volume()
          {
                    double v;
                    v=w*h*d;
          System.out.println("volume="+v);
                    return 0;
          }
}
```

```
class def_const1
{
          public static void main(String args[])
          {
                    box b1 = new box();
                    b1.volume();
          }
}
```

Parameterized Constructor

```
class Main{

//creating a parameterized constructor
Main(int a)
{System.out.println(“Parameterized constructor is created:" +a);}

//main method
public static void main(String args[]){

//calling a parameterized constructor
Main b=new Main(2);
}
}
```

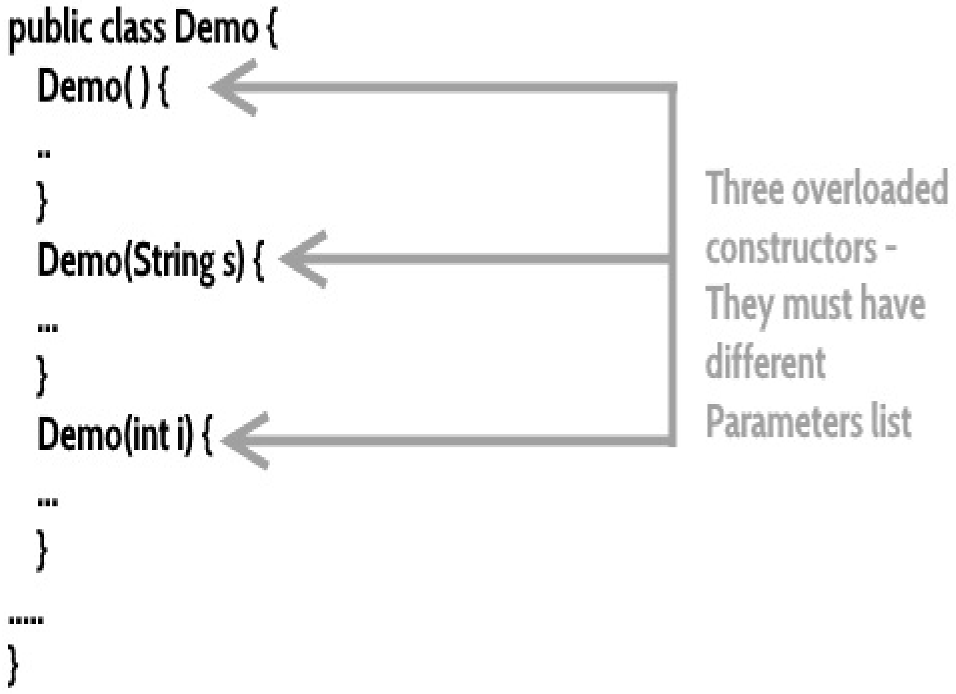
Parameterized Constructor

Ex: Parameterized constructor

```
class box1
{
		double w, h, d;
		box1(double wi, double ht, double de)
		{
				w=wi;
				h=ht;
				d=de;
		}
		double volume()
		{
				double v;
				v=w*h*d;

		System.out.println("volume="+v);
				return 0;
		}
}
```

```
class const_param1
{
		public static void main(String args[])
		{
			box1 b2 = new box1 (10,20,30);
			b2.volume();
		}
}
```

Constructor Overloading



```
public class Demo {
   Demo( ) {
   ..
   }
   Demo(String s) {
   ...
   }
   Demo(int i) {
   ...
   }
.....
}
```

Three overloaded constructors - They must have different Parameters list

```
class Temp //Constructor Chaining
{
   Temp()   {
      // calls constructor 2
      this(5);
      System.out.println("The Default constructor");
   }

      Temp(int x){
      // calls constructor 3
      this(5, 15);
      System.out.println(x);
   }
       Temp(int x, int y)   {
      System.out.println(x * y);
   }
    public static void main(String args[])
   {
      // invokes default constructor first
      new Temp();
   }}
```

Constructor Overloading

```
public class Main {
String name;
Main(){
System.out.println("this a default constructor");
}
 Main(String n){
name = n;
}

public static void main(String[] args) {
Main s = new Main();
System.out.println("\nDefault Constructor values: \n");
System.out.println("\nStudent Name : "+s.name);

System.out.println("\nParameterized Constructor values: \n");
Main student = new Main("David");
System.out.println("\nStudent Name : "+student.name);
}
}
```

Destructor

- Is destructor available in java?
- No destructor in java.
- Garbage collector in java performs
  - i) freeing up memory allocated for objects and
  - ii) Cleaning up resources.

Java Finalize() Method

- We can use the object.finalize() method which works exactly like a destructor in Java.
- An Object.finalize() method is inherited in all Java objects. It is not a destructor but is used to make sure or provide additional security to ensure the use of external resources like closing the file, etc before the program shuts down.

Java Finalize() Method

```
public class A {

public void finalize() throws Throwable{
System.out.println("Object is destroyed by the Garbage Collector");
}

public static void main(String[] args) {

A test = new A();
test = null;
System.gc();
}
}
```

Destructor

- Is destructor available in java?
- No destructor in java.
- Garbage collector in java performs
  - i) freeing up memory allocated for objects and
  - ii) Cleaning up resources.

Java Finalize() Method

- We can use the object.finalize() method which works exactly like a destructor in Java.
- An Object.finalize() method is inherited in all Java objects. It is not a destructor but is used to make sure or provide additional security to ensure the use of external resources like closing the file, etc before the program shuts down.

Reference Data Fields and the NULL Value

- The data fields can be of reference types.

```
class Student {
String name; // name has the default value null
int age; // age has the default value 0
boolean isScienceMajor; // default value false
char gender; // default value '\u0000'
}
```

- If a data field of a reference type does not reference any object, the data field holds a special Java value, null.
- null is a literal just like true and false.
- While true and false are Boolean literals, null is a literal for a reference type

Reference Data Fields and the NULL Value

```
public class Main {
        char name; //unassigned
                public void print(){
                System.out.println("Default value of char num: "+ name);
        }

        public static void main(String[] args) {
                 Main d = new Main();
                d.print();
        }

}
```

Using Classes from the Java Library

The Java API contains a rich set of classes for developing Java programs.
For examples:
Math, Arrays, Scanner, String, Date, Random, Point2D etc...

Using Classes from the Java Library

The Java API contains a rich set of classes for developing Java programs.
For examples:
Math, Arrays, Scanner, String, Date, Random, Point2D etc...

Math Class and Arrays Class

- Math Class:
- Unit#2

- Arrays Class:
- Unit#3

- Scanner Class
- Unit#2 and Unit#3

Date Class

- Date(year,month,date,[hrs,min,[sec]])
- Date()
- Date(elapseTime)
- after(Date when)
- before(Date when)
- compareTo(Date anotherDate)
- equals(Object obj)
- getTime()
- setTime(elapseTime)
- toString()

Date Class

```
import java.util.*;
public class Main
{
    public static void main(String[] args)
    {
        // Creating date
        Date d1 = new Date();  // Current date
        System.out.println("Current Date:" +d1);

        Date d2 = new Date(2030, 11, 21);

        Date d3 = new Date(2022, 6, 3);
          boolean a = d3.after(d2);
        System.out.println("Date d3 comes after " + "date d2: " + a);
    }}
```

String Class

- String is a collection of characters.
- Sequence of Unicode characters treated as single unit. In java string defines the object.
- Strings are immutable. Strings are Thread safe.
- The java.lang.String class implements Serializable, Comparable and CharSequence interfaces.

› String Constructor
› String Concatenation
› length() method
› charAt() method
› getChars() method
› Equals() method
› equalsIgnoreCase() method
› startsWith() method
› endsWith() method
› compareTo() method
› indexOf() method with char
› indexOf() method with substring
› substring() method
› concat() method
› replace() method
› trim() method
› Changing the case

› Refer DemoString Example

String Class

- How to create strings?
- Two methods
    1. String literal
        1. Represented as sequence of characters enclosed in double quotes.
        2. Ex: String s = “Hello”

    2. Using new operator
        1. String s = new String(“Hello”);

› Refer DemoString Example

Immutable string (Cannot change object)

- String class represents immutable string that means once one has created a string object it cannot be changed.
- If want to modify string object need to use StringBuffer class.

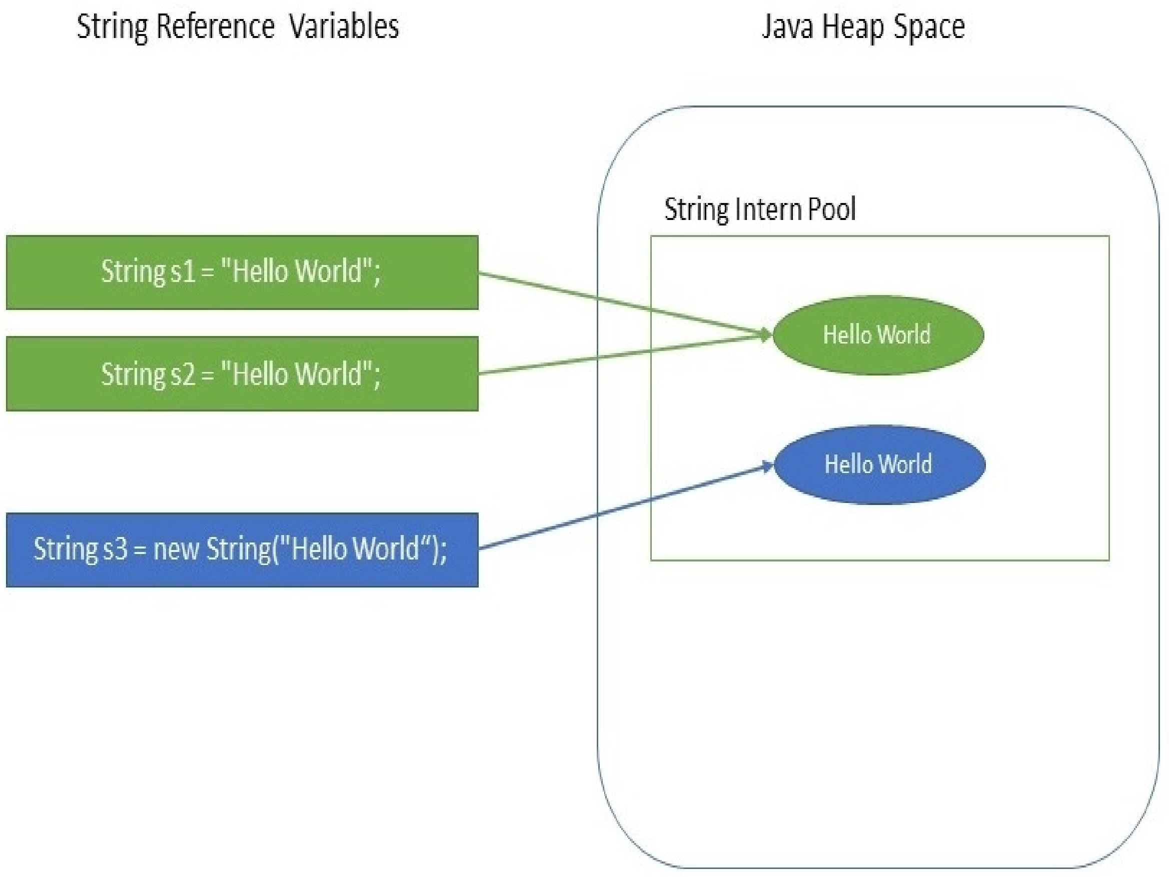
```
class a
{
public static void main (String[] args) {
{
String s = “Hi”;
s.concat(“Hello”); //appends string at end
System.out.println(s); //prints only Hi because strings are immutable.
}}
```

Advantages of Immutable String

- The key benefits of keeping this class as immutable are caching, security, synchronization, and performance.

What is Java String Pool?

Java String Pool is the special memory region where Strings are stored by the JVM. Since Strings are immutable in Java, the JVM optimizes the amount of memory allocated for them by storing only one copy of each literal String in the pool. This process is called interning.


String Reference Variables
Java Heap Space
String Intern Pool
String s1 = "Hello World";
String s2 = "Hello World";
String s3 = new String("Hello World");
Hello World
Hello World

Advantages of Immutable String

String Pool example

```
class Main{
public static void main (String[] args) {
   {
      String s1 = "Hello World";
      String s2 = "Hello World";
      String s3 = "Hello";

      System.out.println(s1.equals(s2));
      System.out.println(s1.hashCode());
      System.out.println(s2.hashCode());
      System.out.println(s3.hashCode());
   }
}
}
```

String Class

```
class Main
{
public static void main(String args[])
{
String s ="Hello";
System.out.println(s.length());

char ch;
ch =s.charAt(2);
System.out.println(ch);


//substring
System.out.println("Substring:"
+s.substring(2,5));
```

```
//replace
String str;
str=s.replace('H','Y');
System.out.println(str);


//lower-upper
String str_lower = s.toUpperCase();
System.out.println(str_lower);
}
}
```

Overriding toString() method

- The default toString() method from object class prints “Class name @ hashcode”.
- Default toString() method can be overridden

```
class Main
{
int rollno;
String sname;

Main(int rollno, String sname)
{
this.rollno=rollno;
this.sname=sname;
}
public String toString() //override
{
return rollno + " " +sname;
}
public static void main(String args[])
{
Main x1 = new Main(101,"Ram");
System.out.println(x1);
//xxx@3e25a5
}}
```

Java StringBuffer Class

- Java StringBuffer class is used to create mutable (modifiable) String objects.
- The StringBuffer class in Java is the same as String class except it is mutable i.e. it can be changed.
- StringBuffer and StringBuilder classes are used for creating mutable strings.

```
class Main{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello ");
sb.append("Java");//now original string is changed
System.out.println(sb);//prints Hello Java
}
}
```

Java StringBuffer Class

```
class Main{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.reverse();
System.out.println(sb);//prints olleH
}
}
```

Static Variables

- A static variable is shared by all objects of the class.
- If you want all the instances of a class to share data, use static variables, also known as class variables.
- Static variables store values for the variables in a common memory location.
- The memory of static fields will be stored in constant pole instead heap organization.
- Only one copy of memory created and shared
- The default value of static field is zero.

4 Ways of using Static Keyword

1) Static field – Using “this” keyword

2) Static method – Using static keyword

3) Static class – Using static keyword in nested class

4) Static block - It is executed before the main method at the time of class loading.

Static variable

```
class Main
{
  static int count=0;
  public void increment()
  {
     count++;
  }
  public static void main(String args[]){
     Main obj1=new Main();
    Main obj2=new Main();
    obj1.increment();
    obj2.increment();
     System.out.println("Obj1: count is="+obj1.count);
     System.out.println("Obj2: count is="+obj2.count);
  }}
```

Static Method

```
class Main
{
        static int age=37;
        static int height=147;

        static void display()
        {
                System.out.println("Age ="+age);
                System.out.println("Height ="+height);
        }
public static void main(String args[])
{
        display(); } }
```

Static Inner Class

```
class Main
{
          static int a=10;
          static class inner
          {
                    void msg()
                    {
                    System.out.println("Value=" +a);
                    }
          }
public static void main(String args[])  {
          Main.inner obj1= new Main.inner();
          obj1.msg(); //msg() is not static so creating object
          }
}
```

Static Block

```
class Main{
  static{System.out.println("static block is invoked");}
  public static void main(String args[]){
   System.out.println("Hello main");
  }
}
```

Static Restrictions

There are two main restrictions for the static method.

They are:

1. The static method can not use non static data member or call non-static method directly.
2. this and super cannot be used in static context.

- Why is the Java main method static?
- It is because the object is not required to call a static method.
- If it were a non-static method, JVM creates an object first then call main() method that will lead the problem of extra memory allocation.

Constants and Methods

A constant is a variable whose value cannot change once it has been assigned. Java doesn't have built-in support for constants.

A constant can make our program more easily read and understood by others. In addition, a constant is cached by the JVM as well as our application, so using a constant can improve performance.

static final datatype identifier_name = constant;

```
static final float pi = 3.14f;
```

Constants and Methods

```
class Demo{

  final int MAX_VALUE=99;
  void myMethod(){
    MAX_VALUE=101;
  }
  public static void main(String args[]){
    Demo obj=new  Demo();
    obj.myMethod();
  }
}
```

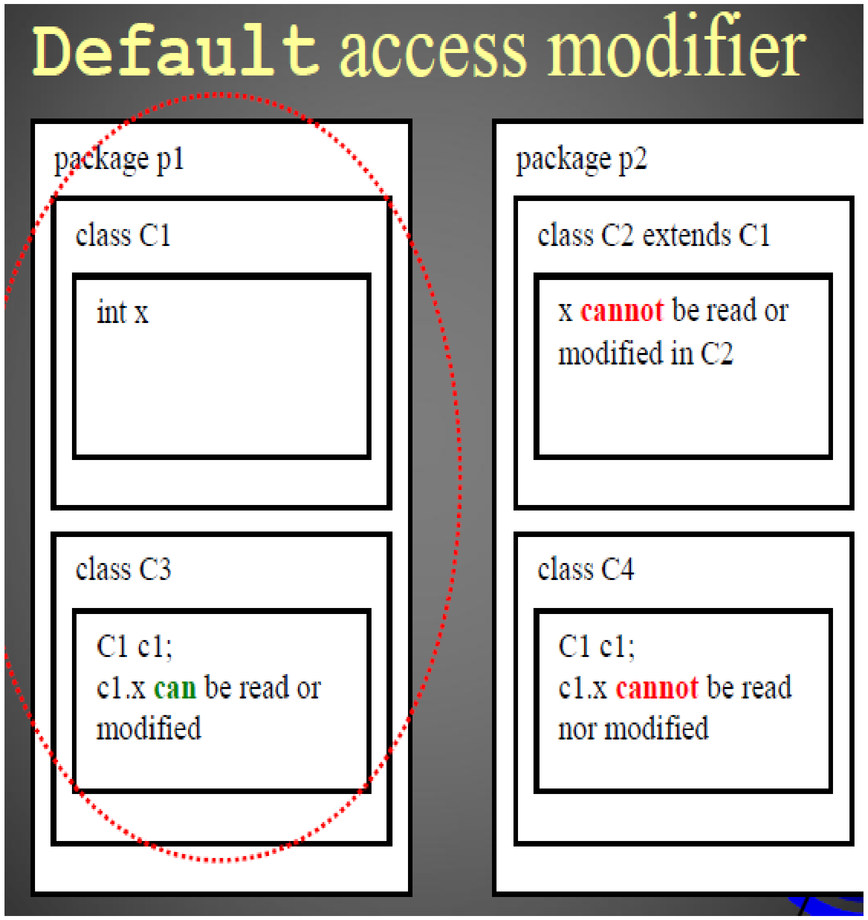
```
class Demo{

  static final int MAX_VALUE=99;
  static void myMethod(){
     MAX_VALUE=101;
   }
   public static void main(String args[]){
   myMethod();
   }
}
```

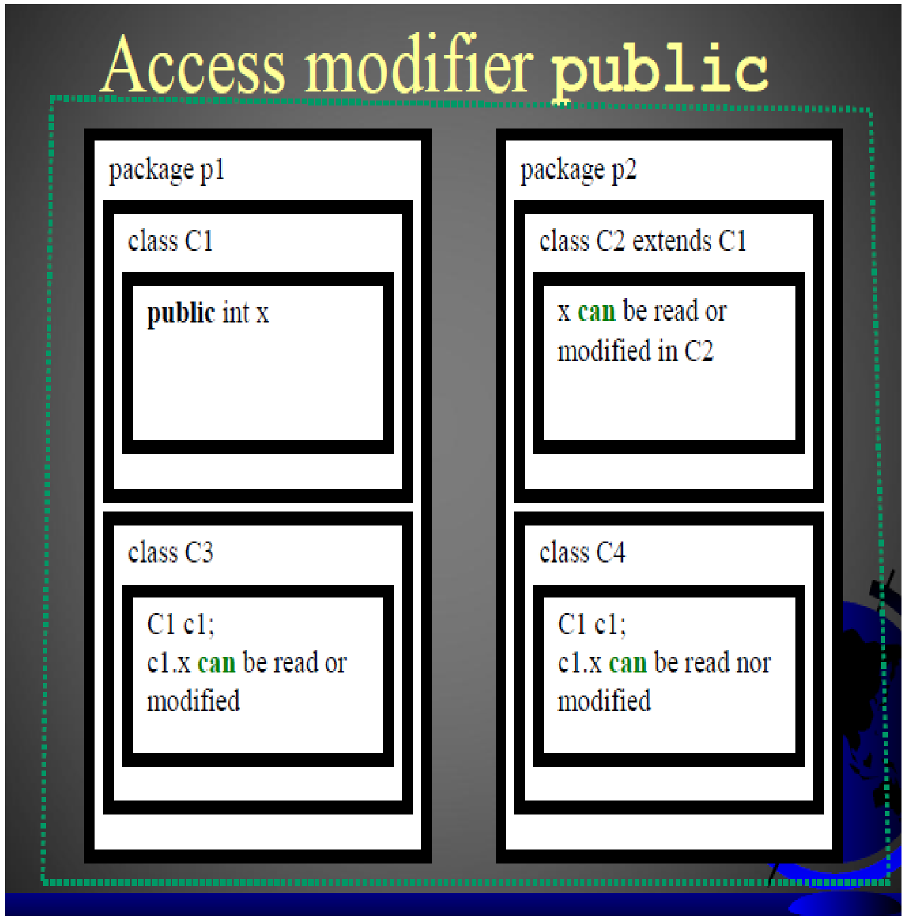
Visibility/Access Modifiers

1. Default
2. Public
3. Protected
4. Private

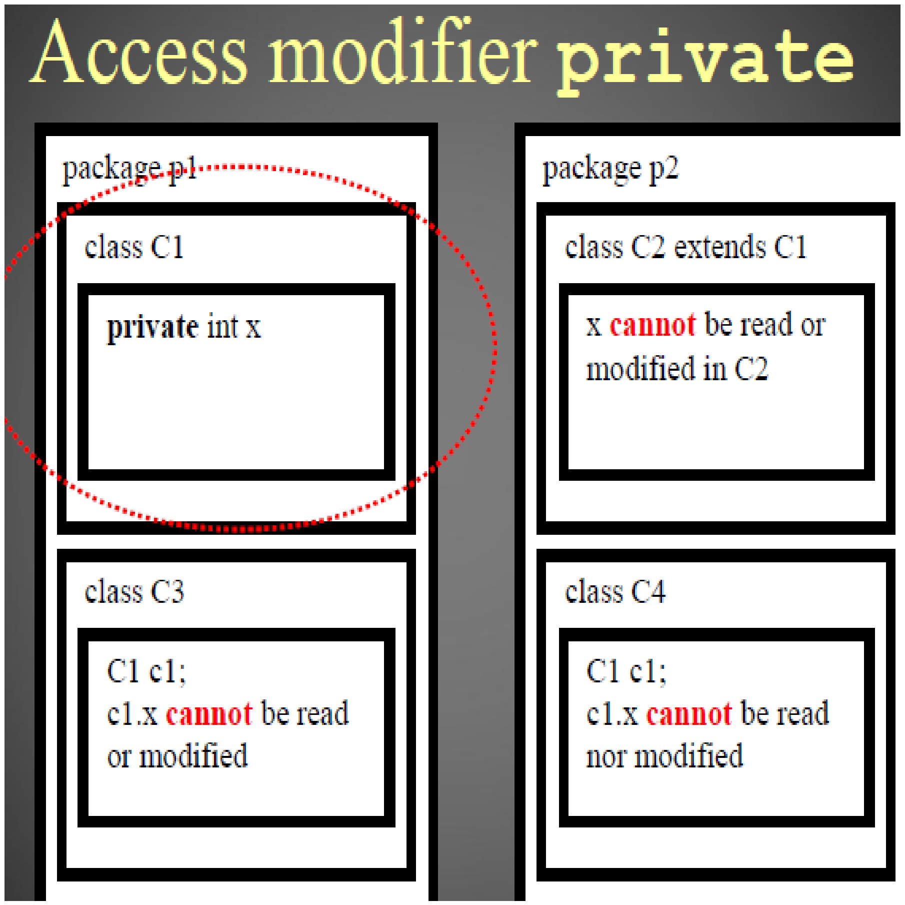
Access Location	Access Modifier			
	Public	Protected	Default	Private
Same class	Yes	Yes	Yes	Yes
Sub class in same package	Yes	Yes	Yes	No
Other classes in same package	Yes	Yes	Yes	No
Subclass in other packages	Yes	Yes	No	No
Non-subclass in other package	Yes	No	No	No

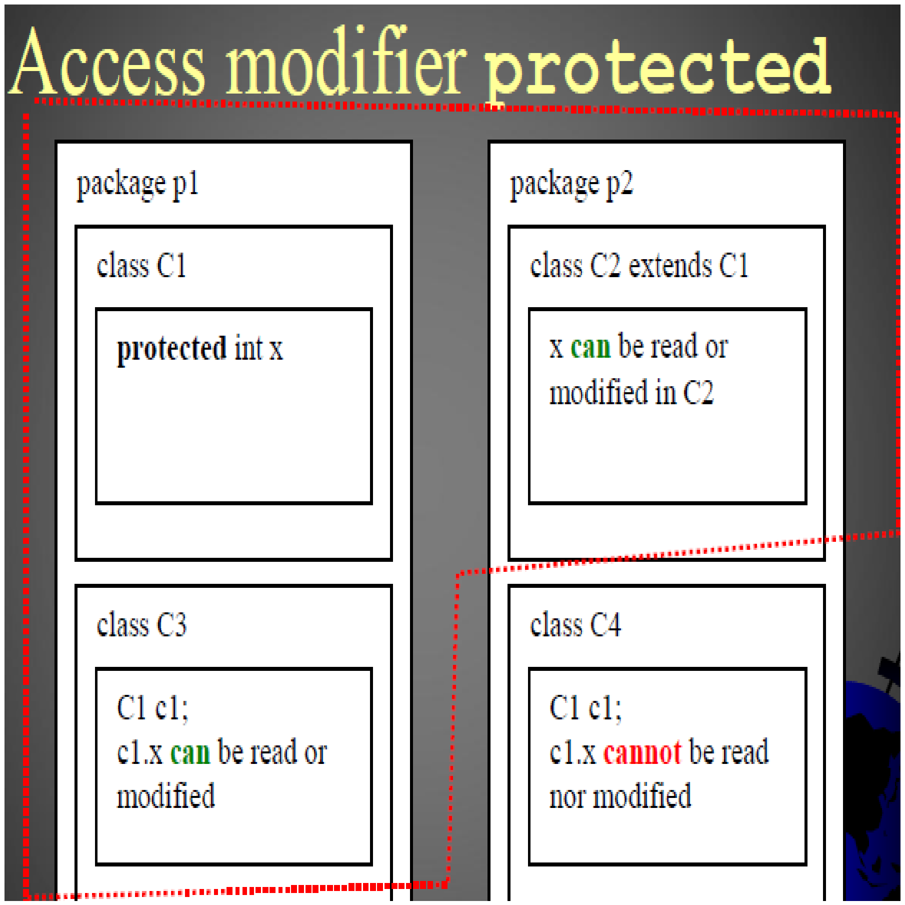
Visibility/Access Modifiers





Visibility/Access Modifiers

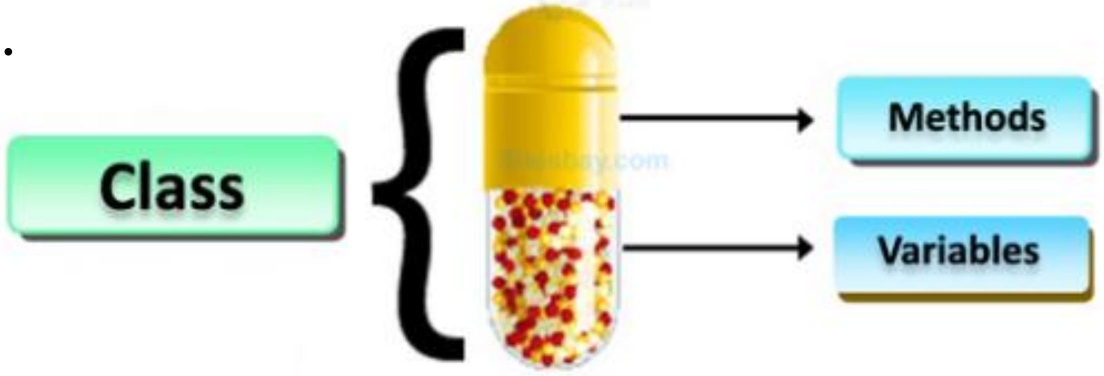



Access modifier protected
package p1
class C1
protected int x
package p2
class C2 extends C1
x can be read or modified in C2
class C3
C1 c1;
c1.x can be read or modified
class C4
C1 c1;
c1.x cannot be read nor modified

Data Field Encapsulation

- Public data fields can be modified directly. So to prevent this encapsulation is used.
- Encapsulation simply means binding object state(fields) and behavior (methods) together. If you are creating a class, you are doing encapsulation.

- The whole idea behind encapsulation is to hide the implementation details from users.
- If a data member is private it means it can only be accessed within the same class. No outside class can access private data member (variable) of other class.



Data Field Encapsulation

- However, if we set up public getter and setter methods to update (for example void setSSN(int ssn))and read (for example int getSSN()) the private data fields then the outside class can access those private data fields via public methods.

- This way data can only be accessed by public methods thus making the private fields and their implementation hidden for outside classes. That’s why encapsulation is known as data hiding. Let’s see an example to understand this concept better.

Data Field Encapsulation

```
class Encapsulate {
private String Name;
public String getName()
{ return Name; }

public void setName(String newName)
   {      Name = newName;      }
}
public class Main{
   public static void main(String[] args){
      Encapsulate obj = new Encapsulate();
         obj.setName("Harsh");
         System.out.println("name: " + obj.getName());
         //System.out.println("name: " + obj.Name); //Cannot access directly
}}
```

Passing Objects to Methods

Passing an object to a method is to pass the reference of the object.

```
class Printer
{
public void print(Paper p){
p.setText("Get Lost");}
}

class Paper
{
String text;
public void setText(String t)
{ text=t;}
public String getText()
{ return text;}
}
```

```
{
public static void main(String[]
args){
Paper p = new Paper();
p.setText("Hello");
System.out.println(p.getText());

Printer np = new Printer();
np.print(p);
System.out.println(p.getText());
}}
```

Array of Objects

- An array can hold objects as well as primitive type values.

- When an array of objects is created using the new operator, each element in the array is a reference variable with a default value of null.

```
class Main {
        public static void main(String args[])
        {
                // Creating an array of objects declared with initial values
                Object[] javaObjectArray = { “abc”, 1234 };
                // Printing the values
                System.out.println(javaObjectArray[0]);
                System.out.println(javaObjectArray[1]);


        }
}
```

Final Keyword

Final keyword can be applied in 3 places,

1. For declaring variables
2. For declaring methods
3. For declaring the class

```
//Final keyword with variable
class Demo{

  final int MAX_VALUE=99;
  void myMethod(){
    MAX_VALUE=101;
  }
  public static void main(String args[]){
    Demo obj=new  Demo();
    obj.myMethod();
  }
}
Output: Compile time error
```

Final Keyword

#final keyword for declaring method

```
Class test{
final void fun()
{
        System.out.println(“function using final”);
}
}
Class test1 extends test{
final void fun()
{
        System.out.println(“another function using final”);
}
}
Output: Error cannot override fun()
```

Final Keyword

#final class to stop inheritance

```
final class test
{
void fun()
{
        System.out.println("function using final");
}
}
Class test1 extends test                    //error  occurs here
{
final void fun()
{
        System.out.println("another function using final");
}
}
```

Immutable Objects and Classes

- **How to Create an immutable class in Java?**

- Declare the class as final so it can’t be extended.
- Make all fields private so that direct access is not allowed.
- Don’t provide setter methods for variables.
- Make all mutable fields final so that its value can be assigned only once.
- Initialize all the fields via a constructor performing deep copy.
- Perform cloning of objects in the getter methods to return a copy rather than returning the actual object reference.

Immutable Objects and Classes

```
//Immutable class
final class Immutable {
private String name;
  Immutable(String name, int date) {
this.name = name;
    this.date = date;
  }
public String getName() {
    return name;
  }
}
class Main {
  public static void main(String[] args) {

    Immutable obj = new Immutable("Programiz", 2011);

    System.out.println("Name: " + obj.getName());
    System.out.println("Date: " + obj.getDate());
  }
}
```

Immutable Objects and Classes

For immutable objects refer strings class or wrapper class or file class. All these three concepts are immutable objects.

The Scope of variables

Scope of a variable is the part of the program where the variable is accessible.

```
public class Main
{
public static void main(String args[])
{
int x=10;
{
//y has limited scope to this block only
int y=20;
System.out.println("Sum of x+y = " + (x+y));
}
//here y is unknown
y=100;
//x is still known
x=50;
}
}
```

To run the program remove or comment y=100

This (keyword) Reference

The 6 usage of java this keyword.

- this can be used to refer current class instance variable.
- this can be used to invoke current class method (implicitly)
- this() can be used to invoke current class constructor.
- this can be passed as an argument in the method call.
- this can be passed as argument in the constructor call.
- this can be used to return the current class instance from the method.

This (keyword) Reference

```
class Student{
int rollno;
String name;
Student(int rollno,String name,){
this.rollno=rollno;
this.name=name;
}
void display(){System.out.println(rollno+" "+name);}
}
class Main{
public static void main(String args[]){
Student s1=new Student(111,"ankit");
s1.display();
}}
```

Marwadi
education foundation

END OF UNIT - 4