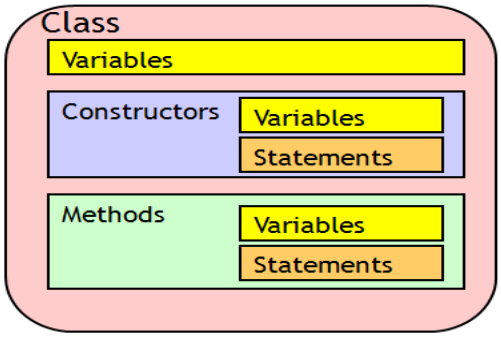
Defining a class: Class is a user defined data type. Once a new data type is defined using a class, it can be used to create objects of that type. A class is a template for an object and object is an instance of a class. Classes create objects and objects use methods to communicate between them.
If we consider the real-world we can find many objects around us, Cars, Dogs, and Humans etc. All these objects have a state and behavior. If we consider a dog then its state is Name, breed, color, and the behavior is barking, wagging and running.
A class is declared by using "class" keyword.



Structure of the program

Syntax:
class classname [**extends** superclassname]
{
[fields declaration;]
[methods declaration;]
}

Rules for defining classes:

- Everything inside the square brackets is optional.
- No semicolon after closing curly bracket.
- Classname and superclassname are any valid java identifiers.
- The keyword extends indicates that the properties of the superclassname class are extended to the classname class. This concept is known as inheritance.
- The data or variables defined within a class are called instance variables.
- Collectively the methods and variables defined within class are called members of the class.

Declaration of fields and methods

Field declaration: A class definition typically includes one or more fields that declare data of various types. The data or variables, defined inside the class are called an instance variable. Declaration of Instance variables is similar as declaration of local variables.

Example:

```
class dog
{
String name;
int age;
}
```

The class dog contains two types of instance variables (name and age). These variables are only declared and therefore no storage space has been created in the memory. Instance variables are also known are **member variables.**

Methods declaration:

Why methods are required? Only data fields in a class has no life. The objects created by such class cannot respond to any messages. Methods are necessary for manipulating the data contained in the class. A class definition typically includes one or more methods, which carry out some action with the data. A method takes some parameters, performs some computations and then optionally returns a value (or object).

Syntax:

```
type methodname (parameter-list)
{
method-body;
}
```

The type specifies the type of data returned by the method. This could be int as well as any class type even be void type, if the method does not return any value. The parameter list is the values passed to the method, listed with type and name.

Examples:

```
class rectangle
{
int length;
int width;
void getdata(int x, int y)
{
        length = x;
        width = y;
}
}
```

We pass two integer values to the method getdata() which is then assigned to the instance variables length and width.

```
class rectangle
{
int length, width;
void getdata(int x, int y)
{       length = x;
        width = y;
}
int rectarea()
{
int area=length * width;
 return(area);
}
}
```

The new method rectarea() computes area of the rectangle and returns the result. Result would be an integer; the return type of the method has been specified as int.

Creating Objects

An object in java is essentially a block of memory that contains space to store all the instance variables.

Steps for creating objects:

1. Declaring an object: Declaring a variable to hold an object is just like declaring a variable to hold a value of primitive type. Declaration does not create new objects. It is simply variable that can refer to an object. For example: Rectangle rect1;

2. Instantiating an object: Creating an object is also referred to as *instantiating* an object. Objects in java are created using the new operator. The new operator creates an object of the specified class and returns a reference to that object.

For example: rect1 = new Rectangle ();

Assigns the object reference to the variable. The variable rect1 is now an object of the rectangle class.

3. Initializing an object: classes provide constructor methods to initialize a new object of that type. A class may provide multiple constructors to perform different kinds of initialization on new objects.

For example:

Rectangle rect1=new Rectangle (10, 20);

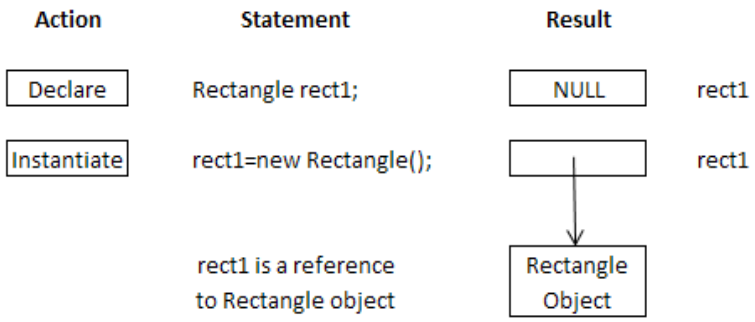


Fig. Creating object reference

Each object has its own copy of instance variables of its class. I.e. any changes to the variables of one object have no effect on the variables of another. It is also possible to create two or more references to the same object.

For example:

Rectangle R1=new Rectangle ();

Rectangle R2=R1;

R1 and R2 refer to same object. The assignment of R1 to R2 did not allocate any memory or copy any part of the original object.

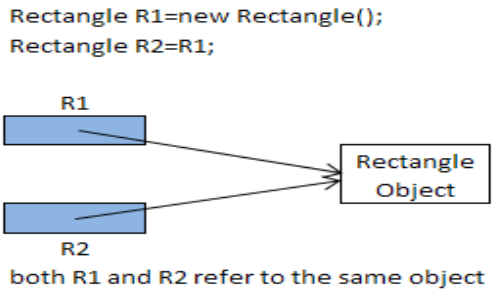


Fig. Assigning one reference of object to another

Differentiate between Class and Object:

Class	Object
A class is a piece of the program's source code that describes a particular type of objects. Object Oriented programmers write class definitions.	An object is called an instance of a class. A program can create and use more than one object (instance) of the same class.
Class is a piece of the program's source code.	Object is an entity in a running program.

Class	Object
Classes written by a programmer.	Object created when the program is running (by the main method or a constructor or another method).
Class specifies the structure (the number and types) of its objects' attributes — the same for all of its objects.	Object holds specific values of attributes; these values can change while the program is running.
Class specifies the possible behaviors of its objects.	Object behaves appropriately when called upon.

Accessing Class members: Object containing its own set of variables; we would assign values to these variables in order to use them in our program. All variables must be assigned values before they are used. Since we are outside the class, we cannot access the instance variables and the methods directly. We can access those using the dot operators.

Syntax:

```
objectname . variablename = value;
objectname . methodname (parameterlist);
```

Example:

```
rect1.length=10;
rect1.width=20;
//creating an object
Rectangle r1 = new Rectangle();
//calling the method using the object
r1.getdata(10,20);
```

Program to calculate area of rectangle using classes and objects

```
class rectangle
{
int length, width;                    //declaration of variables
void getdata(int x, int y)            //definition of method
{
        length = x;
        width = y;
}
int  rectarea()                       //definition of another method
{
int area = length * width;
return (area);
}
}
class Rectarea                        //class with main method
{
public static void main (String args[ ])
{
int area1, area2;
Rectangle rect1 = new Rectangle ();   //creating objects
Rectangle rect2 = new Rectangle ();
rect1.length=15;                      //accessing variables
rect1.width=10;
area1=rect1.length * rect1.width;
rect2.getdata (20, 12);               //accessing methods
```

```
area2=rect2.rectarea ();
System.out.println("Area1= " + area1);
System.out.println("Area2= " + area2);
}
}
```

Output:

Area1 = 150
Area2 = 240

Constructor

Constructors are used to assign initial values to instance variables of the class. A default constructor with no arguments will be called automatically by the Java Virtual Machine (JVM). Constructor is always called by new operator. Constructors are declared just like as we declare methods, except that the constructors don't have any return type, not even void because they return the instance of the class itself. Constructors have the same name as the class itself.

```
class Rectangle
{
int length, width;
 Rectangle (int x, int y)   //constructor method
{
length = x;
 width = y;
}
 int  rectarea()
{
return(length * width); }
}
```

Program to calculate area of rectangle using constructor.

```
class Rectangle
{
int length, width;
Rectangle (int x, int y)    //defining constructor
{
length = x;
width = y;
}
int  rectarea()
{
return(length * width);
}
}
class Rectanglearea       //class with main method
{
public static void main (String args[ ])
{
//calling constructor
 Rectangle rect1 = new Rectangle (15,10);
int area1=rect1.rectarea();
System.out.println("Area1= " + area1);
}
```

}

Points to be remember

- Constructor is a special method that gets invoked "automatically" at the time of object creation.
- Constructor is normally used for initializing objects with default values unless different values are supplied.
- Constructor has the same name as the class name.
- Constructor cannot return values.
- A class can have more than one constructor as long as they have different signature (i.e., different input arguments syntax).

Types of constructors

1. Default Constructor: Default constructor is a constructor which is inserted by Java compiler when no constructor is provided in the class. Every class has a constructor within it. Even abstract classes have a default constructor. By default, Java compiler inserts the code for a zero parameter constructor. The default constructor is the no-argument constructor automatically generated unless you define another constructor. It initializes any uninitialized fields to their default values. The default constructor automatically initializes all class variables to zero. Once you define your constructor, the default constructor is no longer used. The default constructor doesn't serve any purpose other than only fulfilling the requirement of having a valid constructor.

Example:

```
class Number
{
int a,b;
Number()   //non-parameterized constructor
{
a=0;
b=0;
}

Number (int a1, int b1) //parameterized constructor
{
a=a1;
b=b1;
}
}
```

2. Parameterized Constructor: We require a constructor which assigns the different values to variable when different objects are created. So we make some constructor which takes some argument whose value will be assigned to variable. Constructors which have arguments are known as parameterized constructor.

Example

```
Class Studentinfo
{
int rollno, marks1, marks2, total;
Studentinfo(int r, int m1, int m2)          //parameterized constructor
{
rollno=r;
marks1=m1;
marks2=m2;
```

```
}
int total()                          //method to calculate total
{
total=marks1 + marks2;
return total;                        // return value of total where this function is called
}
void display()                       //method to display the result
{
System.out.println("Roll no of student is " + rollno );
System.out.println("marks1 are " + marks1);
System.out.println("marks2 are " + marks2);
System.out.println("Total is " + total);
}
}

class StuResult
{
public static void main(String args[])
{
int total1, total2, grandtotal;
Studentinfo s1=new Studentinfo(1, 50, 80);     //create first object with values
Studentinfo s2=new Studentinfo(2, 40, 60);     //create second object with values
total1=s1.total();                             //calls method total of s1 object and return total
total2=s2.total();                             //calls method total of s2 object and return total
grandtotal=total1 + total2;
s1.display();                                  //call method display
s2.diaplay();
System.out.println("Grand total is" + grandtotal);
 }
 }
```

3. Copy constructor

Like C++, Java also supports copy constructor. But, unlike C++, Java doesn't create a default copy constructor if you don't write your own. A **copy constructor** is a constructor that takes only one parameter which is the same exact type as the class in which the copy constructor is defined. A copy constructor is used to create another object that is a copy of the object that it takes as a parameter. But, the newly created copy is actually *independent* of the original object. It is independent in the sense that the copy is located at a completely different address in memory than the original.

Program for copy constructor:

```
class copyc
{
private double a, b;
 // A normal parametrized constructor
public copyc(double a, double b)
{
this.a=a;
this.b=b;
}

// copy constructor
copyc(copyc c)
```

```
{
System.out.println("Copy constructor called");
a=c.a;
b=c.b;
}

// Overriding the toString of Object class
public String toString()
{
return "(" + a + " " + b + ")";
}
}

public class Testcopyc
{
public static void main(String[] args)
{
copyc c1 = new copyc(10, 15);
// Following involves a copy constructor call
copyc c2 = new copyc(c1);
// Note that following doesn't involve a copy constructor call as
// non-primitive variables are just references.
copyc c3 = c2;
System.out.println(c2);          // toString() of c2 is called here
}
}
```

Output:

```
Copy constructor called
(10.0 + 15.0)
```

Constructor Overloading

Constructor overloading in java allows having more than one constructor inside one Class. Constructor with same name and different arguments then it's called constructor overloading. Overload constructors - define multiple constructors which differ in number and/or types of parameters. Its best practice to have one primary constructor and let overloaded constructor calls that. This way your initialization code will be centralized and easier to test and maintain. Constructor overloading is flexibility which allows you to create object in different way and classic examples are various Collection classes.

Example:

```
import java.lang.*;
import java.io.*;
class Room
{
int length,breadth,height;
Room(int l,int b,int h)          //parameterized constructor
{
length=l;
breadth=b;
height=h;
}
```

```
Room()                          //default constructor
{
length=1;
breadth=1;
height=1;
}

Room(int len)                   //constructor overloading
{
length=breadth=height=len;
}

int volume()
{
return length*breadth*height;
}

}

class OverloadConstructor
{
public static void main(String args[])
{
Room a=new Room(20,30,40);
Room b=new Room();
Room c=new Room(10);

int vol;
vol=a.volume();
System.out.println("Volume of room a is " + vol);

vol=b.volume();
System.out.println("Volume of room b is " + vol);

vol=c.volume();
System.out.println("Volume of room c is " + vol);
}
}
```

Output:

Volume of room a is 24000
Volume of room b is 1
Volume of room c is 1000

Difference between Constructor and Methods

Constructor	Methods
Constructor will be automatically invoked when	Method has to be called explicitly.

an object is created.	
Constructor is a special method of a class but can't be invoked directly by method call.	Methods are member of a class.
Name of constructor is the same as that of class in which it is declared.	Method name need not be the same as the class name.
A constructor is used only once with an object, when you create it, and its code helps initialize its members, and possibly do other things.	Methods can be called if and when you need them, an unlimited number of times on the same object.
There is no return statement in the body of the constructor.	Methods may/may not contains return type.
Constructor is invoked using the new operator.	Methods are invoked using the dot operator.
A constructor can never be abstract or static.	A method is of two types defined (implemented) or undefined (abstract). The method implementation can be further categorized as static or non-static. An abstract method can't be static or final.
Constructor can be specified as public, none (default), protected or private.	Access-specifier public, none (default), protected or private are applicable.
Can't be final, native, or synchronized. (Constructor can take only access specifier)	Can be final, native, static or synchronized.

Methods Overloading

Polymorphism is the capability of an object to respond uniformly to achieve specific behavior to the method calls of the same name but with different implementations.
In java the concept of Polymorphism is achieved in two ways:
1. Method Overloading
2. Method Overriding

In java it is possible to create methods that have the same name, but different parameter lists and different definitions. This is called **method overloading.**

Method overloading is used when objects are required to perform similar tasks but using different input parameters. When we call a method in an object, java matches up the method name first and then the number and type of parameters to decide which one of the definitions to execute. This process is known as **polymorphism.** Method Overloading allows the user to achieve the **compile time polymorphism.** The arguments passed to an overloaded method may differ in type or in number, or both. Overloaded methods may have the same or different return types.

Program of Method Overloading.

```
class Overload
{
void test(int a)
{
System.out.println("a: " + a);
}
void test(int a, int b) {
System.out.println("a and b: " + a + "," + b);
}
double test(double a) {
System.out.println("double a: " + a);
return a*a;
}
```

```
}
class MethodOverloading
{
public static void main(String args[])
{
Overload overload = new Overload();
double result;
overload.test(10);
overload.test(10, 20);
result = overload.test(5.5);
System.out.println("Result : " + result);
}
}
```

Output:

```
a: 10
a and b: 10,20
double a: 5.5
Result : 30.25
```

Method Overriding

In a class hierarchy, when a method in a subclass has the same name and type signature as a method in its superclass, then the method in the subclass is said to ***override*** the method in the superclass. Subclass uses extends keyword to extend the super class. In the example class B is the sub class and class A is the super class. In overriding methods of both subclass and superclass possess same signatures. Overriding is used in modifying the methods of the super class. In overriding return types and constructor parameters of methods should match.

Program of method overriding

```
class A
{
int i;
A(int a, int b)
{
i = a+b;
}

void add()
{
System.out.println("Sum of a and b is: " + i);
}
}

class B extends A
{
int j;
B(int a, int b, int c)
{
super(a, b);
j = a+b+c;
}
```

```
void add()
{
super.add();
System.out.println("Sum of a, b and c is: " + j);
}
}

class MethodOverriding
{
public static void main(String args[])
{
B b = new B(10, 20, 30);
b.add();
}
}
```

Output:

Sum of a and b is: 30
Sum of a, b and c is: 60

Differentiate between method overloading and method overriding

Method Overloading	Method Overriding
It is a process of using several procedures /methods with the same name but having different signatures. The signature changes in number of parameters, type of parameter, and order of parameter.	It is the process of writing functionality for methods with same signature and return type, both in super class and subclass.
Method overloading is used when objects are required to perform similar task but using different input parameters.	In Method overriding object is respond to same method but different behavior when that method is called. Defining Method in subclass has the same name, same parameters and return types as the method in the super class.
A relationship between methods is available in the same class.	A relationship between methods is available in the super class and its sub class.
Inheritance does not blocked by method overloading.	Method overriding blocks the inheritance.
One method can overload unlimited number of times.	Method overriding can be done only once per method in the sub class.
Only method name can be reused.	The complete method signature can be reused.
Method signature should not be same.	Method signature should be same.
Method can have any return type	Method return type should be same as super class method
Overloaded method is bounded by static binding	Overridden methods are subject to dynamic binding.
Method Overloading is an example of compile time polymorphism. The actual method called is decided at compile-time, based on the number and types of arguments.	Method Overriding is an example of run time polymorphism. The JVM does not know which version of method would be called until the type of reference will be passed to the reference variable. It is also called Dynamic Method Dispatch.
// Parent class	//parent class

Method Overloading	Method Overriding

public class PClass
{
public void func(int a)
{
}
}
// Child class
public class CClass
{
public void func(int a, int b)
{
/* overriden method that has the same signature but different arguments */
}
}
	public class PClass
	{
	public void func()
	{
	}
	}
	// Child class
	public class CClass
	{
	public void func()
	{
	// overriden method that has the same signature
	}
	}

Dynamic method Dispatch

- Dynamic method dispatch is the mechanism by which a call to a overridden method is resolved at runtime rather than at compile time
- It forms the basis for runtime polymorphism
- When an overridden method is called through a super class reference
- java determines which method to execute based upon the type of the object being referred at the time the call is made

Example:

```
import java.lang.*;
class A
{
void show()
{
System.out.println("in show of A");
}
}

class B extends  A
{
void show()
{
System.out.println("in show of B");
}
}

class C extends  B
{
void show()
{
System.out.println("in show of C");
}
}
```

```
class DynaMethod
{
public static void main(String args[])
{
A  a = new A();          // object of class A
B  b = new B();          // object of class B
C  c = new C();          // object of class C
A r;                     // obtain a reference of type A
r = a;                   //r refer to object of A
r.show();                // call show() of A
r = b;                   //r refer to object of B
r.show();                // call show() of B
r = c;                   //r refer to object of C
r.show();                // call show() of C
}
}
```

Output:

```
in show of A
in show of B
in show of C
```

Reference of class A stores the reference of class A, class B and class C alternatively. Every call to show() method would result in execution of show() method of that class whose reference is currently stored by a reference of class A.

Final variables, Methods and class

A parameter to a function can be declared with the keyword ***final***. This indicates that the parameter cannot be modified in the function. The final keyword can allow you to make a variable a constant.

Example: public final float PI = 3.14;

final void show() {----------}

Then the final variable can be assigned only once. A variable that is declared as final and not initialized is called a ***blank final variable***. A blank final variable forces the constructors to initialize it. Making a method final ensures that the functionality defined in this method will never be altered in any way. The value of a final variable can never be changed. Methods declared as final cannot be overridden. Final is used to prevent inheritance.

Final methods:

A final method can't be overridden by subclasses. This is used to prevent unexpected behavior from a subclass altering a method that may be crucial to the function or consistency of the class.

```
public class MyClass
{
public void myMethod() {...}
public final void myFinalMethod() {...}
}

public class AnotherClass extends MyClass
{
public void myMethod() {...}              // Ok
public final void myFinalMethod() {...}   // forbidden
}
```

A common misconception is that declaring a class or method final improves efficiency by allowing the compiler to directly insert the method inline wherever it is called. In fact the compiler is unable

to do this because the method is loaded at runtime and might not be the same version as the one that was just compiled. Only the runtime environment and JIT compiler have the information about exactly which classes have been loaded, and are able to make better decisions about when to inline, whether or not the method is final.

Final Classes:
A class that cannot be sub classed is called a ***final class***. This is achieved in java using final keyword. A final class cannot be extended. A final class implicitly has all the methods as final, but not necessarily the data members.
final class Aclass{---------}
final class Bclass extends Someclass {----------}

Use of final with respect to inheritance.
One use of final keyword is to create named/symbolic constants. Two more uses of final can be seen w.r.t inheritance as:

- Prevent overriding of method
- Prevent inheritance

1. Prevent overriding of method: To disallow a method to be overridden final can be used as a modifier at the start of declaration. Methods written as final cannot be overridden.
Example:
```
class test
{
final void disp()
{
System.out.println("In superclass");
}
}
```

2. Final can be used to even disallow the inheritance, to do this a class can be defined with "final" modifier, declaring a class as final declares all its method as final.
Example:
```
final class test
{
void disp()
{
 System.out.println("In superclass");
}
}
Class test 1 extends test // error as class test is final { … }
```

Abstract Class and Abstract Methods

Abstract class:
An abstract class is a class that is declared by using the **abstract** keyword. Abstract classes cannot be instantiated, but they can be extended into sub-classes. Java provides a special type of class called an abstract class. This helps us to organize our classes based on common methods. An abstract class lets you put the common method names in one abstract class without having to write the actual implementation code. Abstract class has at least one method declared as abstract. Java does not allow creating object of abstract class. If any class contains abstract methods then it must implements all the abstract methods of the abstract class.
`Syntax: public abstract class Name {  //code  }`
Program to Calculate area of any type of shape Using abstract class

```
abstract class type
{
abstract void area();
}
class rectangle extends type
{
int len,wid;
rectangle(int a,int c)
{
len=a;
wid=c;
}
void area()
{
System.out.println("area"+(len*wid));
}
}

class square extends type
{
int x;
square(int a)
{
x=a;
}
void area()
{
System.out.println("area="+x*x);
}
}

class mainprog
{
public static void main(String args[])
{
type q=new rectangle(3,7);
type r=new square(7);
q.area();
r.area();
}
}
```

Output:

```
area21
area=49
```

Abstract method:

- An *abstract method* is a method that is declared without an implementation (without braces, and followed by a semicolon), like this: abstract void total(int x, int y);
- If a class includes abstract methods, the class itself *must* be declared abstract.
- The method must be preceded by reserved keywords abstract.
- Method should not have body.

- Syntax: abstract identifier type Name();

Static members and methods

1. Static Members:

When a number of objects are created from the same class, each instance has its own copy of class variables. But this is not the case when it is declared as **static**. **Static** method or a variable is not attached to a particular object, but rather to the class as a whole. They are allocated when the class is loaded. If some objects have some variables which are common then these variables or methods must be declared static. To create a static variable, precede its declaration with keyword static. Static variables need not be called on any object.

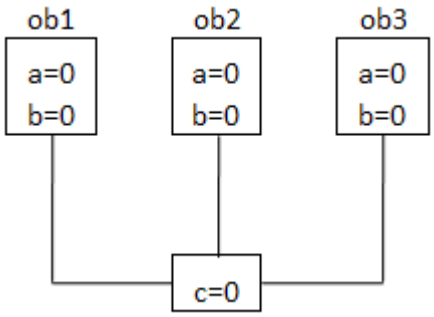
Example:

```
int a;          // normal variable
static int a;   //static variable
```

Program for static variables

```
class s
{
int a,b;
static int c;
}
class statictest
{
public static void main(String args[])
{
s ob1=new s();
s ob2=new s();
s ob3=new s();
}
}
```

In above program variable a and b are declared as int variable whereas the variable c is declared as static variable. Static variable will be common between all the objects.



When we accessing variable as ‘a’ we cannot access it directly without the reference of the object because ‘a’ exist in all three objects. So to access the value of ‘a’ we must link it with the name of the object. But the ‘c’ is static variable and it is common between all the objects so we can access it directly. But when we are accessing it outside the class in which it is declared, we must link it with the name of the class i.e. with S as S.c=20;

2. Static Method:

In java we have two types of methods, **instance methods** and **static methods**. Static methods can't use any instance variables. A static method can be accessed without creating an instance of the

class. Static method can call only other static methods and static variables defined in the class. Static methods can be declared as static similar to declare the variable as static.

Program for static method

```
public class AccessStaticMethod
{
int i;
static int j;
public static void staticMethod()
{
System.out.println("you can access a static method this way");
}
public void nonStaticMethod()
{
i=100;
j=1000;
System.out.println("Don't try to access a non static method");
}
public static void main(String[] args)
{
// i=100;
j=1000;
//  nonStaticMethod();
staticMethod();
}
}
```

Output:

You can access a static method this way.

When we try to call both the method without constructing a object of the class. You will find that only static method can be called this way.

Difference between static and final

Static	Final
Static keyword can be applied to instance variables and methods but not to classes.	Final keyword can be applied to all constructs – variables, methods and classes.
It is not compulsory to initialize the static variable at the time of its declaration.	It is compulsory to initialize the final variable at the time of its declaration.
The static variable can be reinitialized.	The final variable cannot be reinitialized.
Static variable is a global variable shared by all the instances of objects and it has only single copy.	Final variable is a constant variable and it can't be changed.
Static variable can change their values,	Final variables cannot be changed they are constants.

Nesting of Methods

If a method in java calls a method in the same class it is called ***Nesting of methods***. When a method calls the method in the same class dot (.) operator is not needed. A method can call more than one method in the same class. Successive calls can be made (First method can call the second method the second method can call a third method and so on.)

Program for Nesting Method

```
class Sample
{
int x=10,y=20;
void display()
{
System.out.println("Value of X= "+x);
System.out.println("Value of Y= "+y);
}
void add()
{
display(); //Calling method in the same class
System.out.println("X+Y= "+(x+y));
}
}

/* Main class */
class NestingMethod
{
public static void main(String args[])
{
Sample s=new Sample(); //Object of creation
s.add();
}
}
```

Output:

Value of X= 10
Value of Y= 20
X+Y= 30

Command line arguments

When we provided the input at the time of execution, it is known as command line arguments. Command line arguments are parameters that are supplied to the application program at the time of invoking it for execution.
Args is declared as an array of strings. Any arguments provided in the command line are passed to the array args as its elements.

```
class testarg
{
public static void main(String args[])
{
for(int i=0;i<args.length;i++)
{
System.out.println("Args[" + i + "]  is " + args[i] );
}
}
}
```

Steps to run program
Javac testarg
Java testarg C C++ Java

Output

Args[0] is C

Args[1] is C++
Args[2] is Java

This keyword

The keyword **this** is useful when you need to refer to instance of the class from its method. **This** can be used inside any method to refer to the *current* object. That is, **this** is always a reference to the object on which the method was invoked. You can use **this** anywhere a reference to an object of the current class' type is permitted. The keyword helps us to avoid name conflicts. In the program we have declare the same name for instance variable and local variables. Now to avoid the confliction between them we use **this** keyword.

Example:

Without using this	Using this

class th
{ int a;
void assign(int a1)
{ a=a1; }
}
class thcheck
{
public static void main(String args[])
{
th ob1=new th();
ob1=assign(3);
}
}
	class th
	{ int a;
	void assign(int a)
	{ this.a=a; }
	}
	class thcheck
	{
	public static void main(String args[])
	{
	th ob1=new th();
	ob1=assign(3);
	}
	}

In first example object ob1 of class th is created and invoked the assign() method of class th to passed the value of 3 which is stored in local variable a1. The value of a1 i.e. 3 will be stored in the global variable a of object ob1. In second example both the variables have the same variable name. One is local and other is instance variable. When the local variable has the same name as instance variable, the local variable hides the instance variable and in statement a=a, both 'a' will be considered as local variable. In order to differentiate between these two variables, the variable on the left hand side is always followed by this keyword. So this.a is an instance variable which will get value of its local variable.

Program to calculate area of rectangle using this keyword

```
class Rectangle
{
int length,breadth;
void show(int length,int breadth)
{
this.length=length;
this.breadth=breadth;
}
int calculate(){
return(length*breadth);
}
}
public class UseOfThisOperator
{
public static void main(String[] args)
{
Rectangle rectangle=new Rectangle();
```

```
rectangle.show(5,6);
int area = rectangle.calculate();
System.out.println("The area of a Rectangle is :  " + area);
}
}
```

Output: The area of a Rectangle is: 30

Usage of this keyword

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

Varargs: Variable length arguments

Variable argument or varargs in Java allows you to write more flexible methods which can accept as many argument as you need. Before varargs either we use overloaded method or take an array as the method parameter but it was not considered good because it leads to the maintenance problem. If we don't know how many argument we will have to pass in the method, varargs is the better approach. Varargs is a helper syntax and it enables use of variable number of arguments in a method call. In method definition variable aruguments are indicated by elipsis (…) and is called as 'variable arity method' and 'variable arity parameter' in java language specification. While invoking the varargs method we can use any number of arguments of type specified and separated by comma.
Syntax: type … variable Name.
Ellipses stands for variable argument java treats variable argument as an array of same data type. 3 dots is used to denote variable argument in a method and if there are more than one parameter, varargs arguments must be last, as better listed below
Some points which should be taken care when use varargs:

- Ellipse can be used once in method parameter list.
- Ellipse with type must be used in parameter list at the end of the method

Syntax:

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

Example:

```
class VarargsExample
{
static void display(String... values)
{
System.out.println("display method invoked ");
for(String s:values)
{
System.out.println(s);
}
}

public static void main(String args[])
{
display();                          //zero argument
display("hello");                   //one argument
```

```
display("One","Two","Three","Four");    //four arguments
}
}
```

Output:

```
display method invoked
display method invoked
hello
display method invoked
One
Two
Three
Four
```

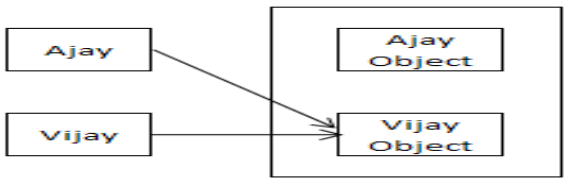
Garbage collection

When the object is not needed any more, the space occupied by such objects can be collected and use for later reallocation. Java performs release of memory occupied by unused objects automatically, it is called ***garbage collection***. JVM runs the garbage collector (gc) when it finds time. Garbage collector will be run periodically but you cannot define when it should be called. We cannot even predict when this method will be executed. It will get executed when the garbage collector, which runs on its own, picks up the object.

Programs that do not de-allocate memory can eventually crash when there is no memory left in the system to allocate. These programs are said to have memory leaks. In Java, Garbage collection happens automatically during the lifetime of a java program, eliminating the need to de-allocate memory and avoiding memory leaks.

Example:

```
Student Ajay= new Student();
Student Vijay= new Student();
Ajay=Vijay;
```



Both Ajay and Vijay points to the same object. Now Ajay Object becomes garbage, It is unreferenced Object. A garbage collector can be invoked explicitly by writing following statement:

```
System.gc();
```

Advantages of Garbage Collection

- Garbage Collection eliminates the need for the programmer to deallocate memory blocks explicitly.
- Garbage collection helps ensure program integrity.
- Garbage collection can also dramatically simplify programs.

Disadvantages of Garbage Collection:

- Garbage collection adds an overhead that can affect program performance.
- Garbage Collection requires extra memory.
- Programmers have less control over the scheduling of CPU time.

Finalize() Methods

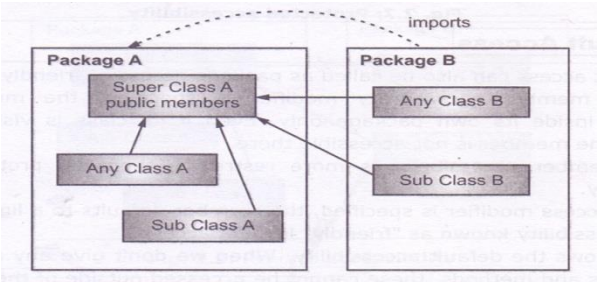
Java runtime is an automatic garbage collecting system. It automatically frees up the memory resources used by the objects. But objects may hold other non object resources such as file

descriptors or window system fonts. The garbage collector cannot free these resources. In order to free these resources we must use a finalizer method. This is similar to destructors in C++. Before an object is garbage collected, the runtime system calls its finalize() method. The intent is for finalize() to release system resources such as open files or open sockets before getting collected. The finalize method has no arguments and no returns value. The garbage collector will always call the object's finalize method before memory is reclaimed. The finalize method is normally declared protected and is not a public service of the class. A finalize method can handle Exceptions (try – catch), but any uncaught Exceptions are ignored.

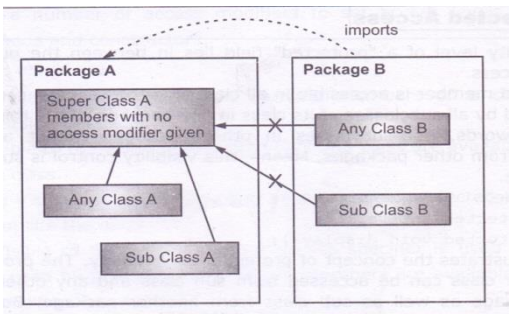
Visibility control: public, private, protected, friendly, private-protected access

Variables and methods of a class are visible everywhere in the program. However it may be necessary in some situations to restrict the access to certain variables and methods from outside the class. For example we may not like the objects of a class directly alter the value of a variable or access a method. We can achieve this in java by applying visibility modifiers to the instance variables and methods. The visibility modifiers are also known as access modifiers. Java provides three types of visibility modifiers: public, private and protected.

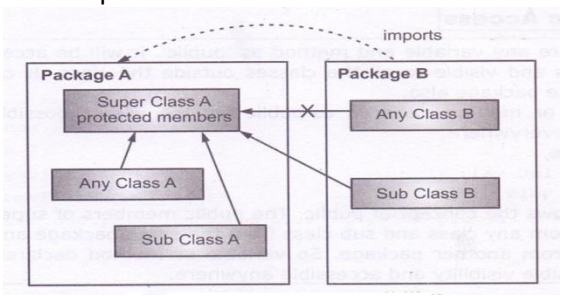
Public access: Public modifier achieves the highest level of accessibility. A class, method, constructor, interface, etc. declared public can be accessed from any other class. Therefore, fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe. The most know public method in Java is the main function. Any variable or methods declared inside the class is visible to the entire class. To make variable and method is visible to all the classes outside this class by declaring the variables or methods as public.
Example: public int number;
Public void sum() {----------}



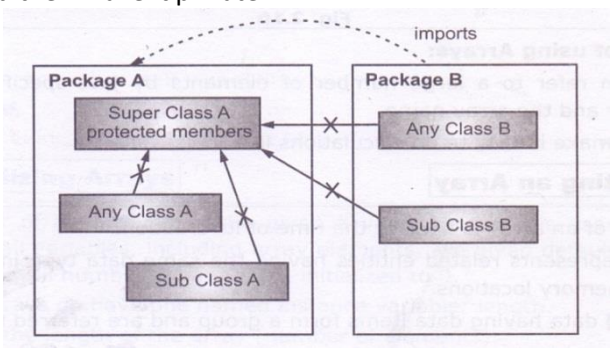
Friendly Access: When no access modifier is specified, the member defaults to a limited version of public accessibility known as “friendly” level of access. The difference between the public access and friendly access is that the public modifier makes fields visible in all classes, regardless of their packages while the friendly access makes fields visible only in the same package, but not in other packages. (A package is a group of related classes stored separately. A package in java is similar to a source file in C)



Protected Access: The visibility level of a "protected" field lies in between the public access and friendly access. That is the protected modifier makes the fields visible not only to all classes and sub classes in the same package but also to sub classes in other packages. Note that non sub classes in other packages cannot access the "protected" members.



Private Access: Private fields enjoy the highest degree of protection. Methods, variables, and constructors that are declared private can only be accessed within the declared class itself. They are accessible only with their own class. They cannot be inherited by sub classes and therefore not accessible in sub classes. A method declared as private behaves like a method declared as final. It prevents the method from being sub classes. Also note that we cannot override a non private method in a sub class and then make it private.



Private-Protected Access:
The visibility level lies between private and protected access. This modifier makes fields visible in all subclasses regardless of what package they are in. These fields are not accessible by other classes in the same package.

Access Levels

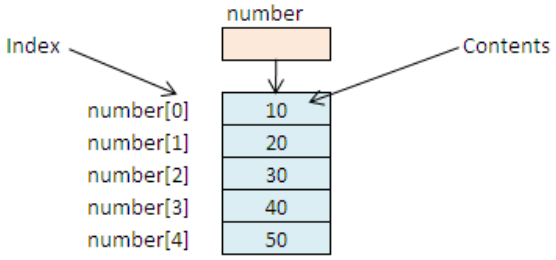
Modifier	Same class	Subclass In Same package	Other classes In same package	Sub class In other classes	Non sub class In other classes
Public	Yes	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	Yes	No
Friendly (default)	Yes	Yes	Yes	No	No
Private Protected	Yes	Yes	No	Yes	No
Private	Yes	No	No	No	No

Array

An array is a collection of homogeneous or contiguous or related data items that share a common name. We can define an array name empname to represent names of employees. A particular value is indicated by writing a number called index or subscript in brackets after the array name. Example: empname[5] represents the names of five employees. The complete set of values is referred to as an array and the individual value are called elements. Arrays can be of any variable type.

1. One dimensional Array:

An array is an object that is used to store a list of values of the same type. It is made out of a continuous block of memory that is divided into a number of "slots". Each slot can be accessed by using its index (or subscript). If there are N slots in an array, the indexes will be 0 through N-1. A list of items can be given one variable name using only one subscript and such a variable is called a **single subscripted variable** or a **one dimensional array.**



Creating an Array

Creation of array involves three steps:

- Declaration of array.
- Creating memory locations.
- Putting values into the memory locations.

Declaration of Array:

Declaration has two components:

- Array's type.
- Array's name.

Array in java may be declared in two ways:

Step 1: type arrayname[];

Step 2: type [] arrayname;

Where type indicates the data type of the contained elements. The square brackets are special symbols indicating that this variable holds an array. Arrayname can be anything you want.

Examples of array declaration:

```
int number[];
float salary[];
int []  marks;
float [] price;
```

Creation of Arrays:

After declaration of array we need to create it in the memory. Java allows us to create array using new operator only.

Syntax:

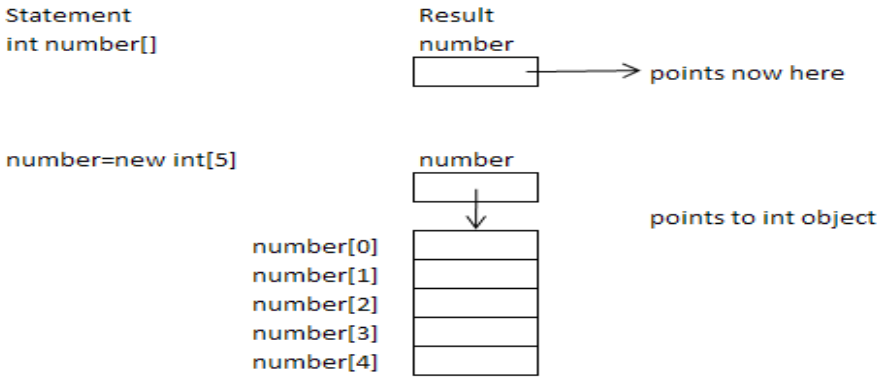
```
arrayname = new type [size];
```

Example:

```
number = new int [5];
salary = new float[5];
```

It is possible to combine the two steps declaration and creation into one statement:

```
int number[] = new int [5];
```



Representation of array in memory

Initialization of Array:

To put values into the array is known as Initialization. Once an array is instantiated, it size cannot be changed.

Syntax: arrayname[subscript] = value;

Example: number[0]=10;

salary[2]= 2500.20;

Java also initializes arrays automatically when they are declared as shown below:

Syntax: type arrayname[]={ list of values};

Example: int number[]={ 10, 20, 30, 40, 50};

Array Length:

'ArrayName.length' gives us the number of elements in the array. The variable length can be directly accessed in a program using the array name and the dot operator.

Example: int size = number.length;

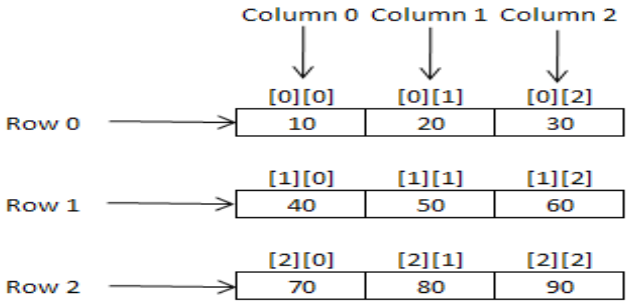
2. Two dimensional Arrays

Two-dimensional arrays are usually represented with a row-column "spreadsheet" style.

Syntax: type array-name [row-size][column-size];

Here type specifies the type of element that will be contained in the array such as int, float or char and row-size specifies the number of rows and column-size specifies the number of column, array-name is the name of table or, matrix.

Example: int v[3][3];



Representation of two dimensional arrays

Creation of two dimensional arrays:

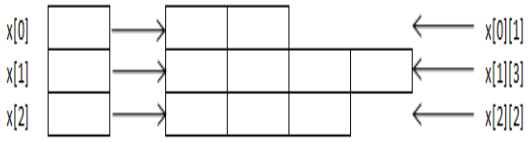
Two way to create an two dimensional array.

Step 1: int number[][];

number = new int [3][3];
Step 2: int array[][] = new int [3][3];

Example of Two dimensional array initialization.
int number[3][3] = { 10, 20, 30, 40, 50, 60, 70, 80, 90};
int marks[][] = {{40, 50, 60},{50, 60, 70}};

Variable size arrays:
Java treats multidimensional array as "arrays of arrays".
int x[][]= new int [3][];
x[0]=new int[2];
x[1]=new int[4];
x[2]=new int[3];
These statements create a two dimensional array as having different lengths for each rows.



String class and String buffer class

The `String` class is commonly used for holding and manipulating strings of text in Java programs. In java strings are class objects and implemented using two classes namely, String and StringBuffer.

String Class:
A java string is an instantiated object of the String class. A java string is not a character array and is not NULL terminated.
Declaration of String:
String stringname;
Stringname = new string ("string");
Example: String name;
name = new String ("vijay");
These two statements may be combined as follows: String name = new String ("vijay");

String class Methods:
The String class defines a number of methods that allow us to accomplish a variety of string manipulation tasks. The list of most commonly used string methods are as follows.

Method Call	Task performed
s2 = s1.toLowerCase;	Converts the string s1 to all lowercase. Example: String s1="COMPUTER"; String s2=s1.toLowerCase(); Output: computer
s2 = s1.toUpperCase;	Converts the string s1 to all uppercase. Example: String s1="computer"; String s2=s1.toUpperCase(); Output: COMPUTER
s2 = s1.replace('x','y');	Replace all appearances of x with y. Example: String s1="computer"; String s2=s1.replace('e','a');

Method Call	Task performed
	Output: computar
s2 = s1.trim();	Remove white spaces at the beginning and end of the string s1. Example: String s1=" computer "; String s2=s1.trim(); Output: computer
s1.equals(s2)	Returns 'true' if s1 is equal to s2. Example: String s1="vijay"; String s2="patil"; s1.equals(s2); If both string are same output is true otherwise false.
s1.equalsIgnoreCase(s2)	Returns 'true' if s1=s2, ignoring the case of characters. Example: String s1="vijay"; String s2="Vijay"; s1.equals(s2); output=true
s1.length()	Gives the length of s1. Example: String s1="computer"; int s2=s1.length(); output: 8
s1.CharAt(n)	Gives nth character of s1. Example: s1.CharAt(2); output: m
s1.compareTo(s2)	Compare two strings and returns int. Returns negative if s1<s2, positive if s1>s2, and zero if s1 is equal s2.
s1.concat(s2)	Concatenates s1 and s2. Example: String s1="vijay"; String s2="patil"; s1.concat(s2); output: vijaypatil
s1.substring(n)	Gives substring starting from nth character. Example: String s1="vijay"; s1.substring(2); output: jay
s1.substring(n,m)	Gives substring starting from nth character up to mth . Example: String s1="vijay"; s1.substring(0,3); Output: vij
String.ValueOf(p)	Creates a string object of the parameter p.
p.toString()	Creates a string representation of the object p.
s1.indexOf('x')	Gives the position of the first occurrence of 'x' in the string s1. Example: String s1="sanjay"; s1.indexOf('a'); Output: 1
s1.indexOf('x',n)	Gives the position of 'x' that occurs after nth position in the string s1. Example: String s1="sanjay"; s1.indexOf('a',2); Output: 4
lastIndexOf(int ch)	Returns the index within this string of the last occurrence of the specified character.

Method Call	Task performed
	Example: String s1=”Welcome to Operating System”; S1.lastIndexOf(‘e’); Output: 25
Str.startsWith(String prefix)	Test if this string starts with the specified prefix Example: String s1=”Welcome to java” S1.startsWith(“Welcome”); Output: true
Str.endsWith(String suffix)	Tests if this string ends with the specified suffix Example: Example: String s1=”Welcome to java” S1.endsWith(“java”); Output: true
String.ValueOf(Variable)	Converts the parameter value to string representation.

StringBuffer class:

The `StringBuffer` class is used to represent characters that can be modified. This is simply used for concatenation or manipulation of the strings. We can insert characters and substring in the middle of a string or append another string to the end. Some methods used in string manipulations are:

Method	Task
s1.setChartAt(n,’x’)	Modifies the nth character to x. StringBuffer s1= new StringBuffer(“vijay”); s1.setCharAt(3,’e’); Output: vijey
s1.append(s2)	Appends the string s2 to s1 at the end. StringBuffer s1 = new StringBuffer("vijay"); StringBuffer s2 = new StringBuffer(" patil"); S1.append(s2); output: vijay patil
s1.insert(n,s2)	Inserts the string s2 at the position n of the string s1. S1.insert(3,s2); Output: vij patil ay
s1.setLength(n)	Sets the length of the string s1 to n. If n<s1.length() s1 is truncated. If n>s1.length() zeros are added to s1.
reverse()	This is the reverse() function used to reverse the string present in string buffer. Example: StringBuffer s1 = new StringBuffer("vijay"); s1.reverse(); Output: yajiv
delete()	This is the delete() function is used to delete multiple character at once from *n* position to *m* position (n and m are will be fixed by you.) in the buffered string. Example: StringBuffer s1 = new StringBuffer("vijay"); s1.delete(2,5); Output: vi
deleteCharAt()	This is the deleteCharAt() function which is used to delete the specific character from the buffered string by mentioning that's position in the string. StringBuffer s1 = new StringBuffer("vijay"); s1.deleteCharAt(2); Output: viay

Difference between string class and string buffer class

String	String buffer
String: It is the most commonly used class which can be used to store string constants. String constants can be any characters from keyboard enclosed in double quotes. Strings can be used when data need not be altered.	**StringBuffer:** It is a peer class of string which can be modified in terms of length & contents.
String is a major class	StringBuffer is a peer class of String
Length is fixed	Length is flexible
Contents of object cannot be modified	Contents of can be modified
Object can be created by assigning String constants enclosed in double quotes.	Objects can be created by calling constructor of StringBuffer class using "new"
Ex:- String s="abc"‖;	Ex:- StringBuffer s=new StringBuffer ("abc");

Vectors

Vector class is in java.util package of java. Vector is dynamic array which can grow automatically according to the requirement. Vector does not require any fix dimension like String array and int array. Vectors are used to store objects that do not have to be homogeneous.
Vector contains many useful methods.

Declaration of vector

Vectors are created like arrays. Vector can be created using constructor of Vector class. Vector cannot directly store simple data types, so objects are created to store in a Vector class. The objects stored in Vector can be retrieved using an index value. Vector class defines three different constructors:

1) Vector list = new Vector();

This constructor creates a default vector, which has an initial size of 10.

2) Vector list = new Vector(int size);

This constructor accepts an argument that equals to the required size, and creates a vector whose initial capacity is specified by size.

3) Vector list = new Vector(int size, int incr);

//create vector with initial size and whenever it need to grows; it grows by value specified by increment capacity. If Vector has to grow and increment is not specified, Vector doubles its capacity when it grows. The vector class supports a number of methods that can be used to manipulate the vectors created.

The methods of Vector class:

Methods	Description
firstElement()	It returns the first element of the vector.
lastElement()	It returns last element.
addElement(item)	Adds the item specified to the list at the end.
elementAt(int index)	Gives the name of the object present at index position.
size()	Gives the number of objects present.
setSize(int newSize)	Sets the size of this vector.
Set(int index, object element)	Replaces the element at the specified position in the vector with the specified element.
capacity()	Returns the current capacity of this vector.
Boolean Contains(object elem)	Tests if the specified object is a component in the vector.
clear()	Removes all of the elements from this vector
remove(int index)	Removes the element at the specified position in this vector.
remove(Object o)	Removes the first occurrence of the specified element in this

Methods	Description
	vector, If the vector does not contain the element, it is unchanged.
removeRange(int fromIndex, int toIndex)	This method removes from this List all of the elements whose index is between fromIndex, inclusive and toIndex, exclusive.
removeElement(item)	Removes the specified item from the list.
removeElementAt(n)	Removes the item stored in the nth position of the list.
removeAllElements()	Removes all the elements in the list.
copyInto(array)	Copies all items from list of array.
insertElementAt(item, n)	Inserts the item at nth position.
boolean contains(Object elem)	Tests if the specified object is a component in this vector.
elements()	It returns an enumeration of the element.
hasMoreElements()	It checks if this enumeration contains more elements or not.
nextElement()	It checks the next element of the enumeration.
int indexOf(Object o)	This method returns the index of the first occurrence of the specified element in this vector, or -1 if this vector does not contain the element.
int indexOf(Object o, int index)	This method returns the index of the first occurrence of the specified element in this vector, searching forwards from index, or returns -1 if the element is not found.
int lastIndexOf(Object o)	This method returns the index of the last occurrence of the specified element in this vector, or -1 if this vector does not contain the element.

Difference between Array and Vector

Array	Vector
An array is a structure that holds multiple values of the same type.	The Vector is similar to array holds multiple objects and like an array; it contains components that can be accessed using an integer index.
An array is a homogeneous data type where it can hold only objects of one data type.	Vectors are heterogeneous. You can have objects of different data types inside a Vector.
After creation, an array is a fixed-length structure.	The size of a Vector can grow or shrink as needed to accommodate adding and removing items after the Vector has been created.
Array can store primitive type data element.	Vector are store non-primitive type data element.
Array is unsynchronized i.e. automatically increase the size when the initialized size will be exceed.	Vector is synchronized i.e. when the size will be exceeding at the time; vector size will increase double of initial size.
Declaration of an array : int arr[] = new int [10];	Declaration of Vector: Vector list = new Vector(3);
Array is the static memory allocation.	Vector is the dynamic memory allocation.
Array allocates the memory for the fixed size ,in array there is wastage of memory.	Vector allocates the memory dynamically means according to the requirement no wastage of memory.
No methods are provided for adding and removing elements.	Vector provides methods for adding and removing elements.
In array wrapper classes are not used.	Wrapper classes are used in vector.
Array is not a class.	Vector is a class.

Wrapper Classes

Vectors cannot handle primitive data types like int, float, long, char and double. Primitive data types may be converted into object types by using the wrapper classes.
The wrapper classes for numeric primitives have the capability to convert any valid number in string format into its corresponding primitive object. For example “10" can be converted into an Integer by using: Integer intVal = new Integer("10");
The wrapper object of a wrapper class can be created in one of two ways: by instantiating the wrapper class with the new operator or by invoking a static method on the wrapper class.

Simple data types and their corresponding wrapper class types are as follows:

Simple type	Wrapper class
boolean	Boolean
char	Character
double	Double
float	Float
int	Integer
long	Long

The wrapper classes have number of unique methods for handling primitive data types and objects, they are listed below:

Converting primitive numbers to object numbers using constructor methods

Constructor calling	Conversion action
Integer IntVal = new Integer(i);	Primitive integer to Integer objects.
Float floatVal = new Float(f);	Primitive float to Float object.
Double DoubleVal = new Double(d);	Primitive double to Double object.
Long LongVal = new Long(l);	Primitive long to Long object.

Note: i, f, d, and l are primitive data values denoting int, float, double and long data types. They may be constants or variables.

Converting object numbers to primitive numbers using typeValue() method

Method calling	Conversion action
int i = IntVal.intValue();	Object to primitive integer.
float f = FloatVal.floatValue();	Object to primitive float.
long l = LongVal.longValue();	Object to primitive long.
double d = DoubleVal.doubleValue();	Object to primitive double.

Converting numbers to Strings using to String() method

Method calling	Conversion action
str = Integer.toString(i);	Primitive integer to string.
str = Float.toString(f);	Primitive float to string.
str = Double.toString(d);	Primitive double to string.
str = Long.toString(l)	Primitive long to string.

Converting String objects to numeric objects using the static method ValueOf()

Method calling	Conversion action
DoubleVal = Double.Valueof(str);	Converts string to Double object.
FloatVal = Float.ValueOf(str);	Converts string to Float object.
IntVal = Integer.Valueof(str);	Converts string to Integer object.
LongVal = Long.ValueOf(str);	Converts string to Long object.

Converting numeric strings to primitive numbers using Parsing methods

Method Calling	Conversion action
int i = Integer.parseInt(str);	Converts string to primitive integer
long i = Long.parseLong(str);	Converts string to primitive long

Autoboxing and Unboxing

The *Autoboxing* and *Unboxing* was released with the Java 5. During assignment, the automatic transformation of primitive type (int, float, double etc.) into their object equivalents or wrapper type (Integer, Float, Double,etc) is known as ***Autoboxing*.** During assignment or calling of constructor, the automatic transformation of wrapper types into their primitive equivalent is known as **Unboxing**.

Example:

```
int i = 0;
i = new Integer(5);   // auto-unboxing
Integer i2 = 5;        // autoboxing
```

Enumerated Types

J2SE 5.0 allows us to use the enumerated type in java using **enum** keyword. Enum type is a type which consists of fixed set of constant fields. This keyword can be used similar to the static final constants.

Example:

```
public class Days
{
public static final int Sunday=0;
public static final int Monday=1;
public static final int Tuesday=2;
public static final int Wednesday=3;
public static final int Thursday=4;
public static final int Friday=5;
public static final int Saturday=6;
}
```

The above code can be rewritten using enumerated are:

```
Public enum Day{Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday}
```

Advantages:

- Compile time type safety.
- We can use the enum keyword in switch statements.

Solved Programs

1. Program to define a class Rectangle having data members length and breadth. Initialize and display data for three objects of class Rectangle and calculate and display area of Rectangle.

```
import java.lang.*;
class Rectangle
{  int length, breadth;              //instance variable
   static int count=1;               //class variable to count no. of objects created
Rectangle()                          // default constructor
{
 length=breadth=0;
}
Rectangle(int x,int y)               // parameterized constructor
{
length=x;
breadth=y;
count++;
}
void display()                       //method to display length and breadth
{
System.out.println("Length of Rectangle="+length);
System.out.println("Breadth of Rectangle="+breadth);
}
int rectarea()
{
int area=length*breadth;
return(area);
}
public static void main(String args[])       //main method
{
Rectangle r1=new Rectangle(10,20);
Rectangle r2=new Rectangle(20,30);
r1.display();
int area1=r1.rectarea();
System.out.println("Area of Rectangle1="+area1);
r2.display();
int area2=r2.rectarea();
System.out.println("Area of Rectangle2="+area2);
Rectangle r3=new Rectangle();
r3.length=30;
r3.breadth=40;
int area3=r3.length*r3.breadth;
System.out.println("Area of Rectangle3="+area3);
System.out.println("Object count="+count);
}
}
```

Output:

```
Length of Rectangle=10
Breadth of Rectangle=20
Area of Rectangle1=200
Length of Rectangle=20
Breadth of Rectangle=30
```

Area of Rectangle2=600
Area of Rectangle3=1200
Object count=3

2. Program to define a class student with four data members such as name, roll no, sub1 and sub2. Define appropriate methods to initialize and display the values of data members also calculate total marks and percentage scored by student

```
class stu
{
String name;
int rollno;
float sub1, sub2;
stu(String n, int r, float s1, float s2)
{
name=n;
rollno=r;
sub1=s1;
sub2=s2;
}

void display()
{
System.out.println("Name="+name);
System.out.println("Rollno="+rollno);
System.out.println("Sub1 marks="+sub1);
System.out.println("Sub2 marks="+sub2);
}

void calculate()
{
float total=sub1+sub2;
System.out.println("total="+total);
float per=(total/200)*100;
System.out.println("Percentage="+per);
}
}

class student
{
public static void main(String args[])
{
stu s=new stu("Vijay",11,90f,80f);
s.display();
s.calculate();
}
}
```

Output:

Name=Vijay
Rollno=11
Sub1 marks=90.0
Sub2 marks=80.0

total=170.0
Percentage=85.0

3. Program to define a class fraction having data member's numerator and denominator. Initialize three objects using different constructors and display it fractional value

```
class frac
{
float num, deno;
frac()
{
num=20;
deno=10;
}

frac(int x,int y)
{
num=x;
deno=y;
}

frac(int a)
{
num=deno=a;
}

void display()
{
float frac=num/deno;
System.out.println("Fraction="+frac);
}
}

class fraction
{
public static void main(String args[])
{
frac f1=new frac();
frac f2=new frac(44,4);
frac f3=new frac(4);
f1.display();
f2.display();
f3.display();
}
}
```

Output:

Fraction=2.0
Fraction=11.0
Fraction=1.0

4. Define a class circle having data members pi and radius. Initialize and display values of data members also calculate area of circle and display it.

```
class abc
{
float pi,radius;
abc(float p, float r)
{
pi=p;
radius=r;
}

void area()
{
float ar=pi*radius*radius;
System.out.println("Area="+ar);
}

void display()
{
System.out.println("Pi="+pi);
System.out.println("Radius="+radius);
}
}

class area
{
public static void main(String args[])
{
abc a=new abc(3.14f,5.0f);
a.display();
a.area();
}
}
```

Output:

```
Pi=3.14
Radius=5.0
Area=78.5
```

5. Program to use of constructor overloading.

```
import java.lang.*;
class sample
{
int a,b;            //data member declaration
sample()            //default constructor initialized to zero
{
a=b=0;
}
sample(int x, int y)        //parameterized constructor
{
```

```
a=x;
b=y;
}
void display()
// method to display the value of variables
{
System.out.println("a="+a+ "b="+b);
}
public static void main(String args[])
// main method
{
sample s1=new sample();
//call default constructor
sample s2=new sample(10,20);
// call parameterized constructor
s1.display();
s2.display();
}
}
```

Output:

```
a=0 b=0
a=10 b=20
```

6. Program to define a class employee to accept emp_id, emp_name, basic_salary from the user and display the gross salary.

```
import java.lang.*;
import java.io.*;
class employee
{
int emp_id;
String emp_name;                    //data member declaration
float basic_salary;
employee(int a, String s, float f) //constructor to initialize data members
{
emp_id=a;
emp_name=s;
basic_salary=f;
}
void display()                      //method to calculate graoss salary
{
float da=basic_salary*15/100;
float hra=basic_salary*10/100;
float gross_salary=basic_salary+da+hra;
System.out.println("Emp_id="+emp_id);
System.out.println("Employee name="+emp_name);
System.out.println("Gross_salary="+gross_salary);
}
public static void main(String args[]) throws IOException
{
```

```
BufferedReader in=new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter Employee id=");
int x=Integer.parseInt(in.readLine());
System.out.println("Enter Employee Name= ");
String n=in.readLine();
System.out.println("Enter Basic Salary=");
float f=Float.parseFloat(in.readLine());
employee e=new employee(x,n,f);
e.display();
}
}
```

Output:

```
Enter Employee id=11
Enter Employee Name= vijay
Enter Basic Salary=15000
Emp_id=11
Employee Name=vijay
Gross_salary=18750.0
```

7. Program to accept principal amount, rate of interest, no. of years from the user and display the simple interest

```
import java.io.*;
class simple
{
private int amount;
private double rate;
private int noyear;
private double interest;

void calculate()
{
DataInputStream d=new DataInputStream(System.in);
try
{
System.out.print("Enter Principle Amount: ");
amount=Integer.parseInt(d.readLine());
System.out.print("Enter Rate of interest: ");
rate=Float.parseFloat(d.readLine());
System.out.print("Enter the Number of Years: ");
noyear=Integer.parseInt(d.readLine());
interest=(amount*noyear*rate)/100;
}

catch(Exception e)
{
System.out.println("inout/Output Error");
}
}

void display()
```

```
{
System.out.println("\nSimple Interest");
System.out.println("Principle Amount: "+amount);
System.out.println("Rate of Interest: "+rate);
System.out.println("Time Period: "+noyear);
System.out.println("Interest: "+interest);
}
}

class intcal
{
public static void main(String args[])
{
simple ob=new simple();
ob.calculate();
ob.display();
}
}
```

Output:

Enter Principle **Output:**
Amount: 1000
Enter Rate of interest: 5
Enter the Number of Years: 1
Simple Interest
Principle Amount: 1000
Rate of Interest: 5.0
Time Period: 1
Interest: 50.0

8. Program to accept a string from the console and count numbers of vowels, constants, digits, tabs and blank spaces in a string.

```
import java.io.*;
class  Stringdemo
{
public static void main(String args[])
{
String s;
int vow=0, con=0, dig=0, tab=0, bs=0;
DataInputStream d=new DataInputStream(System.in);
try
{
System.out.println("Enter the String");
s=d.readLine();
int l=s.length();
for(int i=0; i<l;i++)
{
char ch=s.charAt(i);
if(ch=='a' || ch=='A' || ch=='e' || ch=='E' || ch=='i' || ch=='I' || ch=='o' || ch=='O' || ch=='u' ||
ch=='U')
```

```
{
vow++;
}
if(ch=='~' || ch=='!' || ch=='@' || ch=='#' || ch=='$' || ch=='%' || ch=='^' || ch=='&' || ch=='*' ||
ch=='_' || ch=='+' || ch=='-')
{
con++;
}
if(ch==' ')
{   bs++; }
if (ch>'0' && ch<'9')
{  dig++; }
if(ch=='\t')
{   tab++; } }}

catch(Exception e)
{
System.out.println("I/O Error");
}
System.out.println();
System.out.println("Number of");
System.out.println("Vowels=" +vow);
System.out.println("Constant=" +con);
System.out.println("Digits=" + dig);
System.out.println("Tabs="+tab);
System.out.println("Blank Space="+bs);
}
}
```

Output:

```
Enter the String
java@ programming      1.4.0
Number of
Vowels=5
Constant=1
Digits=2
Tabs=1
Blank Space=0
```

9. Program to accept a string from user and count all occurrences of a particular word in a string.

```
import java.io.DataInputStream;
class word
{
public static void main(String args[])
{
String str=new String();
String search=new String();
int i, len1, len2, index=-1, count=0;
DataInputStream in=new DataInputStream(System.in);

try
```

```
{
System.out.print("Enter String: ");
str=in.readLine();
System.out.print("Enter a word to be searched in string: ");
search=in.readLine();
}

catch(Exception e)
{
System.out.println("I/O Error");
}

len1=str.length();
len2=search.length();
for(i=0;i<=len1-1;i=index+len2)
{
index=str.indexOf(search,i);
if(index!=-1)
count++;
else
break;
}
System.out.println("Total No. of Occurrences of word"+search+"in given string is: " +count);
}
}
```

Output:

```
Enter String:  ajay vijay
Enter a word to be searched in string: ja
Total No. of Occurrences of wordjain given string is: 2
```

10. Program to accept value of apple sales for each day of the week (using array of type float) and then calculate the average sale of the week.

```
import java.io.*;
class apple
{
public static void main (String args[])
{
float app[]=new float[7];
float avg=0;
DataInputStream d=new DataInputStream(System.in);
try
{
for(int i=0;i<7;i++)
{
System.out.print("Enter the sales of apple
on day" +(i+1)+":");
app[i]=Float.parseFloat(d.readLine());
}

for(int i=0;i<7;i++)
```

```
{
avg+=app[i];
}
avg=avg/7;
System.out.println("Average sales of week: "+avg);
}

catch(Exception e)
{
System.out.println("I/O Error");
}
}
}
```

Output:

```
Enter the sales of apple on day1: 20
Enter the sales of apple on day2: 30
Enter the sales of apple on day3: 40
Enter the sales of apple on day4: 50
Enter the sales of apple on day5: 60
Enter the sales of apple on day6: 70
Enter the sales of apple on day7: 80
Average sales of week: 50.0
```

11. Define a class item having data member code and price. Accept data for five object and display data (in tabular format) Using array of objects. Also display the total price of all items.

```
import java.io.*;
class item
{
int code;
int price;
void display()
{
System.out.println(code+"\t"+price);
}
}

class record1
{
public static void main(String args[])
{
int total=0,i=0;
item arr[]=new item[5];
DataInputStream d=new DataInputStream(System.in);
try
{
for(i=0;i<5;i++)
{
arr[i]=new item();
System.out.print("Enter the code: ");
```

```
arr[i].code=Integer.parseInt(d.readLine());
System.out.print("Enter the price: ");
arr[i].price=Integer.parseInt(d.readLine());
total+=arr[i].price;
}

System.out.println("\nCode\tPrice");
for(i=0;i<5;i++)
{
arr[i].display();
}

System.out.println("\nTotal: "+total);
}

catch(Exception e)
{
System.out.println("I/O Error");
}
}
}
```

Output:

```
Enter the code: 11
 Enter the price: 100
Enter the code:  12
Enter the price:  200
Enter the code:  13
Enter the price:  300
Enter the code: 14
 Enter the price: 400
Enter the code:  15
Enter the price:  500
Code    Price
11      100
12      2
```

12. Write a program to accept 10 numbers in an array and display the number in ascending order (use Bubble sort).

```
import java.io.DataInputStream;
class ascending
{
public static void main(String args[])
{
int i,j,t;
int no[]=new int[10];
DataInputStream d=new DataInputStream(System.in);

try
{
for(i=0;i<10;i++)
{
```

```
System.out.print("Enter the numbers: ");
no[i]=Integer.parseInt(d.readLine());
}
System.out.println("Numbers before ascending order: ");
for(i=0;i<10;i++)
{
System.out.print(no[i]+"   ");
}

for(i=0;i<10;i++)
{
for(j=i+1;j<10;j++)
{
if(no[i]>no[j])
{
t=no[i];
no[i]=no[j];
no[j]=t;
}
}
}
}

catch(Exception e)
{
System.out.print("Error");
}
System.out.println("\n"+"Numbers in ascending order: ");
for(i=0;i<10;i++)
{
System.out.print(no[i]+"   ");
}
}
}
```

Output:

```
Enter the numbers:  5
Enter the numbers:  7
Enter the numbers:  6
Enter the numbers:  6
Enter the numbers:  2
Enter the numbers:  9
Enter the numbers:  1
Enter the numbers:  4
Enter the numbers:  3
Enter the numbers:  8
Numbers before ascending order:
5   7   6   6   2   9   1   4   3   8
Numbers in ascending order:
1   2   3   4   5   6   6   7   8   9
```

13. Define a class 'employee' with data members as empid, name and salary. Accept data for 5 objects and print it.

```
import java.lang.*;
import java.io.*;
class employee
{
String empname;
int empid;
int empsalary;
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
void getdata()
{
try
{
System.out.println("Enter Employee ID=");
empid=Integer.parseInt(br.readLine());
System.out.println("Enter Employee Name=");
empname=br.readLine();
System.out.println("Enter Employee Salary=");
empsalary=Integer.parseInt(br.readLine());
}
catch(Exception e)
{
System.out.println("Error");
}
}
void display()
{
System.out.println("Employee ID="+empid);
System.out.println("Employee Name="+empname);
System.out.println("Employee Salary="+empsalary);
}
}

class emp
{
public static void main(String args[])
{
employee e[]=new employee[5];
for(int i=0;i<5;i++)
{
e[i]=new employee();
}
for(int i=0;i<5;i++)
{
e[i].getdata();
}
for(int i=0;i<5;i++)
{
e[i].display();
}
```

```
}
}
```

Output:

```
Employee ID=11
Employee Name=ajay
Employee Salary=5000
Employee ID=12
Employee Name=vijay
Employee Salary=6000
Employee ID=13
Employee Name=sujay
Employee Salary=7000
Employee ID=14
Employee Name=sanjay
Employee Salary=8000
Employee ID=15
Employee Name=santosh
Employee Salary=9000
```

14. Define a class shape having overloaded static member function area() to calculate and display area of square and rectangle.

```
class draw
{
static void area(int a)
{
System.out.println("Area of Square="+a*a);
}

static void area(int l, int b)
{
System.out.println("Area of Rectangle="+l*b);
}
}

class areasr
{
public static void main(String args[])
{
draw.area(6);
draw.area(8,6);
}
}
```

Output:

```
Area of Square=36
Area of Rectangle=48
```

15. Program to implement vector that accepts five items from the command line and store them in a vector and display the objects stored in a vector.

```
import java.util.*;
class vector1
{
public static void main(String args[])
{
Vector v=new Vector();
String names;

for(int i=0;i<args.length;i++)
{
names=args[i];
v.addElement(args[i]);
}
System.out.println("Elements in vector: "+v);
}
}
```

Output:

```
C:\j2sdk1.4.0\bin> javac vector1.java
C:\j2sdk1.4.0\bin> java vector1  c c++ java
Elements in vector: [c, c++, java]
```

16. Write a java Program to implement a vector and perform the following task
1. To add two integer, two float, two characters, two string objects
2. To search a particular object of the vector
3. To display all objects along with their index position.
4. To delete all objects from vector.

```
import java.lang.*;
import java.io.*;
import java.util.*;
class exp5
{
public static void main(String args[]) throws IOException
{
DataInputStream d=new DataInputStream(System.in);
Vector v=new Vector();
int ch;
do
{
System.out.println();
System.out.println("MENU");
System.out.println("1.Creating Vector");
System.out.println("2.To Add 2 int, 2 float, 2 string and 2 character");
System.out.println("3.To dislay all objects along with their index position");
System.out.println("4.To search a paticuar object of the vector");
System.out.println("5.To delete all objects from vector");
System.out.println("6.Exit");

System.out.println("Enter your choice");
```

```
ch=Integer.parseInt(d.readLine());
switch(ch)
{
case 1:
System.out.println("Vector created");
System.out.println(v);
System.out.println("Capacity of vector="+v.capacity());
System.out.println("Size of vector="+v.size());
break;

case 2:
System.out.println("2.To Add 2 int, 2 float, 2 string and 2 character");
System.out.println("Enter two Integer numbers");
int aa=Integer.parseInt(d.readLine());
int bb=Integer.parseInt(d.readLine());
System.out.println("Enter two Float numbers");
float cc=Float.parseFloat(d.readLine());
float dd=Float.parseFloat(d.readLine());
System.out.println("Enter two String Values");
String ee=d.readLine();
String ff=d.readLine();
System.out.println("Enter two Character Values");
String s1=d.readLine();
char gg=s1.charAt(0);
String s2=d.readLine();
char hh=s2.charAt(0);

v.addElement(new Integer(aa));
v.addElement(new Integer(bb));
v.addElement(new Float(cc));
v.addElement(new Float(dd));
v.addElement(ee);
v.addElement(ff);
v.addElement(new Character(gg));
v.addElement(new Character(hh));
System.out.println("your vector is");
System.out.println(v);
System.out.println("Capacity of vector="+v.capacity());
System.out.println("Size of vector="+v.size());
break;

case 3:
System.out.println("Display vector with inde Position");
for(int i=0;i<v.size();i++)
{
System.out.println("v["+i+"]="+v.elementAt(i));
}
break;

case 4:
System.out.println("To search particular object in vector");
```

```
System.out.println(v);
System.out.println("enter object to search");
int ss=Integer.parseInt(d.readLine());
boolean b=v.contains(ss);
if(b==true)
System.out.println("element is present in vector");
else
System.out.println("element is not present in vector");
break;

case 5:
System.out.println("Original Vector is");
System.out.println(v);
v.removeAllElements();
System.out.println("Vector after delete all element");
System.out.println(v);
break;
}
}while(ch!=6);
}
}
```

17. Write a program to accept as many integers as user wants in a vector. Delete a specific element and display remaining list. OR Write a program to implement vector class and its methods for adding and removing elements.

```
import java.io.*;
import java.util.*;
class vectorlist
{
public static void main(String args[]) throws IOException
{
Vector v=new Vector();
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
System.out.println("How many number you wants to enter in vector");
int n=Integer.parseInt(br.readLine());
for(int i=0;i<n;i++)
{
System.out.println("Enter Integer number=");
int no=Integer.parseInt(br.readLine());
v.addElement(new Integer(no));
}
System.out.println("Initial vector Size="+v.size());
System.out.println("Vector elements before remove element=");
for(int index=0;index<v.size();index++)
System.out.println(v.get(index));
System.out.println("Enter index of element to be deleted=");
int index1=Integer.parseInt(br.readLine());
v.removeElementAt(index1);
System.out.println("vector size after deleting element="+v.size());
System.out.println("vector after removing element=");
```

```
for(int index=0;index<v.size();index++)
System.out.println(v.get(index));
}
}
```

Output:

```
How many number you wants to enter in vector 5
Enter Integer number= 10
Enter Integer number= 20
Enter Integer number= 30
Enter Integer number= 40
Enter Integer number= 50
Initial vector Size=5
Vector elements before remove element=
10
20
30
40
50
Enter index of element to be deleted= 2
vector size after deleting element=4
vector after removing element=
10
20
40
50
```

19. Program to define class student with roll no. and name. Derive class attendance with member as present days. Write method to calculate and print average attendance assuming today days as 180.

```
import java.lang.*;
import java.io.*;
class student
{
int rollno;
String name;
student(int r, String n)
{
rollno=r;
name=n;
}
void display()
{
System.out.println("Roll No="+rollno);
System.out.println("Name="+name);
}
}

class att extends student
{
double presentday;
att(int r, String n, int p)
```

```
{
super(r,n);
presentday=p;
}
void calculate()
{
double avg=(100*presentday)/180;
display();
System.out.println("Present days="+presentday);
System.out.println("Average Attendance="+avg);
}
}

class stuatt
{
public static void main(String args[])
{
att s1=new att(11,"vijay",150);
s1.calculate();
}
}
```

Output:

```
Roll No=11
Name=vijay
Present days=150.0
Average Attendance=83.33333333333333
```

20. Program to convert number into binary, Octal and hexadecimal number.

```
class numberconversion
{
public static void main(String args[])
{
int num=10;
String b=Integer.toBinaryString(num);
System.out.println("Binary number of"+num+" is="+b);
String o=Integer.toOctalString(num);
System.out.println("Octal number of"+num+" is="+o);
String h=Integer.toHexString(num);
System.out.println("Hexadecimal number of"+num+" is="+h);
}
}
```

Output:

```
Binary number of10 is=1010
Octal number of10 is=12
Hexadecimal number of10 is=a
```

21. Program to perform addition of two matrixes.

```
class MatrixSum
{
```

```
 public static void main(String[] args)
 {
 int array[][]= {{1,2,3},{4,5,6}};
 int array1[][]= {{2,3,4},{5,6,7}};
 System.out.println("Number of Row= " + array.length);
 System.out.println("Number of Column= " + array[1].length);
 int l= array.length;
 System.out.println("Matrix 1 : ");
 for(int i = 0; i < l; i++)   {
 for(int j = 0; j <= l; j++) {
 System.out.print(" "+ array[i][j]);
 }
 System.out.println();
 }
 int m= array1.length;
 System.out.println("Matrix 2 : ");
 for(int i = 0; i < m; i++)   {
 for(int j = 0; j <= m; j++) {
 System.out.print(" "+array1[i][j]);
 }
 System.out.println();
 }
 System.out.println("Addition of both matrix : ");
 for(int i = 0; i < m; i++)   {
 for(int j = 0; j <= m; j++) {
 System.out.print(" "+(array[i][j]+array1[i][j]));
 }
 System.out.println();
 }
}
}
```

Output:

```
Number of Row= 2
Number of Column= 3
Matrix 1 :
 1 2 3
 4 5 6
Matrix 2 :
 2 3 4
 5 6 7
Addition of both matrix :
 3 5 7
 9 11 13
```

22. Program to search element into array (Linear Search).

```
import java.io.*;
import java.lang.*;
class ArrayS
{
int a[], i, j, m, n, x, y, item;
int count = 0;
void search1()
{
try
{
DataInputStream dts=new DataInputStream(System.in);
System.out.println("Enter size of Array A:");
m = Integer.parseInt(dts.readLine());
a = new int[m];
System.out.println("Enter elements of array A:");
for(i=0;i<m;i++)
a[i]=Integer.parseInt(dts.readLine());
System.out.println("Enter element you want to search:");
item=Integer.parseInt(dts.readLine());
}
catch(Exception s)
{
System.out.println(s.getMessage());
}
for(i=0;i<m;i++)
{
if(a[i] == item)
{
System.out.println("Item is found a location"+ i);
count++;
break;
}
}
if(count == 0)
System.out.println("Item not found. Try again..." );
}
}
class Search
{
public static void main(String arg[])
{
ArrayS a1 = new ArrayS();
a1.search1();
}
}
```

Output:

```
Enter size of Array A: 5
Enter elements of array A:
10
```

```
20
30
40
50
Enter element you want to search: 30
Item is found a location 2
```

23. Program to find sum, average, minimum and maximum of N numbers using input from keyboard.

```
import java.io.*;
class arrayoperation
{
public static void main(String args[])throws IOException
{
int k,sum=0,i,count=0,max,min,n;
int arr[] = new int[10];
float avg;
System.out.println("Enter how many numbers you want to enter : ");
BufferedReader br= new BufferedReader(new InputStreamReader(System.in));
String str;
str=br.readLine();
k = Integer.parseInt(str);

for(i=0;i<k;i++)
{
System.out.println("Enter the number : ");
BufferedReader br1= new BufferedReader(new InputStreamReader(System.in));
str=br1.readLine();
arr[i] = Integer.parseInt(str);
}
max = arr[0];
min = arr[0];
while(count < k)
{
sum += arr[count];
if(min > arr[count])
min = arr[count];
if (max < arr[count])
max = arr[count];
count += 1;
}
avg = sum/k;
System.out.println("The sum is " +sum);
System.out.println("The average is " +avg);
System.out.println("Maximum is " +max);
System.out.println("Minimum is " +min);
}
}
```

Output:

```
Enter how many numbers you want to enter: 5
Enter the number:  10
```

```
Enter the number:  20
Enter the number:  30
Enter the number:  40
Enter the number:  50
The sum is 150
The average is 30.0
Maximum is 50
Minimum is 10
```

24. Write a program to check whether the given string is palindrome or not

```
import java.lang.*;
import java.io.*;
import java.util.*;
class palindrome
{
public static void main(String arg[ ]) throws IOException
{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter String:");
String word=br.readLine( );
int len=word.length( )-1;
int l=0;
int flag=1;
int r=len;
while(l<=r)
{
if(word.charAt(l)==word.charAt(r))
{
l++;
r--;
}
else
{
flag=0;
break;
}
}
if(flag==1)
{
System.out.println("Palindrome");
}
else
{
System.out.println("Not palindrome");
}
}
}
```

Output:

```
Enter String: madam
Palindrome
Enter String: madan
```

Not palindrome

25. Write a program to accept a password from the user and authenticate the user if password is correct.

```
import java.io.*;
class test
{
public static void main(String a[]) throws Exception
{
String S1="Admin";
String S2="12345";
String username, userpwd;
try
{
BufferedReader b = new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter the user name");
username=b.readLine();
System.out.println("Enter the password");
userpwd= b.readLine();
if(S1.equals(username) && S2.equals(userpwd))
{
System.out.println("Welcome!! You are authenticated");
}
else
{
System.out.println("You have entered wrong information!!");
}
}
catch(Exception e)
{
System.out.println("I/O Error");
}
}
}
```

Output:

```
Enter the user name
Admin
Enter the password
12345
Welcome!! You are authenticated
Enter the user name
Admin
Enter the password
123
You have entered wrong information!!
```

26. Write a program to accept first name, middle name and surname in three different strings and then concatenate the three strings to make full name.

```
import java.io.*;
class StringConcat
```

```
{
public static void main(String[] args) throws IOException
{
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
String str1,str2,str3,namestr="";
System.out.println("Enter first name:");
str1=br.readLine();
System.out.println("Enter middle name:");
str2=br.readLine();
System.out.println("Enter last name:");
str3=br.readLine();
namestr = namestr.concat(str1);
namestr = namestr + " ";
namestr = namestr.concat(str2);
namestr = namestr + " ";
namestr = namestr.concat(str3);
System.out.println("Your FULL NAME is : " + namestr);
}
}
```

Output:

Enter first name: Vijay
Enter middle name: Dinanath
Enter last name: Chavan
Your FULL NAME is: Vijay Dinanath Chavan

27. Program to accept two numbers as command line arguments and print the addition of two numbers.

```
class commandaddition
{
public static void main(String args[])
{
int num1 = Integer.parseInt(args[0]); //take argument as command line
int num2 = Integer.parseInt(args[0]); //take argument as command line
int Addition = num1 + num2;
System.out.println("Addition of number is : "+Addition);
}
}
```

Output:

Javac commandaddition
Java commandaddition 10 20
Addition of number is: 30

28. Define a class Mobile with data members 'company_name' and 'screen_size'. Initialize and display values of data members for five mobiles using array of objects.

```
import java.io.*;
class mobile
{
String company_name;
int screensize;
```

```
DataInputStream d=new DataInputStream(System.in);

void getdata() throws IOException
{
System.out.println("Enter the Company Name:");
company_name=d.readLine();
System.out.println("Enter the Screensize:");
screensize=Integer.parseInt(d.readLine());
}

void display()
{
System.out.println("Company Name: "+company_name);
System.out.println("Screensize: "+screensize);
}

public static void main(String args[]) throws IOException
{
mobile m[]=new mobile[5];
for(int i=0;i<=4;i++)
{
m[i]=new mobile();
}

for(int i=0;i<=4;i++)
{
m[i].getdata();
}
for(int i=0;i<=4;i++)
{
m[i].display();
}
}
}
```

Output:

```
Company Name: Samsung
Screensize: 5
Company Name: Lenovo
Screensize: 6
Company Name: LG
Screensize: 6
Company Name: Nokia
Screensize: 5
Company Name: Sony
Screensize: 6
```

29. Define a class Cube having data members length, breadth and height. Initialize three objects using different constructors and display its volume.

```
import java.io.*;
class cube
{
```

```
int length,breadth,height,volume;
cube()
{
length=5;
breadth=4;
height=6;
}

cube(int a)
{
length=breadth=height=a;
}

cube(int a, int b, int c)
{
length=a;
breadth=b;
height=c;
}

void display()
{
volume=length*breadth*height;
System.out.println("Length of cube= "+length);
System.out.println("Breadth of cube= "+breadth);
System.out.println("height of cube= "+height);
System.out.println("Volume of cube= "+volume);
}

public static void main (String args[])
{
cube c1= new cube();
cube c2= new cube(3);
cube c3= new cube(2,3,4);
c1.display();
c2.display();
c3.display();
}
}
```

Output:

```
Length of cube= 5
Breadth of cube= 4
height of cube= 6
Volume of cube= 120
Length of cube= 3
Breadth of cube= 3
height of cube= 3
Volume of cube= 27
Length of cube= 2
Breadth of cube= 3
height of cube= 4
```

Volume of cube= 24

30. Write a java program to find all substrings of a string and print them. For example substring of "fun" are "f", "fu", "fun", "u", "un", and "n".

```
class substring
{
public static void main(String args[])
{
String s1="comp";
for(int i=0;i<=s1.length();i++)
{
for(int j=i;j<=s1.length();j++)
{
System.out.println(s1.substring(i,j));
}
}
}
}
```

Output:

```
c
co
com
comp
o
om
omp
m
mp
p
```

31. Implement a program to accomplish the following task using String and StringBuffer class
1. Compare two strings
2. Reverse of string
3. To search last position of a substring
4. To search a word inside a string

```
import java.lang.*;
import java.io.*;
class exp4
{
public static void main(String args[]) throws IOException
{
DataInputStream d=new DataInputStream(System.in);
int ch;
do
{
System.out.println();
System.out.println("MENU");
System.out.println("1.Compare two strings");
System.out.println("2.Reverse of String");
System.out.println("3.To search last position of a substring");
```

```
System.out.println("4.To search word inside a string");
System.out.println("5.Exit");
System.out.println("Enter your choice");
ch=Integer.parseInt(d.readLine());
switch(ch)
{
case 1:
System.out.println("Compare two strings");
String str1=new String();
String str2=new String();
System.out.println("Enter first string");
str1=d.readLine();
System.out.println("Enter second string");
str2=d.readLine();
if(str1.equals(str2))
{
System.out.println("Both strings are equals");
}
else
{
System.out.println("Both strings are not equals");
}
break;

case 2:
System.out.println("Reverse of String");
System.out.println("Enter String");
String s1=new String();
s1=d.readLine();
StringBuffer s2=new StringBuffer(s1);
s2.reverse();
System.out.println("Reverse String="+s2);
break;

case 3:
System.out.println("Last Position of substring");
System.out.println("Enter String");
String str=new String();
str=d.readLine();
String s=new String();
System.out.println("Enter string to be search");
s=d.readLine();
System.out.println("Last position of "+s+" ="+str.lastIndexOf(s));
break;

case 4:
System.out.println("To find word within string");
String fs=new String();
String ss=new String();
System.out.println("Enter String");
fs=d.readLine();
```

```
System.out.println("Searching word");
ss=d.readLine();

int intIndex=fs.indexOf(ss);
if(intIndex == - 1)
{
System.out.println("word not found");
}
else
{
System.out.println("Found word at index "+intIndex);
}
break;
}
}while(ch!=5);
}
}
```