In the following code, shown the initialize and processing of two dimensional array as printing**.**

Suppose:
int x[][]={{1,2,3},{2,3},{1,2,3,4,5}};

Steps for print x by using for loop

```
for (int i=0;i<x.length;i++){
  for (int j=0; j<x[i].length;j++)
    System.out.print(x[i][j]+" ");
  System.out.println();}
```

Steps for print array by using for each loop

```
for (int i[]:x){
   for (int j:i)
     System.out.print(j+" ");
   System.out.println();}
```

Jagged Array in Java

Jagged array is array of arrays such that member arrays can be of different sizes, i.e., we can create a 2-D arrays but with variable number of columns in each row.
Examples for declare jagged array:

1.

```
int arr[][] = new int[4][];
arr[0] = new int[1];
arr[1] = new int[2];
arr[2] = new int[3];
arr[3] = new int[4];
```

2.

```
int arr[][] = new int[2][];
arr[0] = new int[3];        // First row has 3 columns
arr[1] = new int[2];        // Second row has 2 columns
```

3.

```
int r = 5;
int arr[][] = new int[r][];
 //Ceating a 2D array,first row  has 1 element, second row has 2  elements and so on
for (int i=0; i<arr.length; i++)
  arr[i] = new int[i+1];
```

Class Declaration

The syntax for declaring the class is:

class ClassName **{**
  class member declarations
}

Where:

- ClassName is the name of the class, We can use any valid identifier that is not reserved to name the class.
- class member declarations is a sequence of class member declarations.
- The word **class** is a reserved word used to mark the beginning of a class declaration. A class member is either variable or a method.

Example: class Customer{

String name;
int age;
double salary;

Variables

void calc(){ }

void information(){ }

Methods

Class Declaration

}

Object Creation

An object is created from a class. In Java, the **new** keyword is used to create new objects.

syntax :

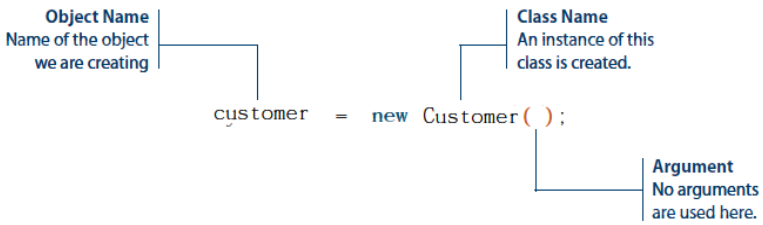
ClassName object-name = **new** ClassName **(**[arguments] **)** ;

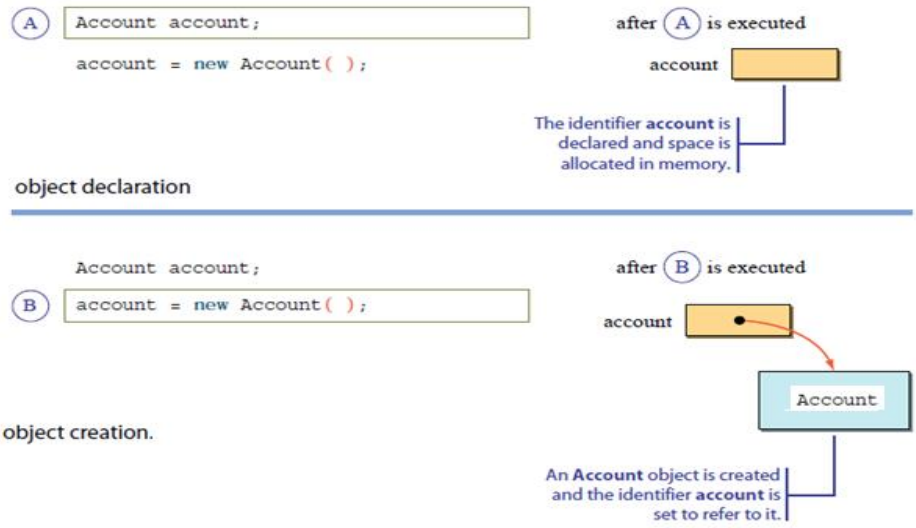
where

- object-name is the name of a declared object, Any valid identifier that is not reserved for other uses can be used as an object name.
- ClassName is the name of the class to which the object belongs,
- arguments is a sequence of values represent initial values of class variables .

Customer customer = **new** Customer**( )**;

o Customer customer;
customer = **new** Customer**( )**;





Consider the following object declaration and two statements of object creation:

Customer customer;

customer = **new** Customer**( )**;

customer = **new** Customer**( )**;

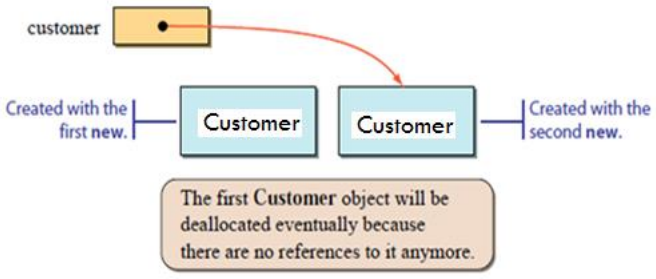
syntax to access an instance variable and method:

ObjectName.variableName;

ObjectName.MethodName();

Q: What do you think will happen? An error?

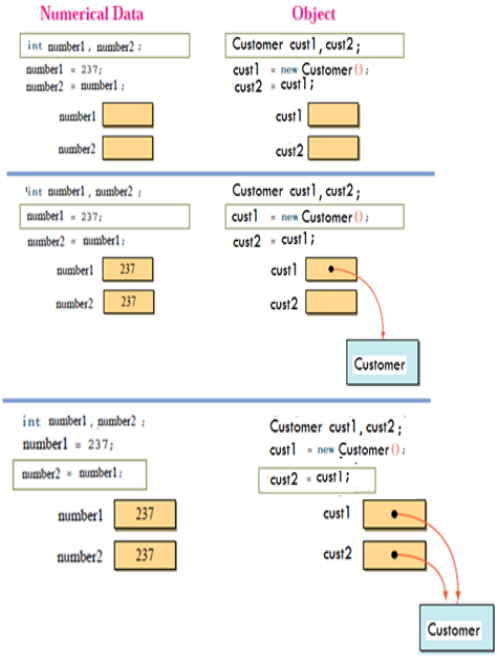
Ans.: No. It is permissible to use the same name to refer to different objects of the same class at different times.

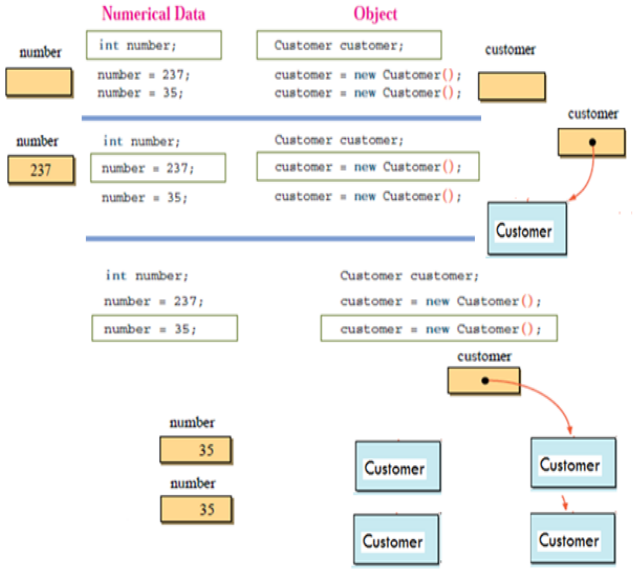


Q: Why objects are called reference data types?

Ans.: Because the contents are addresses that refer to memory locations where the objects are actually stored.

The following examples shown the difference between a variable for numbers and a variable for objects and the effect of assigning the content of one variable to another :





Q1: Create Employee class has:

- five instance variables: empName(string), empId(int), empSalary(double), empBonus(double) and empTotalPay(double)
- two instance methods: printEmployee() to Print the Employee details and empSalary() to calculate the net salary: empBouns+mpSalary and assign it to the variable empTotalPay.

Q2: Create Employee class has:

three instance variables: name(string), age(int) and salary(double).

four instance methods:

1.printEmployee() to Print the Employee details.

2. empSalary(double) to assign the salary to the variable salary.

3. empAge(int) to assign the age of the Employee to the variable age.

4. empName(String) to assign the name of the Employee to the variable name.

Constructors

- A constructor has the same name as its class and is syntactically similar to a method, but constructors have no explicit return type.
- constructor use to give initial values to the instance variables defined by the class.
- You must have noticed, you can create objects without any default constructor defined in your class, there comes the concept of auto-generated default constructor. As whenever object is created, **default constructor** is called. But, If you define **parameterized constructor** in your class, that means you restrict the objects to have that parameters.

There are two types of constructors:

1. Default constructor (noarg constructor)
2. Parameterized constructor

In the **default constructor** there are no parameters.

Syntax: class_name(){ ... }

Example1:

```
class MyClass{
   int x; }

public class Example1 {
  public static void main(String args[]){
  MyClass t1 = new MyClass();
  System.out.println("x= "+t1.x );}
}
```

Example 2:

```
class MyClass {
  int x;
  MyClass() {
      x = 10;}
 }

public class Example1 {
 public static void main(Stringargs[]) {
    MyClass t1 = new MyClass();
    MyClass t2 = new MyClass();
    System.out.println(t1.x + " "+ t2.x); }
}
```

In the **parameterized constructor, a** constructor have parameters. Parameters are added to a constructor in the same way that they are added to a method, just declare them inside the parentheses after the constructor's name.

Example1:

```
class Student{
   int studentNo;
   String studentName;
   Student(String name, int no){
       studentName = name;
       studentNo = no;}
}
```

```
public class Example1 {
  public static void main(String []args){
    Student stud = new Student("Sarah", 123);
    System.out.println("Student's name is :" + stud.studentName + "Student's
                        number is :" + stud.studentNo);}
}
```

Example2:

```
class MyClass {
    int x;
    MyClass(int i ) {
          x = i;}
  }

public class Example2{
  public static void main(String args[]) {
    MyClass t1 = new MyClass( 10 );
    MyClass t2 = new MyClass( 20 );
    System.out.println(t1.x + " " + t2.x);}
}
```

Q: What are difference between constructor and method in java?

- ☒ **Constructor Overloading means** constructors that differ in parameter lists. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

Example:

```
class Student{
    int id;
    String name;
    int age;

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

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

```
 void display(){
   System.out.println(id+" "+name+" "+age);}

 public static void main(String args[]){
   Student s1 = new Student (343,"Sara");
   Student s2 = new Student(292,"Zain", 18);
   s1.display();
   s2.display();}
}
```

- There are two ways to copy the values of one object into another:

1. By constructor
2. By assigning the values of one object into another

Example1:

```
 class Student{
  int id;
  String name;

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

 Student(Student s){
   id = s.id;
   name =s.name;}

 void display(){
    System.out.println(id+" "+name);}

 public static void main(String args[]){
   Student s1 = new Student(292,"Zain");
   Student s2 = new Student(s1);
   s1.display();
   s2.display();}
}
```

Example2:

```
 class Student{
   int id;
   String name;
```

```
  Student(int i, String n){
    id = i;
    name = n; }
 Student(){}
 void display(){
   System.out.println(id+" "+name);}
public static void main(String args[]){
  Student s1 = new Student(292,"Zain");
  Student s2 = new Student();
   s2.id=s1.id;
   s2.name=s1.name;
   s1.display();
   s2.display();}
 }
```

Q:Can constructor perform other tasks instead of initialization?

Ans.: Yes, like object creation, calling method , etc. You can perform any operation in the constructor as you perform in the method.

Variables

A class can contain any of the following variable types:

- Local variables
- Class variables (Static Variables)
- Instance variables (Non-static Variables)

Local variables

- These variables will be declared and initialized within the methods (constructors, or blocks) and destroyed when the method has completed.
- Access modifiers cannot be used for local variables.
- Local variables are visible only within the declared method, constructor, or block.
- There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.

Example:

```
class MyClass{
   void show(int x, int y){
      int z;// local variable
      z=x+y;
      System.out.println("z= "+ z);}
 public static void main(String args[]){
    MyClass t1 = new MyClass();
    t1.show(3,4); }
}
```

Instance variables

- Instance variables are declared in a class(outside a method, constructor or any block).
- These variables are initialized when the class is instantiated.
- Access modifiers can be given for instance variables.
- The instance variables are visible(accessed) for all methods, constructors and block in the class. Normally, it is recommended to make these variables private.
- Instance variables have default values. For numbers, the default value is 0, for Booleans it is false, and for object references it is null.
- Values can be assigned during the declaration or within the constructor.
- Instance variables can be accessed directly by calling the variable name inside the class using the syntax:

ObjectReference.VariableName

Example

```
public class Employee{
    public String name;          // instance variable
    private double salary;
    Employee (String empName){
        name = empName; }

     void setSalary(double empSal){
          salary = empSal;}

    void printEmp(){
        System.out.println("name  : " + name );
        System.out.println("salary :" + salary);
    }

    public static void main(String args[]){
      Employee emp = new Employee("Ali");
    emp.name=" ";
      emp.setSalary(1000);
      emp.printEmp(); }
}
```

Class variables

- Class variables (also known as **static variables)** are declared with the ***static*** keyword in a class.
- Static variables are created when the program starts and destroyed when the program stops.

- Visibility is similar to instance variables. However, most static variables are declared public .
- Default values are same as instance variables. Values can be assigned during the declaration or within the constructor.
- Static variables can be accessed by calling with the class name ***ClassName.VariableName*** *or only* ***VariableName*** as in the following example:

Example

```
public class Employee{
   double salary;
   static String empName;  }
   public static void main(String args[]){
     salary = 1000;
     System.out.println("Name is "+ empName + "    and salary:" +
                                    Employee.salary); }
}
```

Note: If the variables are accessed from an outside class, the variable should be accessed as syntax ClassName.instanceVariable

This keyword

this is a keyword in Java which is used as a reference to the object of the current class. *this* is used only within instance variables and methods or constructors of class. There are many case of usage of this keyword**:**

Case 1: in this case **this** keyword used to distinguish between local variable and instance variable.

Example:

```
class Student{
  int id;
  String name;

Student(int id, String name){
  id = id;
  name = name;}
void display(){
   System.out.println(id+" "+name);  }

public static void main(String args[]){
  Student s1 = new Student(343,"Sara");
  s1.display();  }
}
```

Output:
0 null

In the following example, parameter (formal arguments) and instance variables are same that is why we are using **this** keyword to distinguish between local variable and instance variable :

```
class Student{
  int id;
  String name;

  Student(int id, String name){
    this.id = id;
    this.name = name;}

 void display(){
    System.out.println(id+" "+name);}

 public static void main(String args[]){
   Student s1 = new Student(343,"Sara");
   s1.display();}
 }
```

Note: If local variables(formal arguments) and instance variables are different, there is no need to use this keyword.

Case 2: (explicit constructor invocation(استدعاء)) in this case **this** used to call another constructor in the same class. Call to this() must be the first statement in constructor.

Example1:

```
 class Student{
  int id;
  String name;
  String city;

  Student(){
    System.out.println("default constructor is invoked");}

  Student(int id, String name){
     this();
     this.id = id;
     this.name = name;}

 Student(int id, String name, String city){
   this(id, name);
   this.city=city;}

void display(){
    System.out.println(id+" "+name+" "+city);}

 public static void main(String args[]){
   Student e1 = new Student(343,"Sarah");
   Student e2 = new Student(292,"Zain", "Basrah");
```

Output: ?

```
   e1.display();
   e2.display();}
 }
```

Example 2:

```
 class Student{
  int id;
  String name;

  Student(){
    System.out.println("default constructor is invoked");}

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

Student(int i){
     this ();
     this(int i, "Sara"); }

 void display(){
   System.out.println(id+" "+name);}

 public static void main(String args[]){
   Student e1 = new Student(343,"Sarah");                    Output: ?
   e1.display(); }
 }
```

Variable Arguments(var-args)

Java has a feature that creation of methods that need to take a variable number of arguments. This feature is called *varargs* and it is short for ***var****iable-length* ***arg****ument****s***.

Variable-length arguments could be handled two ways:

1. If the maximum number of arguments was small and known, then you could create overloaded versions of the method, one for each way the method could be called.
2. If the maximum number of arguments was larger or unknowable, in this approach was used in which the arguments were put into an array, and then the array was passed to the method.

or varargs can be specified by three periods (**...**). The varrags allows the method to accept zero or multiple arguments of the same type.

Syntax of varargs:

```
return_type method_name(data_type... variableName)
 { ... }
```

Example1: program to use an array to pass a variable number of arguments to a method.

```
class PassArray {
   static void vaTest(int v[]) {
      System.out.print("Number of args: " + v.length +" Contents: ");
      for (int x : v)
         System.out.print(x + " ");
      System.out.println();}

 public static void main(String args[]){
   int n1[] = { 10 };
   int n2[] = { 1, 2, 3 };
   int n3[] = { };
   vaTest(n1);
   vaTest(n2);
   vaTest(n3);}
 }
```

```
vaTest(new int[]{10 });
vaTest(new int[]{1,2,3});
vaTest(new int[]{ });
```

Example2:

```
class VarArgs {
    static void vaTest(int ... v) {
    System.out.print("Number of args: " + v.length +" Contents: ");
    for(int x : v)
       System.out.print(x + " ");
    System.out.println();}

public static void main(String args[]){
  vaTest(10);
  vaTest(1, 2, 3);
  vaTest();}
}
```

Example3:

```
  public class VarargsExample {
    public static void main(String args[]) {
        printMax(34, 3, 3, 2, 56.5);
        printMax(new double[]{1, 2, 3});}

  public static void printMax( double... numbers) {
      if (numbers.length == 0) System.out.println("No argument passed");
      double result = numbers[0];
      for (int i = 1; i < numbers.length; i++)
         if (numbers[i] > result)    result = numbers[i];
         System.out.println("The max value is " + result);
     }
```

```
  }
```

Example 4:

```
class VarArgs2 {
  static void vaTest(String msg, int ... v) {
    System.out.print(msg + v.length +" Contents: ");
    for(int x : v)
      System.out.print(x + " ");
    System.out.println();
}

public static void main(String args[]){
  vaTest("One vararg: ", 10);
  vaTest("Three varargs: ", 1, 2, 3);
  vaTest("No varargs: ");}
}
```

Note: The variable-length parameter must be one and the last parameter declared by the method.

Examples:

1. The following method declaration is perfectly acceptable:
int doIt(int a, int b, double c, int ... vals)
2. the following declarations is incorrect:
a**.** int doIt(int a, int b, double c, int ... vals, boolean stopFlag)
b. int doIt(int a, int b, double c, int ... vals, double ... morevals)

Method Overloading

If a class have multiple methods by same name but different parameters, it is known as **Method Overloading**.

Q: Why method overloading?

Ans.: If we have to perform only one operation, but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b (int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs. So, we perform method having same name of the methods increases the readability of the program.

Q: **What is the advantage of method overloading?**

Ans.: Method overloading increases the readability of the program.

Different ways to overload the method

There are two ways to overload method :

1. By changing number of arguments
2. By changing the data type

Example1*:* Method overloading by changing the number of arguments

```
class Calculation{
   void sum(int a, int b){
        System.out.println(a+b);}
   void sum(int a, int b, int c){
       System.out.println(a+b+c);}

 public static void main(String args[]){
      Calculation obj=new Calculation();
      obj.sum(10,10,10);
      obj.sum(20,20);}
}
```

Example2: Method overloading by changing data type of argument

```
class Calculation2{
    void sum(int a, int b){
        System.out.println(a+b);}
    void sum(double a, double b){
        System.out.println(a+b);}

  public static void main(String args[]){
     Calculation2 obj=new Calculation2();
     obj.sum(10.5,10.5);
     obj.sum(20,20);}
}
```

Example3*:* Method overloading by changing data type and the number of argument

```
class VarArgs3 {
  static void vaTest(int ... v) {
      System.out.print("vaTest(int ...): " +"Number of args: " + v.length +"
                                        Contents: ");
      for(int x : v)
         System.out.print(x + " ");
          System.out.println();}

  static void vaTest(boolean ... v) {
     System.out.print("vaTest(boolean ...) " +"Number of args: " + v.length +"
                    Contents:");
     for(boolean x : v)
       System.out.print(x + " ");
     System.out.println();}

  static void vaTest(String msg, int ... v) {
      System.out.print("vaTest(String, int ...): " +msg + v.length +" Contents: ");
      for(int x : v)
        System.out.print(x + " ");
    System.out.println();}
```

```
public static void main(String args[]){
  vaTest(1, 2, 3); // OK
  vaTest(true, false, false); // OK
  vaTest(); // Error: Ambiguous!
 }
}
```

Note: In java, method overloading is not possible by changing the return type of the method.

Q: **Why Method Overloading is not possible by changing the return type of method?**

Ans.: In java, method overloading is not possible by changing the return type of the method because there may occur ambiguity. In the following program the statement: int result=obj.sum(20,20); //Here how can java determine which sum() method should be called.

```
class Calculation3{
    int sum(int a, int b){
        System.out.println(a+b);}
    double sum(int a, int b){
      System.out.println(a+b);}

  public static void main(String args[]){
    Calculation3 obj=new Calculation3();
    int result=obj.sum(20,20); //Compile Time Error ...already define
  }
}
```

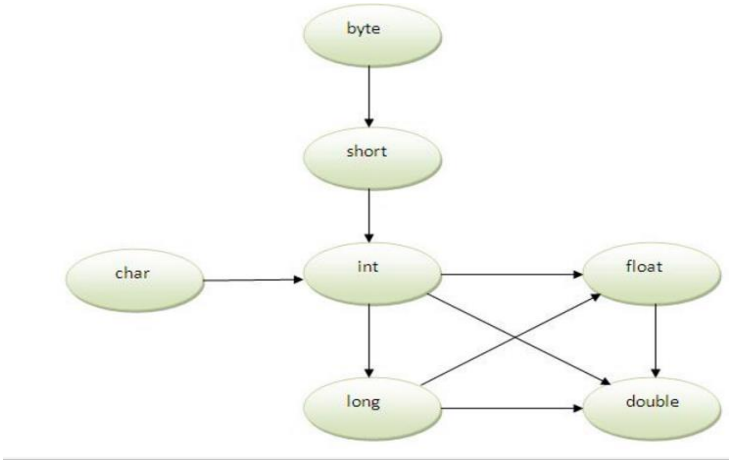
Q: Can we overload main() method?

Ans.: Yes, by method overloading. You can have any number of main methods in a class by method overloading.

```
class Overloading{
  public static void main(int a){
    System.out.println(a); }
    public static void main(String args[]){
      System.out.println("main() method invoked");
      main(10);}
}
```

Method Overloading and Type Promotion(Casting)

One type is promoted to another implicitly if no matching datatype is found. As displayed in the below diagram, byte can be promoted to short, int, long, float or double. The short datatype can be promoted to int,long,float or double. The char datatype can be promoted to int,long,float or double and so on.



Notes:

1. If there are matching type arguments in the method, type promotion is not performed
2. If there are no matching type arguments in the method, and each method promotes similar number of arguments, there will be ambiguity.

Example1:

```
class OverloadingCalculation1{
   void sum(int a, long b){
      System.out.println(a+b);}

   void sum(int a, int b, int c){
       System.out.println(a+b+c);}

   public static void main(String args[]){
      OverloadingCalculation1 obj=new OverloadingCalculation1();
      obj.sum(20,20);//now second int literal will be promoted to long
      obj.sum(20,20,20);}
}
```

Example2:

```
class OverloadingCalculation2{
 void sum(int a, int b){
    System.out.println("int arg method invoked");}

 void sum(long a, long b){
    System.out.println("long arg method invoked");
 }

 public static void main(String args[]){
    OverloadingCalculation2 obj=new OverloadingCalculation2();
    obj.sum(20, 20);}
}
```

Example 3:

```
class OverloadingCalculation3{
   void sum(int a, long b){
       System.out.println("a method invoked");}

   void sum(long a, int b){
       System.out.println("b method invoked");}

  public static void main(String args[]){
      OverloadingCalculation3 obj=new OverloadingCalculation3();
      obj.sum(20, 20);             //ambiguity, Compile Time Error
  }
}
```