Class, Object, Packages and Input/Output

Course: CSL304

Prof. Juhi Ganwani

Syllabus

1. Class, object, method
2. Constructor, static member & methods
3. Method overloading
4. Packages in Java, creating user defined packages, access specifiers
5. Input & Output functions in Java
6. Buffered reader class, scanner class

Classes

Syntax:

```
class ClassName [extends SuperClassName]
{
        [variables declarations;]
        [methods declaration;]
}
```

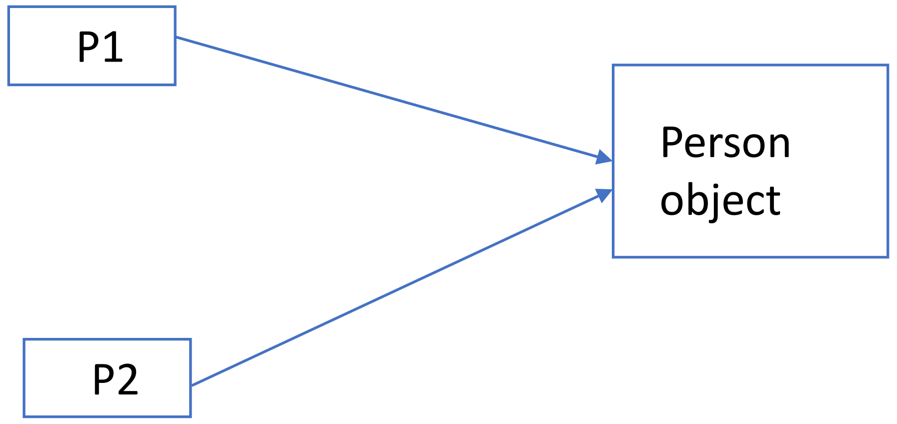
Creating an object

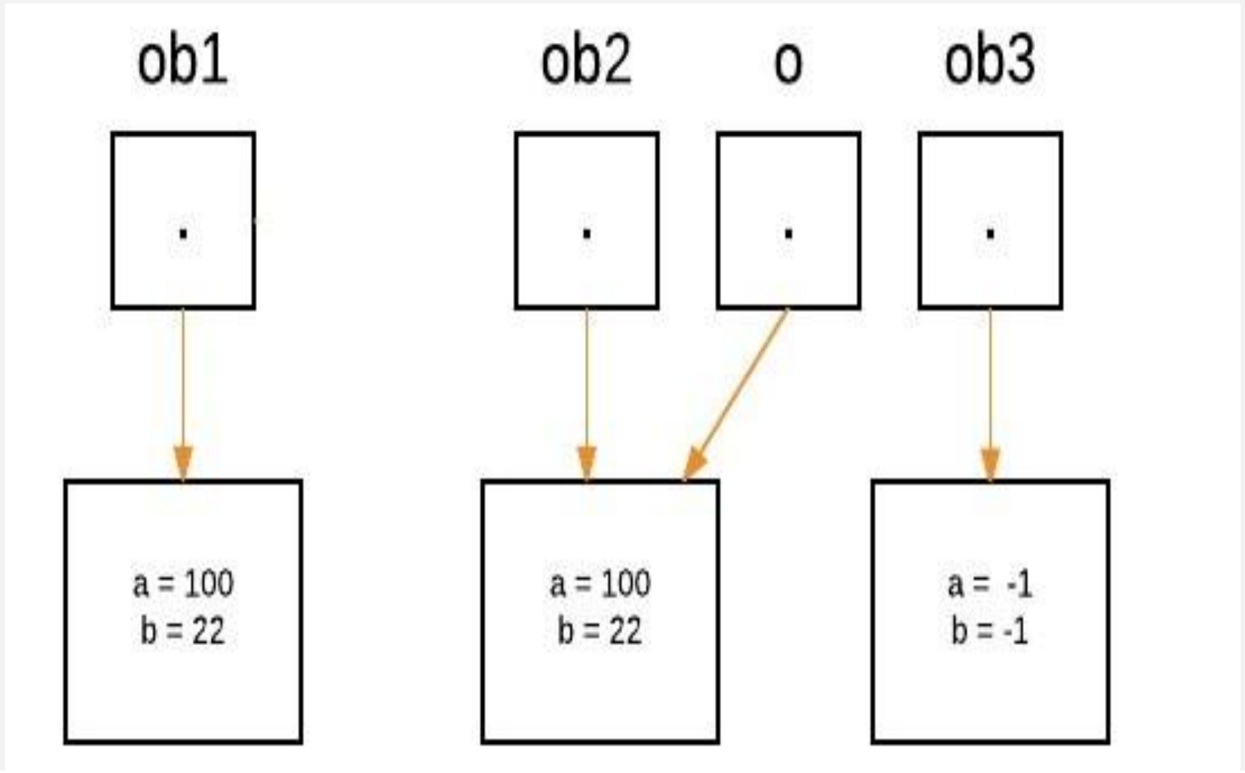
```
ClassName objectname = new ClassName();
```

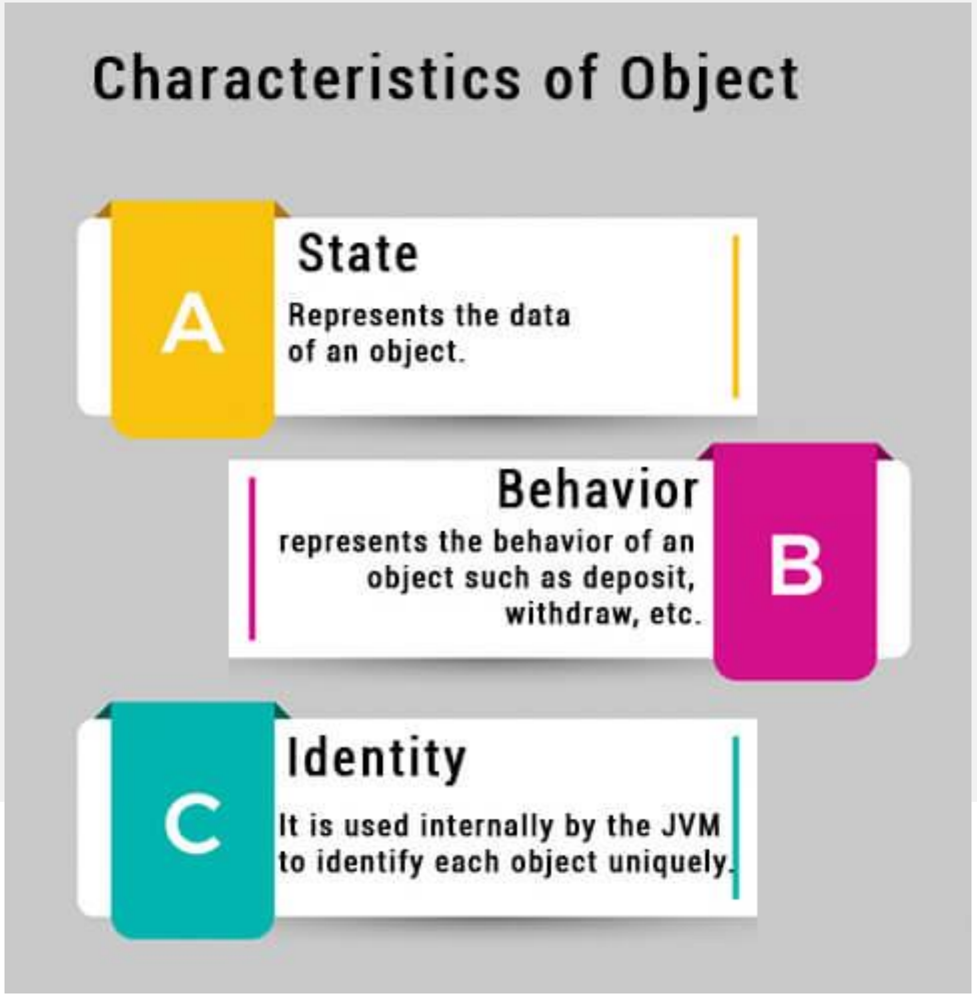
Objects

- Entity that has state & behaviour
- Creating an object is also referred as instantiating an object
- It is creating using a new operator

```
Person p1;
P1=new Person();

Person p2=p1;
```





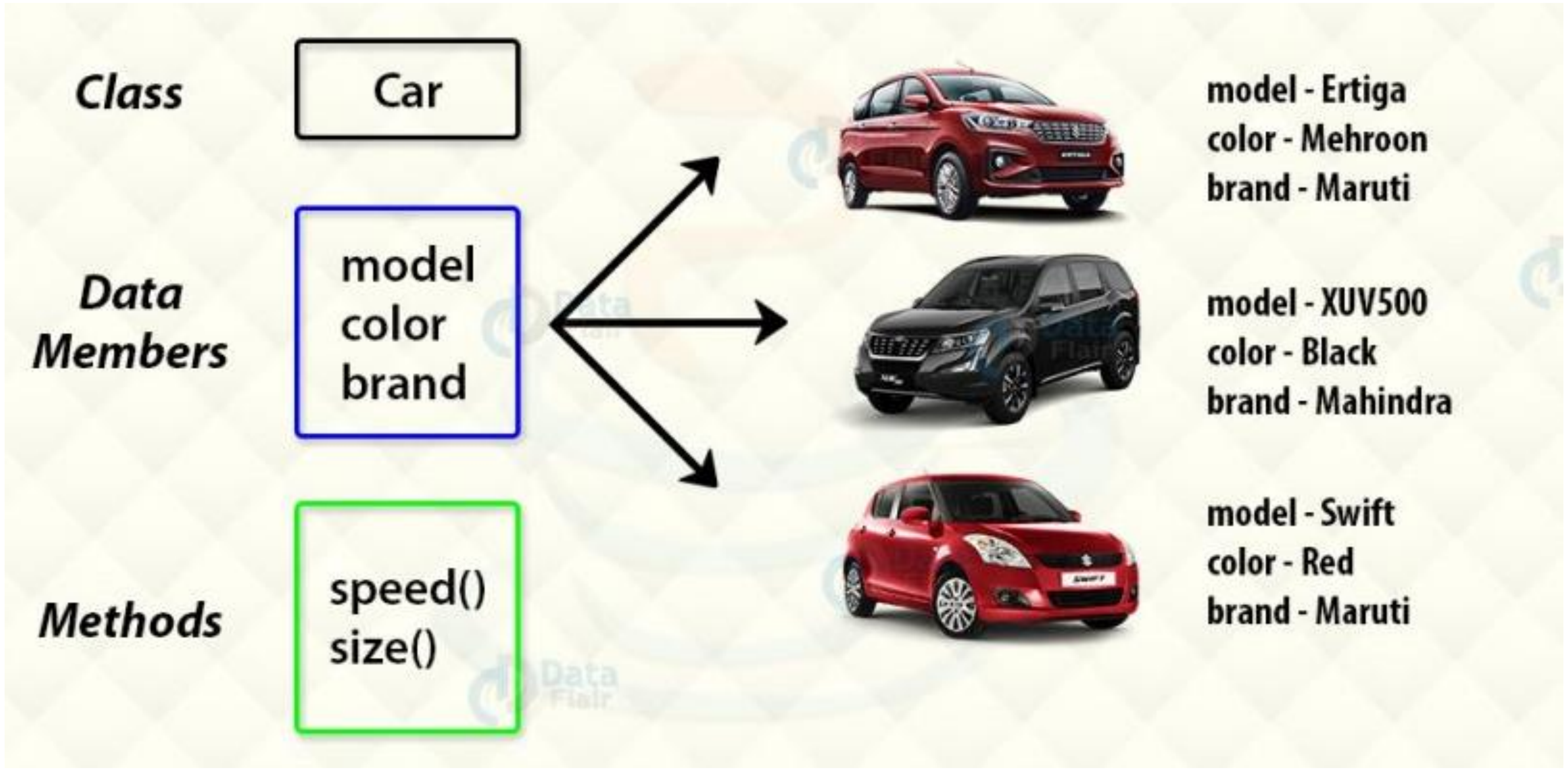

Characteristics of Object

A State
Represents the data of an object.

B Behavior
represents the behavior of an object such as deposit, withdraw, etc.

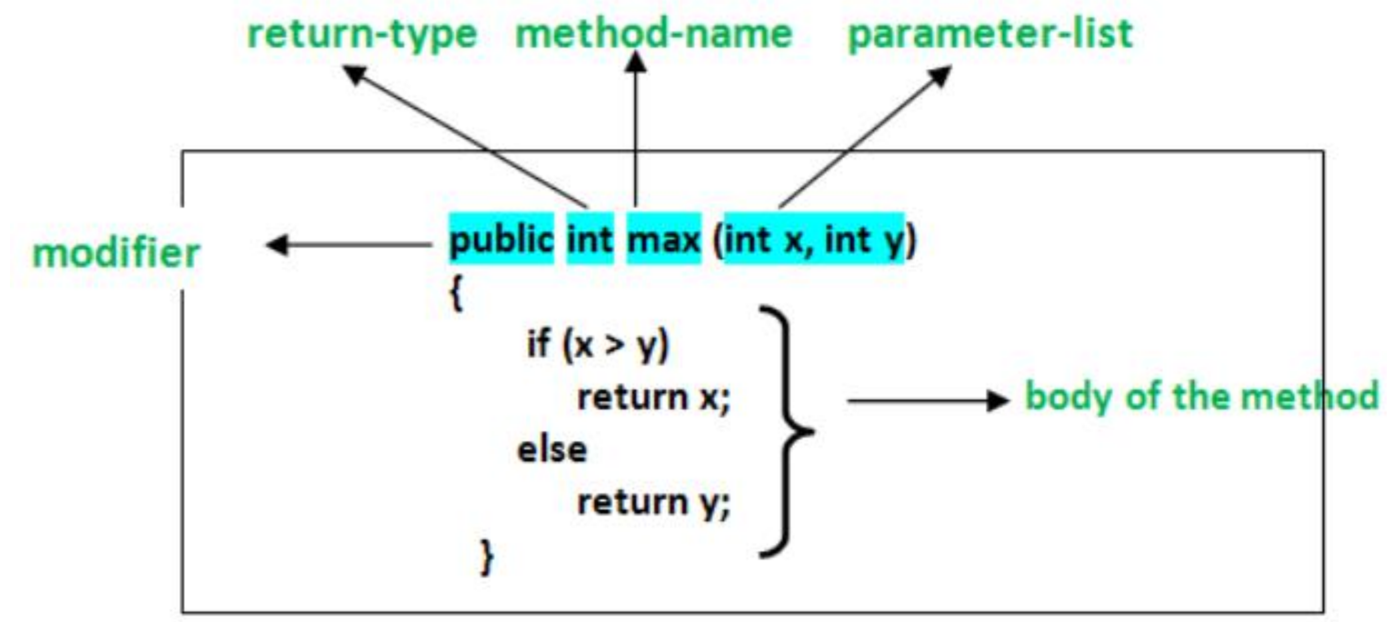
C Identity
It is used internally by the JVM to identify each object uniquely.

Classes & Objects



Methods

- Collection of statements
- Perform specific task
- Allows reusability



```
public int max (int x, int y)
{
     if (x > y)
          return x;
     else
          return y;
 }
```

How to Initialize object in Java?

There are three ways to initialize an object in Java.

1. Using Constructor
2. Using Reference variable
3. By using Method

Using reference variable

```
class Student
{
 int id;
 String name;
}
class TestStudent1
{
 public static void main(String args[]){
  //Creating objects
  Student s1=new Student();
  Student s2=new Student();
  //Initializing objects
  s1.id=101;
  s1.name=“abc";
s2.id=102;
s2.name=“xyz";
 //Printing data
 System.out.println(s1.id+" "+s1.name);
 System.out.println(s2.id+" "+s2.name);
 }
}
```

Using Method

```
class Student
{
 int id;
 String name;
 void insertRecord(int i, String n)
{
  id=i;
  name=n;
 }
 void displayInformation()
{System.out.println(id+" "+name);}
}
```

```
class TestStudent2{
 public static void main(String args[]){
  Student s1=new Student();
  Student s2=new Student();
  s1.insertRecord(101,“abc");
  s2.insertRecord(102,“xyz");
  s1.displayInformation();
  s2.displayInformation();
 }
}
```

Constructors

- It is a special type of a method
- Used to initialize the objects, when created
- Simpler & more concise to initialize an object

Rules:

1. It have same name as the class itself
2. They do not specify a return type, because they return the instance of the class itself

Using Constructor

```
class Student
{
 int id;
 String name;
 Student( int i, String n)
{
id=i;
name=n;
}
}
```

```
class TestStudent3
{
public static void main(String args[]){
 Student s1=new Student(101,”abc”);
 System.out.println(s1.id+" "+s1.name);
}
}
```

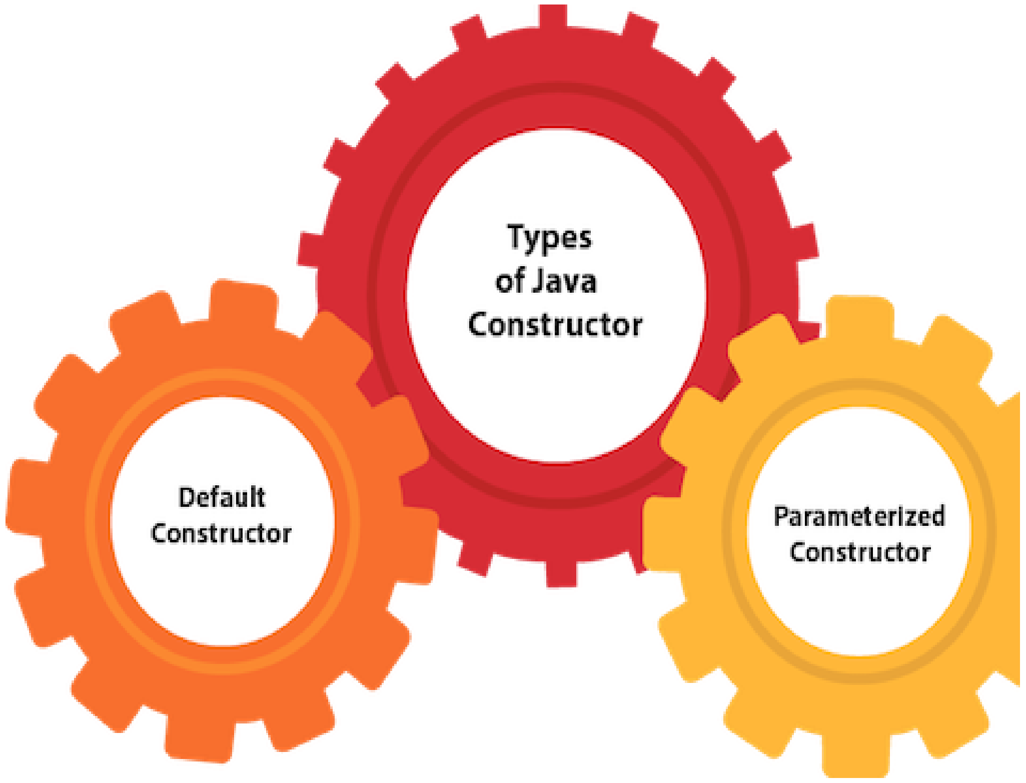
Types of constructors

Default/Simple constructor

- Without parameters
- When no constructor is created explicitly then java first implicitly creates a constructor without any parameter
- Initialize the instance variables to default values

Parameterized constructor

- Constructors having parameters
- Initialize the instance variables with different values



Constructor overloading

- We can create & use more than one constructor in a class
- It can be overloaded on the basis of following facts:

1. No. of parameters
2. Data types of parameters
3. Order of parameters

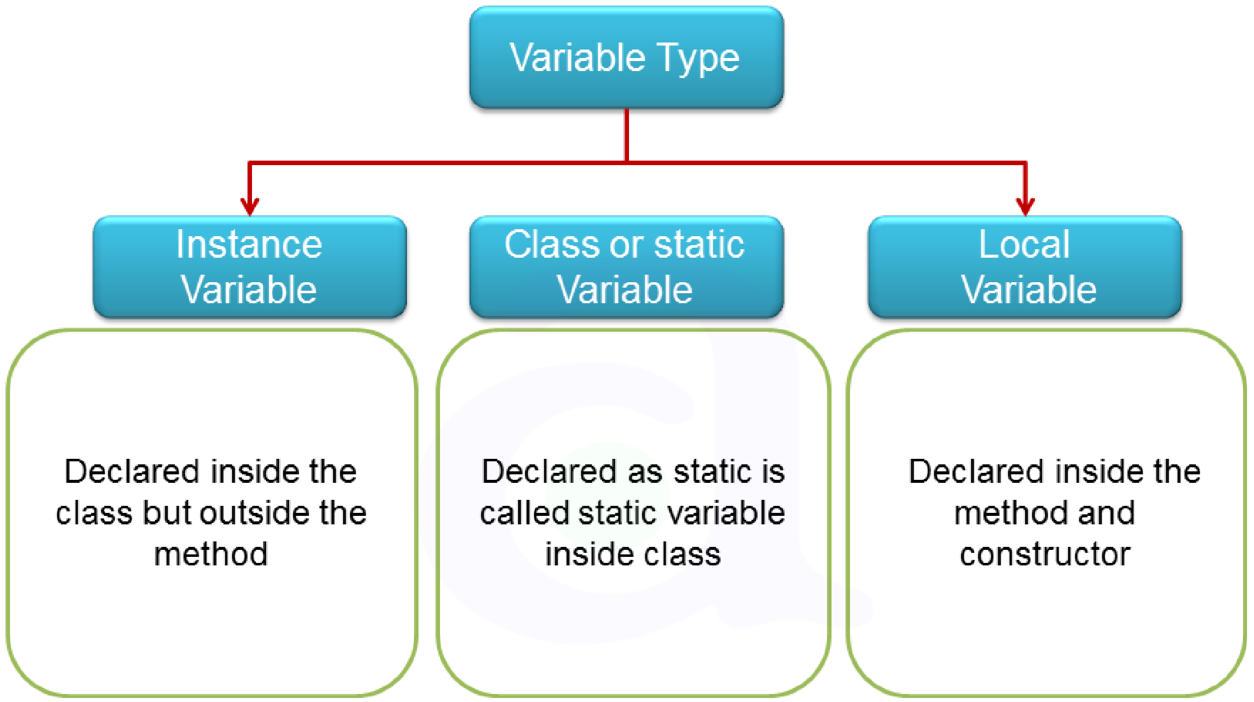
- Instance variables are initialized according to the constructor signature

Constructor overloading

```
class Rectangle
{
double length;
double width;
 public Rectangle()                    //default
{ length = 0.0; width = 0.0; }
public Rectangle(double l, double w) //parameterized
{ length = l; width = w; }
public Rectangle(double x)          //Overloading
{length= width =x;}
public double getPerimeter()
{ return 2 * (length + width); }
}
```

```
class RectangleDemo
{
public static void main(String[] args)
{
Rectangle r1 = new Rectangle();
System.out.println("Perimeter of Rectangle is " +
r1.getPerimeter());
Rectangle r2 = new Rectangle(3.5, 4.2);
System.out.println("Perimeter of Rectangle is " +
r2.getPerimeter());
Rectange r3=new Rectangle(4);
System.out.println("Perimeter of Rectangle is " +
r3.getPerimeter());
}
}
```

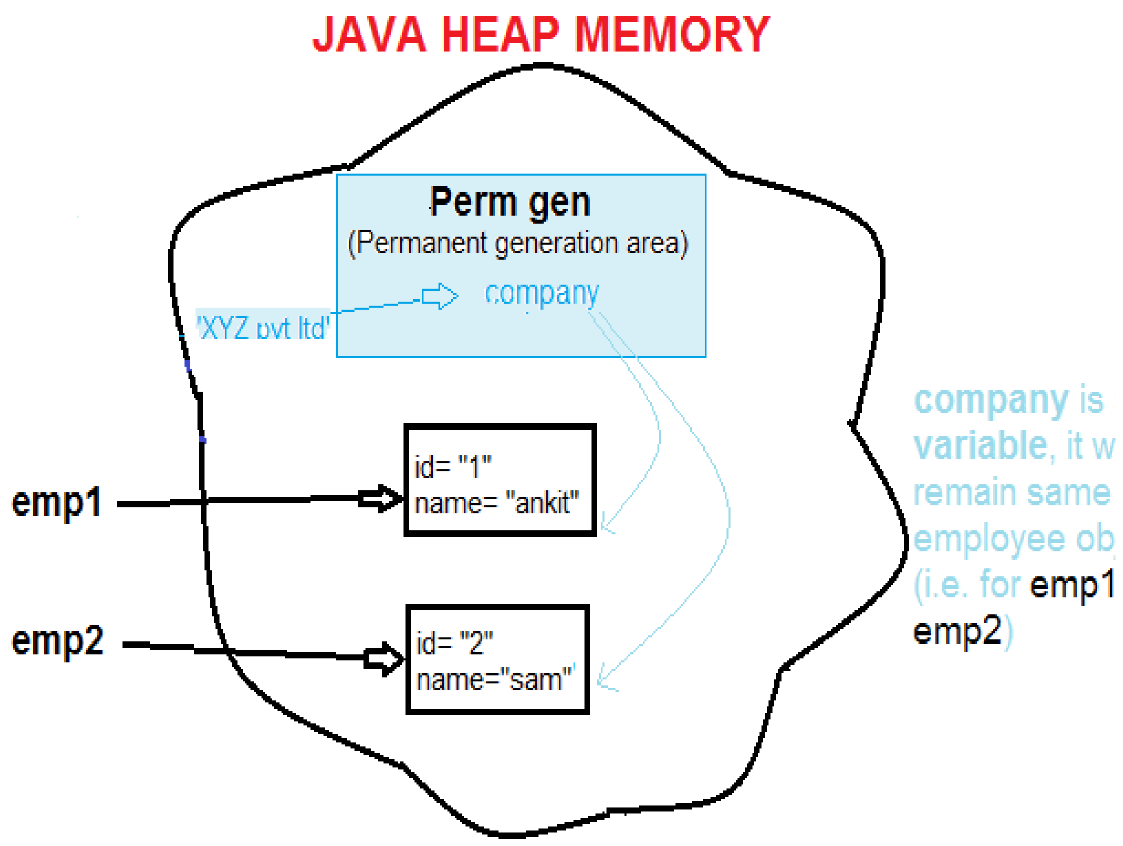
Types of variables


Variable Type
Instance Variable
Class or static Variable
Local Variable
Declared inside the class but outside the method
Declared as static is called static variable inside class
Declared inside the method and constructor

- **Instance variables** - variables are created when an object of the class is created and destroyed when the object is destroyed. Its default value is 0.
- **Static variables** - created at the start of program execution and destroyed automatically when execution ends. we can only have one copy of a static variable per class irrespective of how many objects we create. Its default value is 0.
- **Local variables** – defined within a block or method or constructor. Scope of these variables exists only within the block in which the variable is declared. Initialization is mandatory.

Static variables

- It can be used to refer to the common property of all objects (not unique for each object)
- It gets memory only once in the class area at the time of class loading
- Saves memory



```
class Student
{
int rollno;
String name;
static String college="TSEC";
Student(int r, String n)
{ rollno=r; name=n;}
void display()
{
System.out.println(rollno+"
"+name+" "+college);
}
}
```

```
public class TestStaticVariable
{
public static void main(String args[])
{
Student s1=new Student(1,"John");
Student s2=new Student(2,"Patrick");
s1.display();
s2.display();
}
}
```

```
class Counter
{
        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();
}
}
```

Static methods

- It belongs to a class rather than the object of a class
- It can be invoked without the need for creating an instance of a class
- It can access static data member & can change the value of it

Example

```
class Calculate{
  static int cube(int x)
  {
              return x*x*x;
  }

 public static void main(String args[])
{
              int result=Calculate.cube(5);
              System.out.println(result);
  }
}
```

```
class Student{
int rollno;
String name;
static String college="TSEC";
Student(int r, String n)
{ rollno=r; name=n;}

static void change(){
college="Thadomal Shahani Engineering
College";
}
void display(){
System.out.println(rollno+" "+name+"
"+college);}
}
```

```
public class TestStaticVariable
{
public static void main(String args[])
{
Student.change();
Student s1=new Student(1,"John");
Student s2=new Student(2,"Patrick");
s1.display();
s2.display();
}
}
```

Method overloading

- Class has multiple methods having same name but different in parameters
- Used when objects are required to perform similar tasks but using different input parameters
- Process is known as polymorphism
- Different ways to overload the methods
  - By changing no. of arguments
  - By changing the data type

By changing no. of arguments

```
class Adder
{
        static int add(int a,int b)
        {
                return a+b;
        }
        static int add(int a,int b,int c)
        {
                return a+b+c;
        }
}
```

```
class TestOverloading1
{
        public static void main(String[] args)
        {

        System.out.println(Adder.add(11,11));

        System.out.println(Adder.add(11,11,11));
        }
}
```

By changing the data type

```
class Adder
{
        int add(int a,int b)
        {
                return a+b;
        }
        float add(float a,float b)
        {
                return a+b;
        }
}
```

```
class TestOverloading2
{
        public static void main(String[] args)
        {
        Adder a1=new Adder();
        Adder a2=new Adder();

        System.out.println(a1.add(11,11));

        System.out.println(a2.add(1.1,1.1));
        }
}
```

Can we overload java main() method?

- You can have any number of main methods in a class by method overloading. But JVM calls main() method which receives string array as arguments only

Example:

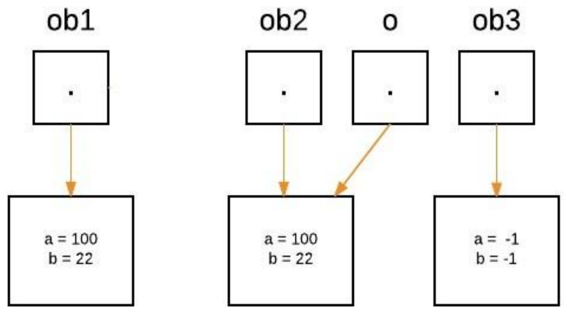
```
class TestOverloading3
{
public static void main(String[] args)
{System.out.println("main with String[]");}
public static void main(String args)
{System.out.println("main with String");}
public static void main()
{System.out.println("main without args");}
}
```

Passing & returning objects

- When we pass a primitive type to a method, it is passed by value.
- But when we pass an object to a method, the situation changes because objects are passed by call-by-reference.
- Changes to the object inside the method do reflect in the object used as an argument.

Objects passing to methods

```
class ObjectPassDemo
{
   int a, b;
   ObjectPassDemo(int i, int j)
   {
      a = i;
      b = j;
   }
   boolean equalTo(ObjectPassDemo o)
   {
      return (o.a == a && o.b == b);
   }
}
```



```
public class ObjectPassMain
{
   public static void main(String args[])
   {
      ObjectPassDemo ob1 = new ObjectPassDemo(100, 22);
      ObjectPassDemo ob2 = new ObjectPassDemo(100, 22);
      ObjectPassDemo ob3 = new ObjectPassDemo(-1, -1);

      System.out.println("ob1 == ob2: " + ob1.equalTo(ob2));
      System.out.println("ob1 == ob3: " + ob1.equalTo(ob3));
   }
}
```

Returning objects

```
class ObjectReturnDemo {
   int a;

   ObjectReturnDemo(int i) {
      a = i;
   }

   ObjectReturnDemo incrByTen() {
      ObjectReturnDemo temp =
         new ObjectReturnDemo(a + 10);
      return temp;
   }
}
```

```
public class ObjectReturnMain {
   public static void main(String args[]) {
      ObjectReturnDemo ob1 = new ObjectReturnDemo(2);
      ObjectReturnDemo ob2;

      ob2 = ob1.incrByTen();

      System.out.println("ob1.a: " + ob1.a);
      System.out.println("ob2.a: " + ob2.a);
   }
}
```

Constructor with object parameter

```
class Box
{
    double width, height, depth;
    Box(Box ob)
    {
        width = ob.width;
        height = ob.height;
        depth = ob.depth;
    }
    Box(double w, double h, double d)
    {
        width = w;
        height = h;
        depth = d;
    }
double volume()
    {
        return width * height * depth;
    }
}
public class BoxMain
{
    public static void main(String args[])
    {

        Box mybox = new Box(10, 20, 15);
        Box myclone = new Box(mybox);
        double vol;
        vol = mybox.volume();
        System.out.println("Volume of mybox is " + vol);
        vol = myclone.volume();
        System.out.println("Volume of myclone is " + vol);
    }
}
```

Packages

- Group of similar types of classes, interfaces and sub-packages
- Used for reusability of a program
- Similar to class libraries in other languages
- Packages act as containers for classes

Advantages:

- Classes contained in the packages of other programs can be easily reused
- Provides access protection.
- Removes naming collision.

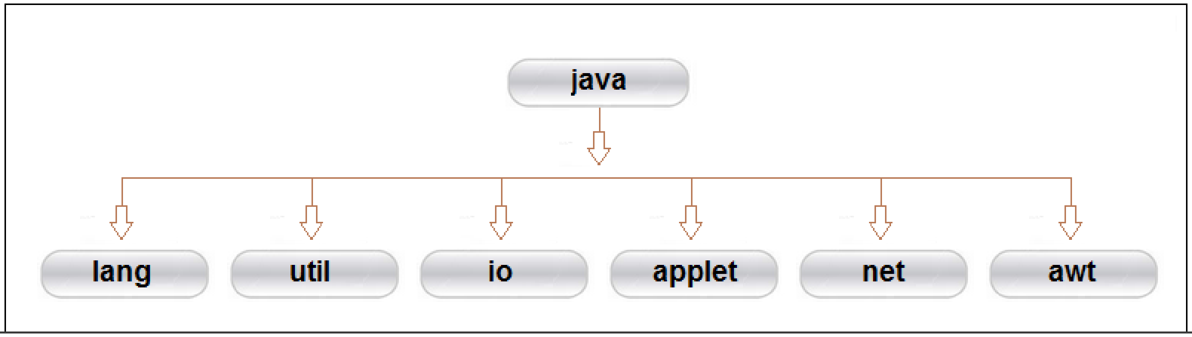
Packages in Java can be categorized in two forms:

1. Built-in packages
2. User defined packages

Built-in packages

- Java API is a library of prewritten classes, that are free to use, included in the Java Development Environment
- Library is divided into packages and classes. Meaning you can either import a single class or a whole package that contain all the classes that belong to the specified package
- To use a class or a package from the library, you need to use import keyword

Built-in packages



java.lang	Language support classes. They include classes for primitive types, string, math functions, thread and exceptions.
java.util	Language utility classes such as vectors, hash tables, random numbers, data, etc.
java.io	Input/output support classes. They provide facilities for the input and output of data.
java.applet	Classes for creating and implementing applets.
java.net	Classes for networking. They include classes for communicating with local computers as well as with internet servers.
java.awt	Set of classes for implementing graphical user interface. They include classes for windows, buttons, lists, menus and so on.

User-defined packages

- Packages created by user
- Packages are created by including a package command followed by name of the package

Syntax:

```
        package packagename;
```

- This must be first statement in java source file
- After this statement, we define a class

eg:

```
package mypack;
public class Employee
{
        String id;
        String name;
}
```

Note

- Package is always defined as a separate folder having the same name as the package name
- Store all the classes in that package folder
- All classes of the package which we wish to access outside the package must be declared public
- All classes within the package must have the package statement as its first line

Package hierarchy

- Allows us to group related packages into a larger package
- This is done by specifying multiple names in a package statement separated by dots

eg: package firstpackage.secondpackage

secondpackage is inside firstpackage

- Accessing a user-defined package is same as predefined packages
- The general format of import statement for searching a class is as follows

import firstpackage[.secondpacakge][.thirdpackage].classname;

```
package package1;
public class ClassA
{
public void displayA()
{
System.out.println(“Class A”);
}
}
```

Note: This file should be stored as ClassA.java in subdirectory package1

```
Javac package1/ClassA.java
```

```
import package1.ClassA;
class PackageTest1
{
public static void main(String args[])
{
        ClassA objectA=new ClassA();
        objectA.displayA();
}
}
```

Note:

We can also write following statement instead of import statement

```
package1.ClassA objectA=new package1.ClassA();
```

We can also use multiple import statements

```
package package2;
public class ClassB
{
public void displayB()
{
System.out.println("Class B");
}
}
```

```
import package1.ClassA;
import package2.*;
class PackageTest2
{
        public static void main(String args[])
        {

                ClassA objectA=new ClassA();
                ClassB objectB=new ClassB();
                objectA.displayA();
                objectB.displayB();
        }
}
```

When we import multiple packages it is possible, that two or more classes have same name. In that case, we have to be more explicit about which one we intend to use

```
package pack1;
public class Teacher
{…}
public class Student
{…}
package pack2;
public class Courses
{…}
public class Student
{…}
```

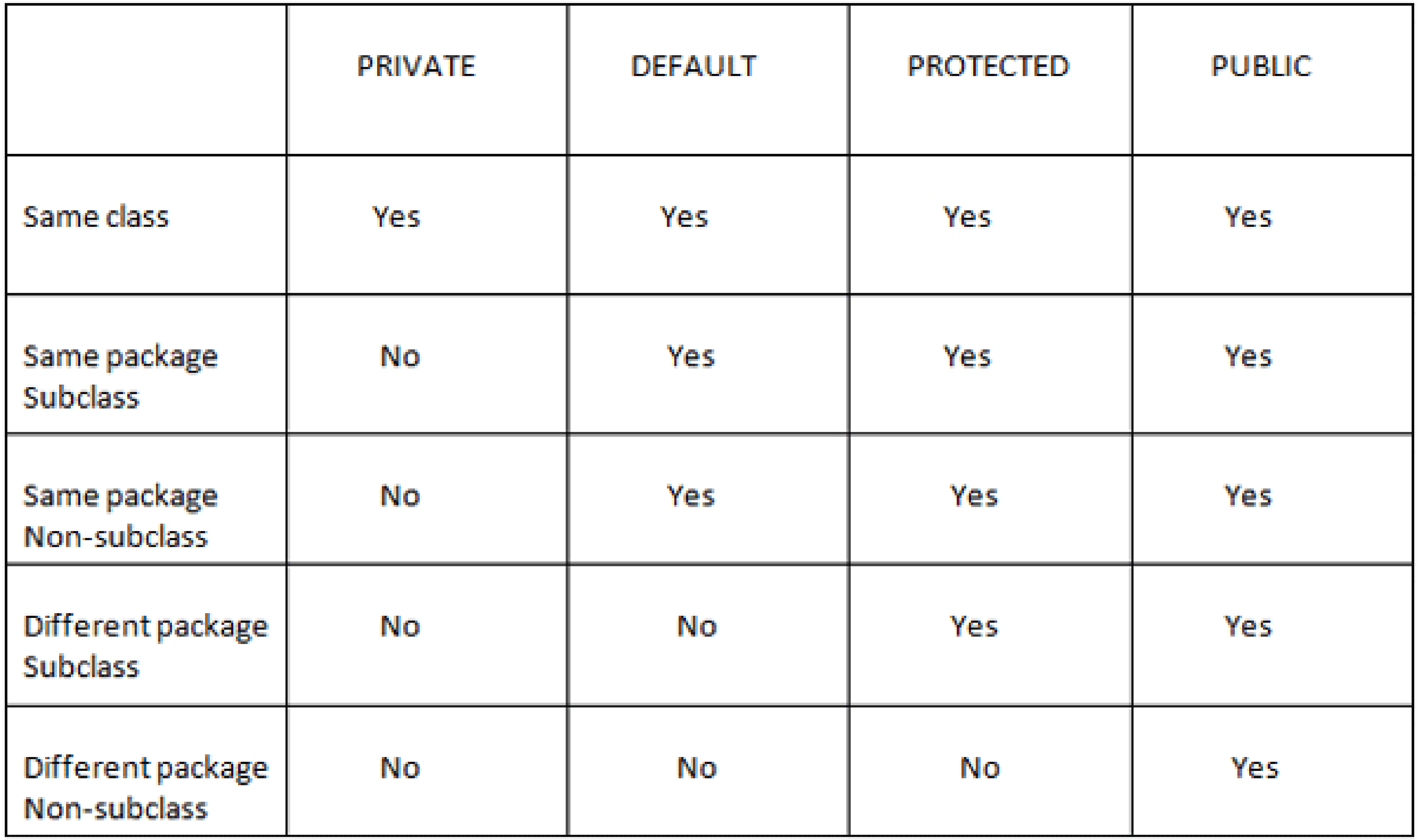
```
import pack1.*;
import pack2.*;

Student s1;           //Invalid
pack1.Student s1      //valid
Teacher t1            //valid
```

Sub-packages

```
package letmecalculate;
public class Calculator
{
        public int add(int a, int b)
        {
                return a+b;
        }
        public static void main(String args[])
        {
                Calculator obj = new Calculator();
                System.out.println(obj.add(10, 20));
        }
}
```

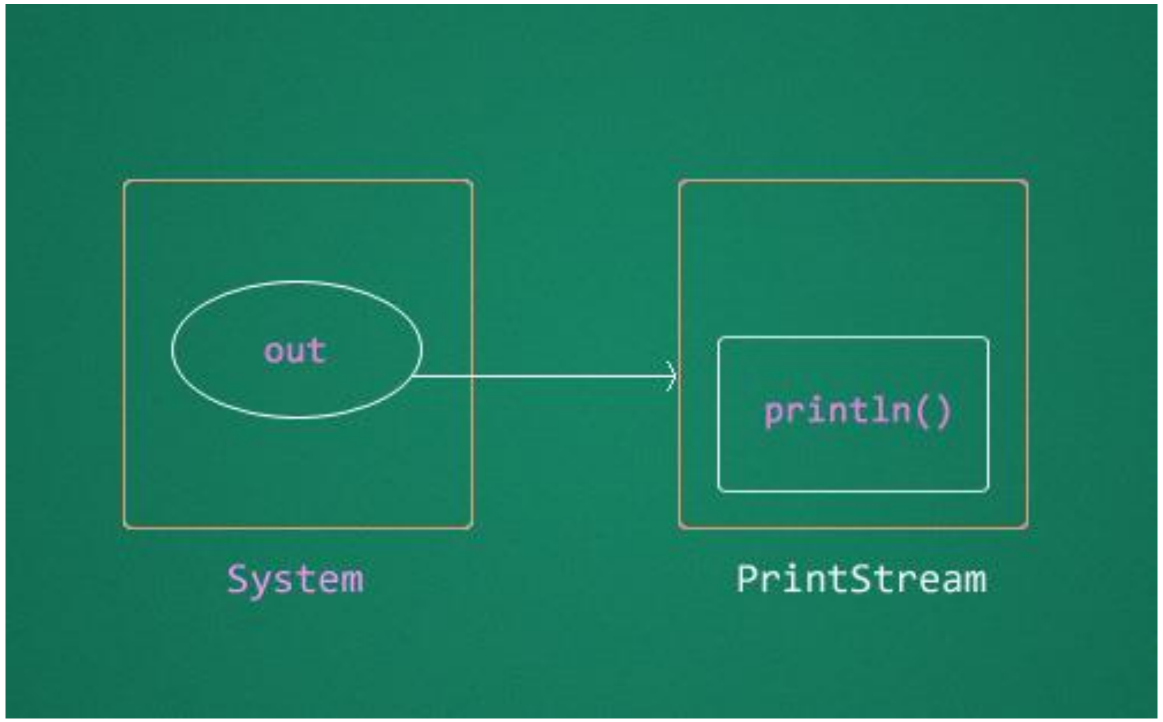
```
package letmecalculate.multiply;
public class Multiplication
{
        public int product(int a, int b)
        {
                return a*b;
        }
}
import letmecalculate.Calculator;
import letmecalculate.multiply.Multiplication;
class letmecalculateMain
{
        public static void main(String args[])
        {
                Calculator obj1 = new Calculator();
                System.out.println(obj1.add(10, 20));
                Multiplication obj2 = new Multiplication();
                System.out.println(obj2.product(10,20));
        }
}
```

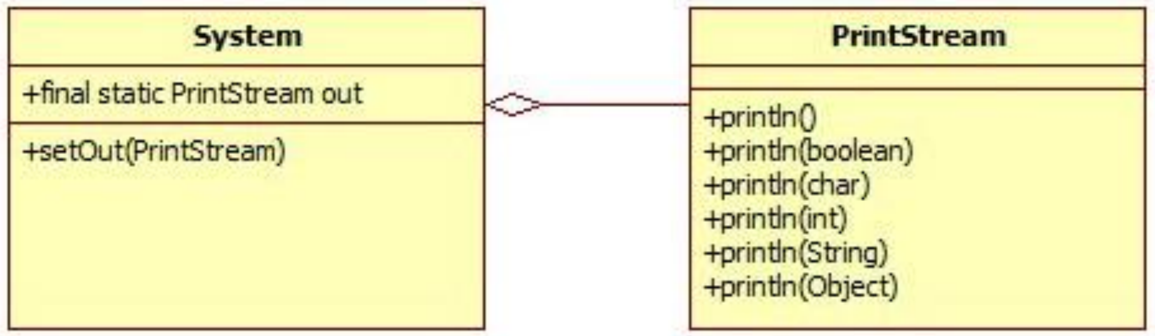
Access Specifiers



	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

System.out.println


out
println()
System
PrintStream


System
+final static PrintStream out
+setOut(PrintStream)
PrintStream
+println()
+println(boolean)
+println(char)
+println(int)
+println(String)
+println(Object)