Unit 3
Chapter 5

INHERITANCE: EXTENDING A CLASS

Reusability is yet another aspect of OOP paradigm. It is always nice if we could reuse something that already exists rather than creating the same all over again. Java supports this concept. Java classes can be reused in several ways. This is basically done by creating new classes, reusing the properties of existing ones. The mechanism of deriving a new class from an old one is called inheritance. The old class is known as the base class or super class or parent class and the new one is called the subclass or derived class or child class.

The inheritance allows subclasses to inherit all the variables and methods of their parent classes. Inheritance may take different forms:

- Single inheritance (only one superclass)
- Multiple inheritance (several superclasses)
- Hierarchical inheritance (one super class, manysubclasses)
- Multilevel inheritance (Derived from a derivedclass)

Java does not directly implement multiple inheritances. However, this concept is implemented using a secondary inheritance path in the form ofinterfaces.

Defining a Subclass

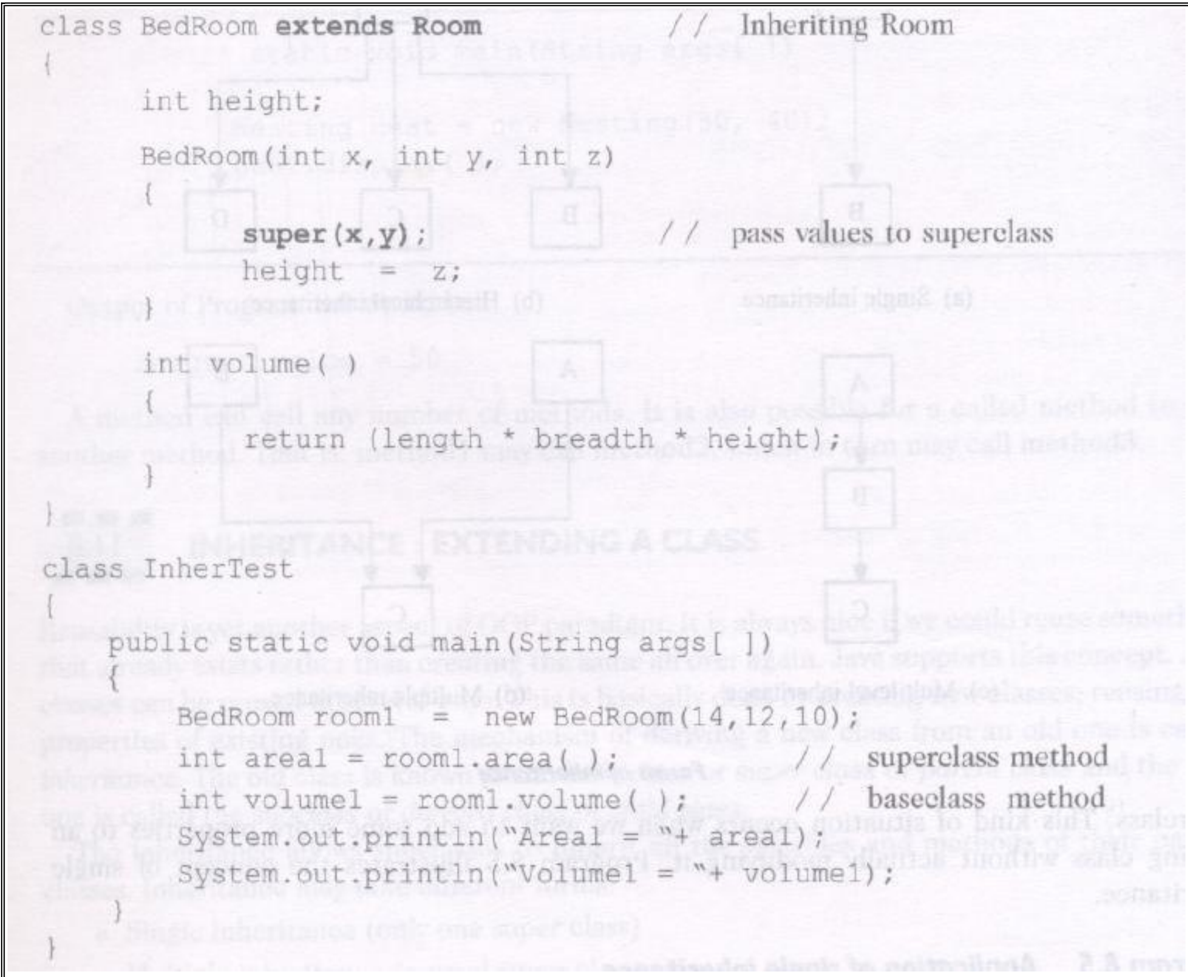
A subclass is defined as follows:

```
class  subclassname extends superclassname
{
     variables declaration ;
     methods declaration ;
}
```

The keyword extends signifies that the properties of the superclassname are extended to the subclass name. The subclass will now contain its own variables and methods as well those of the superclass. This kind of situation occurs when we want to add some more properties to an existing class without actually modifying it.

Program given below shows Application of single inheritance

```
class Room
{
      int length;
      int breadth;
      Room(int x, int y)
      {
         length    = x;
         breadth   = y;
      }

      int area( )
      {
         return (length * breadth);
      }
}
```



```
class BedRoom extends Room            //  Inheriting Room
{
      int height;
      BedRoom(int x, int y, int z)
      {
            super(x,y);                //  pass values to superclass
            height  =  z;
      }
      int volume( )
      {
            return (length * breadth * height);
      }
}
class InherTest
{
     public static void main(String args[ ])
     {
          BedRoom room1  =  new BedRoom(14,12,10);
          int area1 = room1.area( );          //  superclass method
          int volume1 = room1.volume( );      //  baseclass  method
          System.out.println("Area1 = "+ area1);
          System.out.println("Volume1 = "+ volume1);
     }
}
```

The output of the above Program is:

```
Area1 = 168
Volume1 = 1680
```

The program defines a class Room and extends it to another class BedRoom. Note that the class BedRoom defines its own data members and methods. The subclass BedRoom now includes three instance variables, namely, length, breadth and height and two methods, area and volume.

The constructor in the derived class uses the super keyword to pass values that are required by the base constructor. The statement

BedRoom room1 = new BedRoom (14, 12, 10);

calls first the BedRoom constructor method, which in turn calls the Room constructor method by using the superkeyword.

Finally, the object rooml of the subclass BedRoom calls the method area defined in the super class as well as the method volume defined in the subclass itself.

5.2 SubclassConstructor

A subclass constructor is used to construct the instance variables of both the subclass and the superclass. The subclass constructor uses the keyword super to invoke the constructor method of the superclass.

The keyword super is used subject to the following conditions.

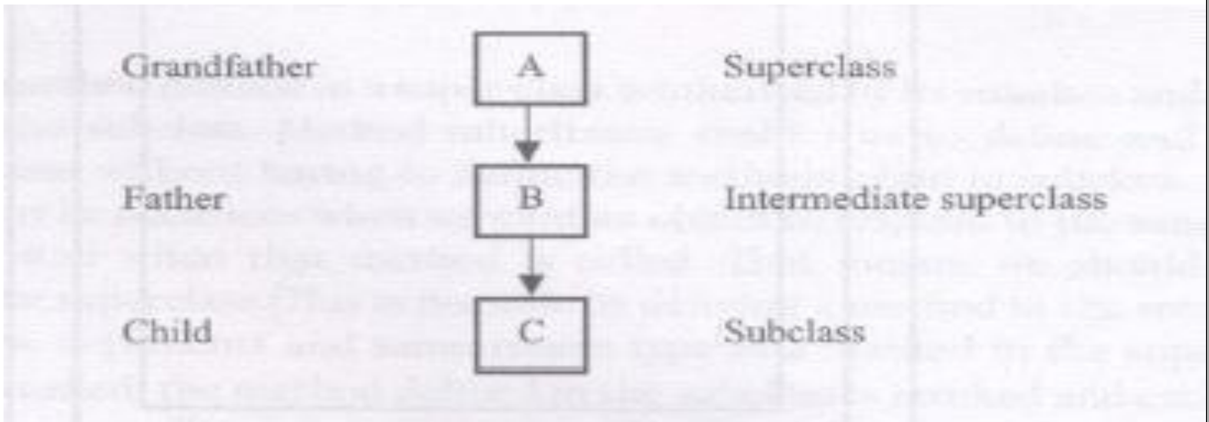
- super may only be used within a subclass constructormethod
- The call to superclass constructor must appear as the first statement within the subclass constructor
- The parameters in the super call must match the order and type of the instance variable declared in thesuperclass.

Above Program illustrated the use of super () method for passing parameters to a superclass.

5.3 Multilevel Inheritance

A common requirement in object-oriented programming is the use of a derived class as a super class. Java supports this concept and uses it extensively in building its class library. This concept allows us to build a chain of classes as shown in bellow.

Fig shows multilevel inheritance.



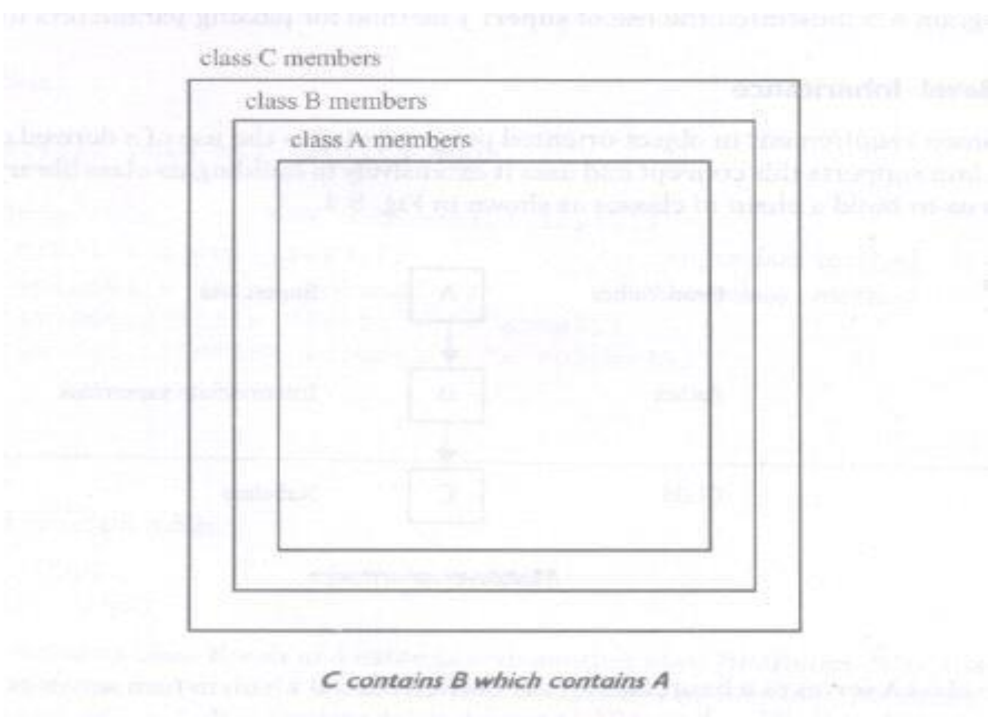
Multilevel inheritance

The class A serves as a base class for the derived class B which in turn serves as a base class for the derived class C. The chain ABC is known *as inheritance path.*

A derived class with multilevel base classes is declared as follows.

```
class A
 {
    ……………………
    ……………………
 }
    Class B extendsA       / / Firstlevel
     {
       ………….. . . . . ……
       …… .. . ……. .........
      }
    Class C extends B / / Second level
     {
       ………………………..
       ………………………..
      }
```

This process may be extended to any number of levels. The class C can inherit the members of both A and B as shown below.

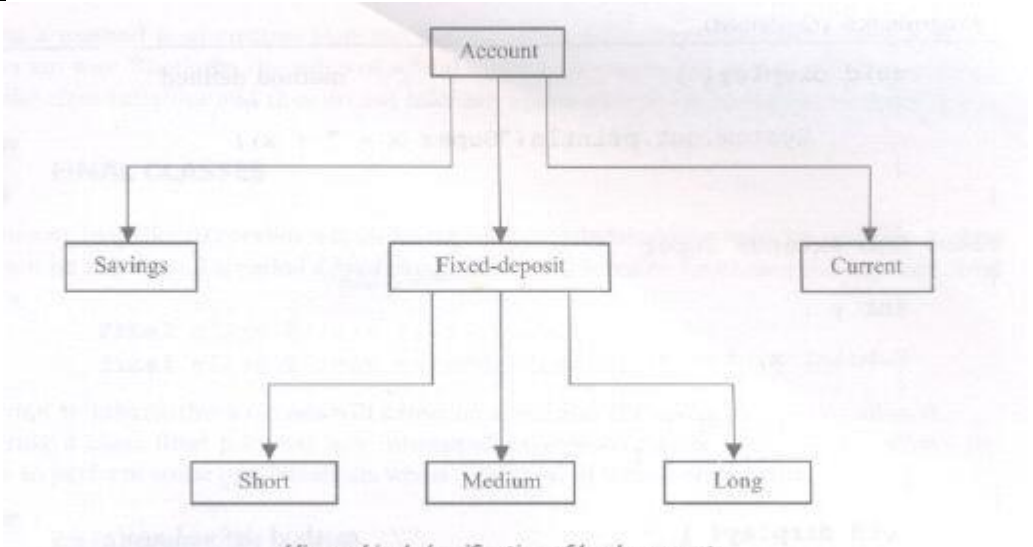


C contains B which contains A

5.4

Hierarchical Inheritance

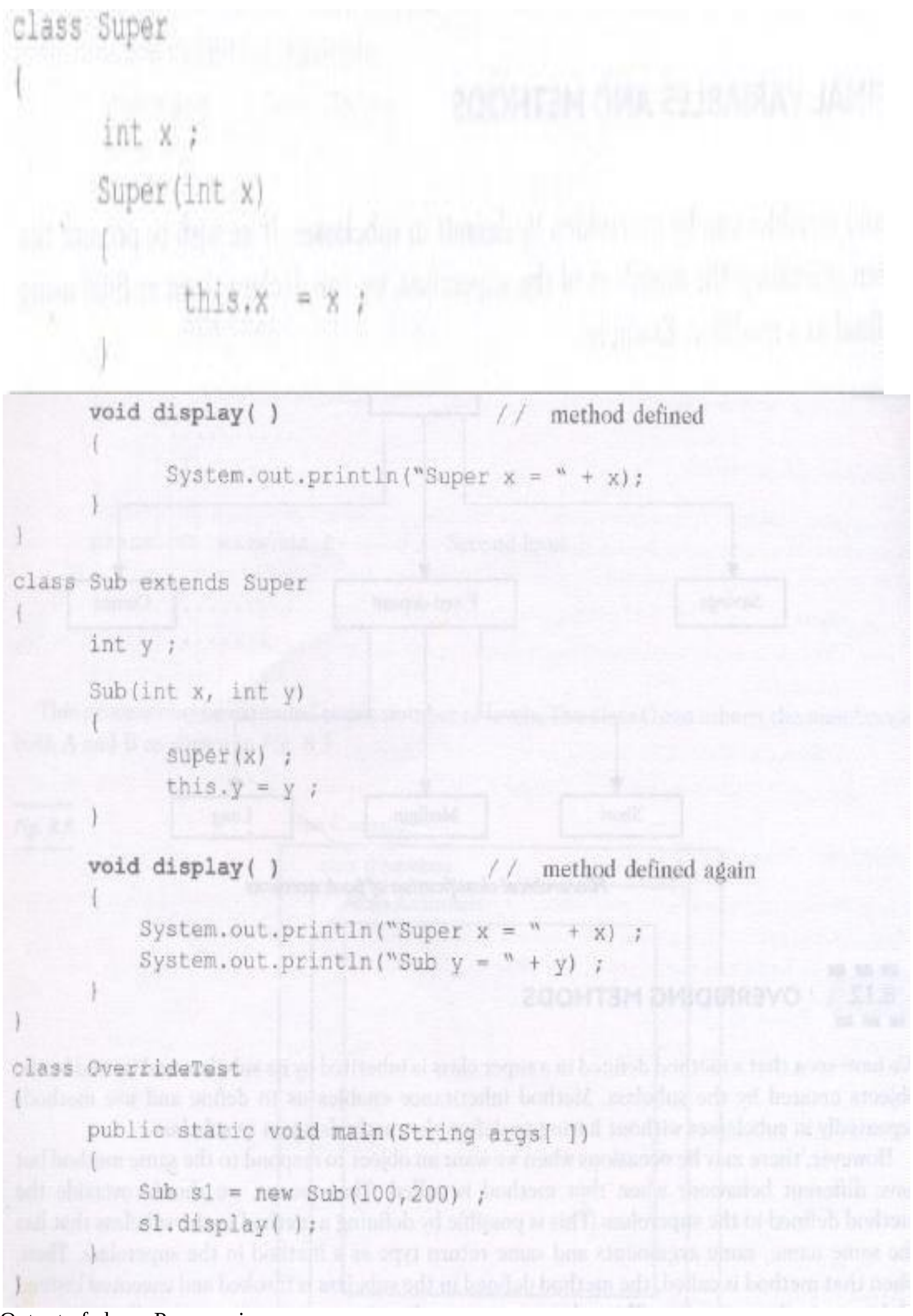
Another interesting application of inheritance is to use it as a support to the hierarchical design of a program. Many programming problems can be cast into a hierarchy where certain features of one level are shared by many others below the level. As an example, Above Figure shows a Hierarchical classification of accounts in a commercial bank. This is possible because all the accounts posses certain commonfeatures.



Hierarchical classification of bank accounts

5.5 OVERRIDINGMETHODS

Method inheritance enables us to define and use methods repeatedly in subclasses without having to define the methods again in subclass. However, there may be occasions when we want an object to respond to the same method but have different behaviour when that method is called. That means, we should override the method defined in the superclass. This is possible by defining a method in the subclass that has the same name, same arguments and same return type as a method in the superclass. Then, when that method is called, the method defined in the subclass is invoked and executed instead of the one in the superclass. This is known as overriding Program illustrates the concept of overriding. The method display ( ) is overridden.



```
class Super
{
      int x ;
      Super(int x)
      {
            this.x  = x ;
      }
      void display( )                    //  method defined
      {
            System.out.println(“Super x = “ + x);
      }
}
class Sub extends Super
{
      int y ;
      Sub(int x, int y)
      {
            super(x) ;
            this.y = y ;
      }
      void display( )                   //  method defined again
      {
          System.out.println(“Super x = “  + x) ;
          System.out.println(“Sub y = “ + y) ;
      }
}
class OverrideTest
{
      public static void main(String args[ ])
      {
          Sub s1 = new Sub(100,200) ;
          s1.display( );
      }
}
```

Output of above Program is:

```
Super x = 100
Sub y = 200
```

FINALVARIABLES AND METHODS

All methods and variables can be overridden by default in subclasses. If we wish to prevent the subclasses from overriding the members of the superclass, we can declare them as final using the keyword final as a modifier.

Example:

```
finalint SIZE = 100;
final void showstatus (){................ }
```

Making a method final ensures that the functionality defined in this, method will never be altered in any way. Similarly, the value of a final variable can never be changed. Final variables, behave like class variables and they do not take any space on individual objects of the class.

5.6 FINALCLASSES

Sometimes we may like to prevent a class being further sub classed for security reasons. A class that cannot be sub classed is called a final class. This is achieved in Java using the keyword final as follows: -

```
final class Aclass {…}
final class Bclass extendsSomeclass{................}
```

Any attempt to inherit these classes will cause an error and the compiler will not allow it.

Declaring a class final prevents any unwanted extensions to the class. It also allows the compiler to perform some optimizations when a method of a final class is invoked.

5.6.1 FINALIZER METHODS

Java supports a concept called finalization, which is just opposite to initialization. Java run-time 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++.

The finalizer method is simply finalize ( ) and can be added to any class. Java calls that method whenever it is about to reclaim the space for that object~ The finalize method should explicitly define the tasks to 'be performed.

5.7 ABSTRACT METHODS ANDCLASSES

We can indicate that a method must always be redefined in a subclass, 'thus making overriding compulsory. This is done using the modifier keyword abstract in the method definition.

Example:

```
abstract class Shape
{
  ABSTRACT METHODS AND CLASSES
   ….……
   ..………
  abstract void draw( };
  ... …. .....
}
```

When a class contains one or more abstract methods, it should also be declared abstract as shown in the example above.While using abstract classes, we must satisfy the following conditions:

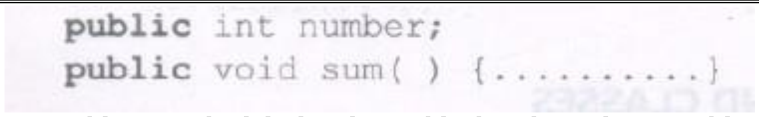
We cannot use abstract classes to instantiate objects directly. For example, Shape s = new Shape ( ) is illegal because shape is an abstract class. The abstract methods of an abstract class must be defined in its subclass. We cannot declare abstract constructors 10r abstract static methods.

5.8 VISIBILITYCONTROL

Visibility control is used 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. They provide different levels of protection as described below

public Access

public Visibility control is visible to all the classes even outside the class where it isdefined. Example:



```
public int number;
public void sum( ) {..........}
```

A variable or method declared as public has the widest possible visibility and accessible everywhere.

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 the "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.

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 subclasses in the same package but also to subclasses in other packages. Note that non-subclasses in other packages cannot access the "protected" members.

private Access

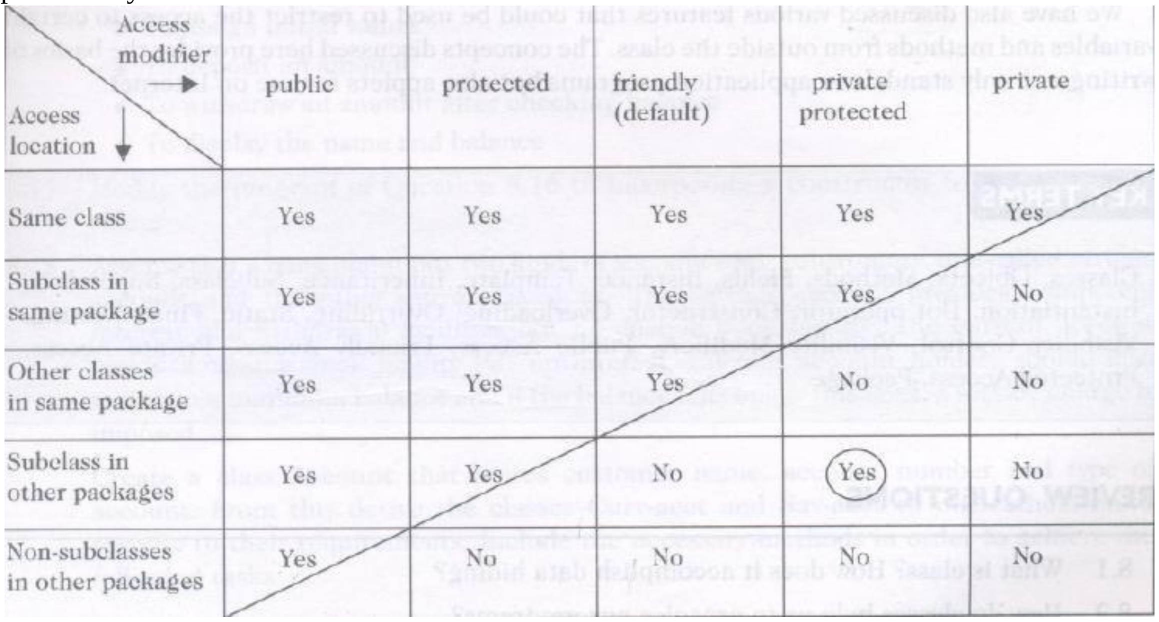
private fields enjoy the highest degree of protection. They are accessible only within their own class. They cannot be inherited by subclasses and therefore not accessible in subclasses. A method declared as private behaves like a method declared as final. It prevents the method from being subclassed. Also note that we cannot override a non-private method in a subclass and then make it private.

private protected Access

A field can be declared with two keywords private and protected together like:

Private protected intcodeNumber;

This gives a visibility level in between the "protected" access and "private" access. This modifier makes the fields visible in all subclasses regardless of what package they are in. Remember, these fields are not accessible by other classes in the same package. Table summarises the visibility provided by various accessmodifiers.



Access modifier → / Access location ↓	public	protected	friendly (default)	private protected	private
Same class	Yes	Yes	Yes	Yes	Yes
Subclass in same package	Yes	Yes	Yes	Yes	No
Other classes in same package	Yes	Yes	Yes	No	No
Subclass in other packages	Yes	Yes	No	Yes	No
Non-subclasses in other packages	Yes	No	No	No	No

Rules of Thumb

Given below are some simple rules for applying appropriate access modifiers.

1. Use public if the field is to be visibleeverywhere.
2. Use protected if the field is to be visible everywhere in the current package and also subclasses in otherpackages.
3. Use "default" if the field is to be visible everywhere in the current packageonly.
4. Use private protected if the field is to be visible only in subclasses, regardless ofpackages.
5. Use private if the field is *not* to be visible anywhere except in its ownclass.

Chapter-6
Arrays Strings and Vectors

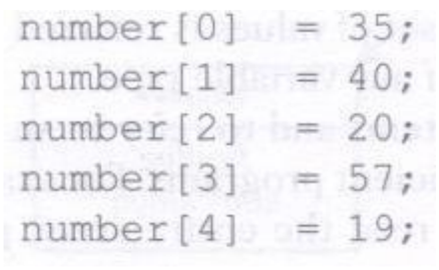
6.1 Introduction to Arrays

An array is a group of contiguous or related data items that share a common name. For instance, we can define an array name salary to represent a set of salaries of a group of employees. A particular value is indicated by writing a number called *index* number or *subscript* in brackets after the array name. For example, salary[10] represents the salary of the 10th employee. While the complete set of values is referred to as an *array,* the individual values are called *elements.* Arrays can be of any variable type.

6.2 One-dimensionalarrays

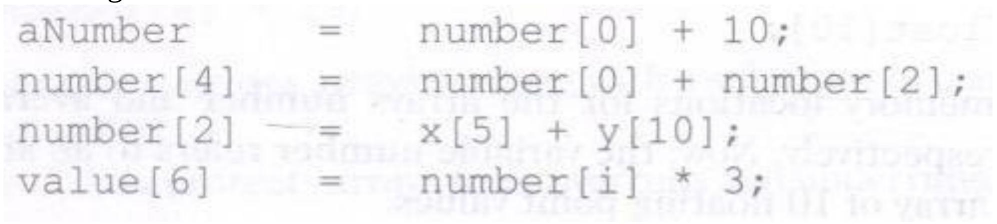
A list of items can be given one variable name using only one subscript and such a variable is called a *single-subscripted* variable or a *one-dimensional* array. In mathematics, we often deal with variables that aresingle-subscripted.
In Java, single-subscripted variable x I can be expressed as
x[l], x[2], x[3] x[n] The subscript can begin with number 0. That is x[0] is allowed. For example, if we want to represent a set of five numbers, say (35, 40, 20, 57, 19), by an array variable number, then we may create the variable number as follows
int number[ ] = **new** int[5]; and the computer reserves five storage location

The values to the array elements can be assigned as follows:



```
number[0]  = 35;
number[1]  = 40;
number[2]  = 20;
number[3]  = 57;
number[4]  = 19;
```

These elements may be used in programs just like any other Java variable. For example, the following are valid statements:



```
aNumber     =   number[0] + 10;
number[4]   =   number[0] + number[2];
number[2]   =   x[5] + y[10];
value[6]    =   number[i] * 3;
```

The subscript of an array can be integer constants, integer variables like i , or expressions that yield integers.

Creating an array
Like any other variables, arrays must be declared and created in the computer memory before they are used. Creation of an array involves three steps:

1. Declare thearray
2. Create memorylocations

3. Put values into the memorylocations.

Declaration of Arrays

Arrays in Java may be declared in two forms:

Form1 type arrayname[ ];
Form2 type [ ] arrayname;

Examples:

```
int           number[ ];
float         average[ ];
int[ ]        counter;
float[ ]      marks;
```

We do not enter the size of the arrays in the declaration.

Creation of Arrays

After declaring an array, we need to create it in the memory. Java allows us to create arrays using new operator only, as shown below

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

Examples:

```
number       = new int[5];
average      = new float[10];
```

These lines create necessary memory locations for the arrays number and average and designate them as int and float respectively. Now, the variable number refers to an array of 5 integers and average refers to an array of 10 floating point values. It is also possible to combine the two steps-declaration and creation-into one as shown
below:
int number [ ] = new int[5];

Initialization of Arrays

The final step is to put values into the array created. This process is known as *initialization* *This*is done using the array subscripts as shown below.

```
arrayname[subscript] = value ;
```

Example:

```
number[0] = 35;
number[1] = 40;
```

Note that Java creates arrays starting with a subscript of 0 and ends with a value one less than the *size* specified.

We can also initialize arrays automatically in the same way as the ordinary variables when they are declared, as shown below:

```
type arrayname[ ] = {list of values};
```

The array initializer is a list of values separated by commas and surrounded by curly braces. Note that no size is given. The compiler allocates enough space for all the elements specified in the list.

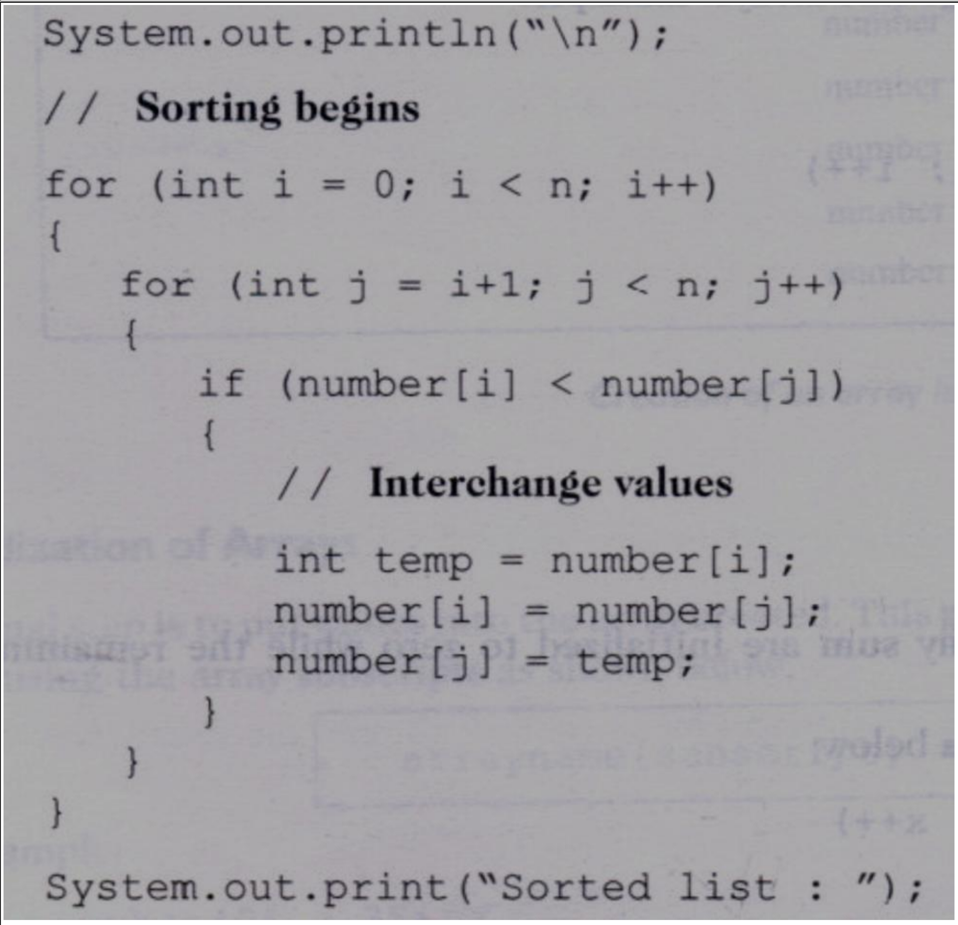
Example: int number [ ] = {35, 40, 20, 57, 19};

Array Length

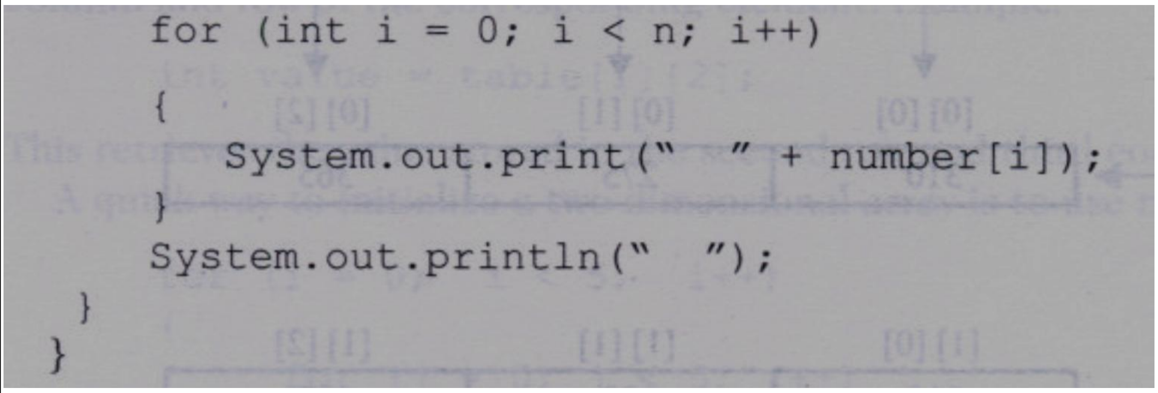
In Java, all arrays store the allocated size in a variable named length. We can access the length of the array a using a.length. Example: intaSize =a.length;
This information will be useful in the manipulation of arrays when their sizes are not known.

Sorting a *list* of numbers

```
class NumberSorting
{
   public static void main(String args[ ])
   {
      int number[] = { 55, 40, 80, 65, 71 };
      int n = number.length;
      System.out.print("Given list : ");
      for (int i = 0; i < n; i++)
      {
         System.out.print("  " + number[i]);
      }
```

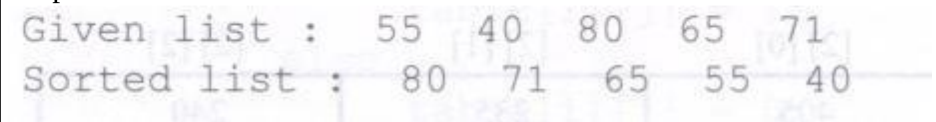


```
System.out.println("\n");
// Sorting begins
for (int i = 0; i < n; i++)
{
   for (int j = i+1; j < n; j++)
   {
      if (number[i] < number[j])
      {
         // Interchange values
         int temp = number[i];
         number[i] = number[j];
         number[j] = temp;
      }
   }
}
System.out.print("Sorted list : ");
```



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

Output



```
Given list :  55  40  80  65  71
Sorted list :  80  71  65  55  40
```

6.3 Two-dimensionalarrays

In mathematics, we represent a particular value in a matrix by using two subscripts such as V_{ij} Here v denotes the entire matrix and vii refers to the value in the i^{th} row and j^{th} column. Java allows us to define such tables of items by using *two-dimensional*arrays.
As with the single dimensional arrays, each dimension of the array is indexed from zero to its maximum size minus
one; the first index selects the row and the second index selects the column within that row.
For creating two-dimensional arrays, we must follow the same steps as that of simple arrays.
We may create a two-dimensional array like this

```
int myArray[ ][ ];
myArray = new int[3][4];
```

Or

```
int myArray[ ][ ] = new int[3][4];
```

This creates a table that can store 12 integer values, four across and three down.

Like the one-dimensional arrays, two-dimensional arrays may be initialized by following their declaration with a list of initial values enclosed in braces. Forexample,

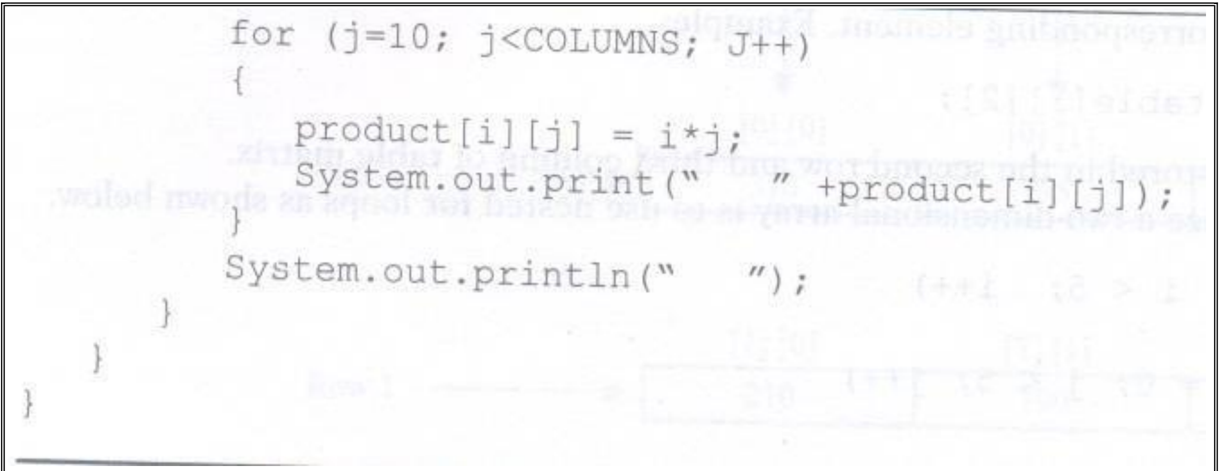
int table[2] [3] = {O,O,O,l,l,l}; initializes the elements of the first row to zero and the second row to one. The initialization is done row by row. The above statement can be equivalently written as int table [ ] [ ] = {{O,O,O}, {l,l,l}}; by surrounding the elements of each row by braces. We can also initializea two-dimensional array in the form of a matrix as shown below:

```
int table[ ][ ] = {
                    {0,0,0},
                    {1,1,1}
                  };
```

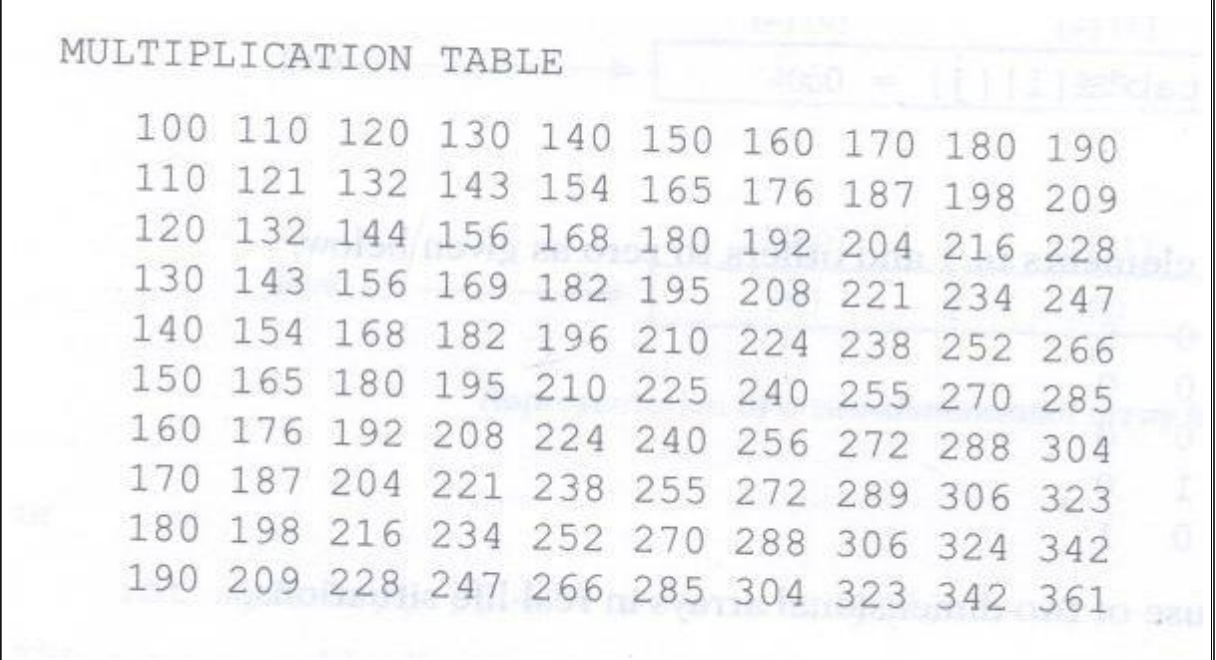
We can refer to a value stored in a two-dimensional array by using subscripts for both the column and row of the corresponding element. Example: int value = table[1][2]; This retrieves the value stored in the second row and third column of tablematrix.

Application of *two-dimensional* ***arrays***

```
class MulTable
{
   final static int ROWS = 20;
   final static int COLUMNS = 20;

   public static void main(String args[ ])
   {
      int product[][] = new int[ROWS][COLUMNS];
      int row, column;

      System.out.println("MULTIPLICATION TABLE");
      System.out.println("  ");

      int i,j;
      for (i=10; i<ROWS; i++)
      {
```

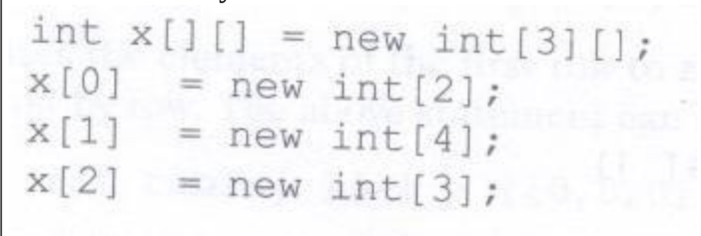


```
        for (j=10; j<COLUMNS; J++)
        {
            product[i][j] = i*j;
            System.out.print("   " +product[i][j]);
        }
        System.out.println("   ");
    }
  }
}
```



```
MULTIPLICATION TABLE

    100 110 120 130 140 150 160 170 180 190
    110 121 132 143 154 165 176 187 198 209
    120 132 144 156 168 180 192 204 216 228
    130 143 156 169 182 195 208 221 234 247
    140 154 168 182 196 210 224 238 252 266
    150 165 180 195 210 225 240 255 270 285
    160 176 192 208 224 240 256 272 288 304
    170 187 204 221 238 255 272 289 306 323
    180 198 216 234 252 270 288 306 324 342
    190 209 228 247 266 285 304 323 342 361
```

Variable Size Arrays
Java treats multidimensional arrays as "arrays of arrays". It is possible to declare a two-dimensional array as follow



```
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 row.

6.4 Strings

String manipulation is the most common part of many Java programs. Strings represent a sequence of characters. The easiest way to represent a sequence of characters in Java is by using a character array. Example:

```
char charArray[ ] = new char[4];
charArray[0]  = 'J';
charArray[1]  = 'a';
charArray[2]  = 'v';
charArray[3]  = 'a';
```

Although character arrays have the advantage of being able to query, their length, they themselves are not good enough to support the range of operations we may like to perform on strings. For example, copying one character array into another might require a lot of book keeping effort. Java is equipped to handle these situations more efficiently. In Java, strings are class objects and implemented using two classes, namely, **String** and **StringBuffer.** A Java string is an instantiated object of the **String** class. Java strings, as compared to C strings, are more reliable and predictable. This is basically due to C's lack of bounds-checking. A Java string is not a character array and is not NULL terminated. Strings may be declared and created asfollows:

```
String   stringName;
StringName = new String ("string");
```

Example:
String firstName;
firstName = new String("Anil");
These two statements may be combined as follows:
String firstName = new String("Anil");
Like arrays, it is possible to get the length of string using the **length** method of the **String** class.
int m = firstName.length( ) ;
Note the use of parentheses here. Java strings can be concatenated using the + operator.
Examples:

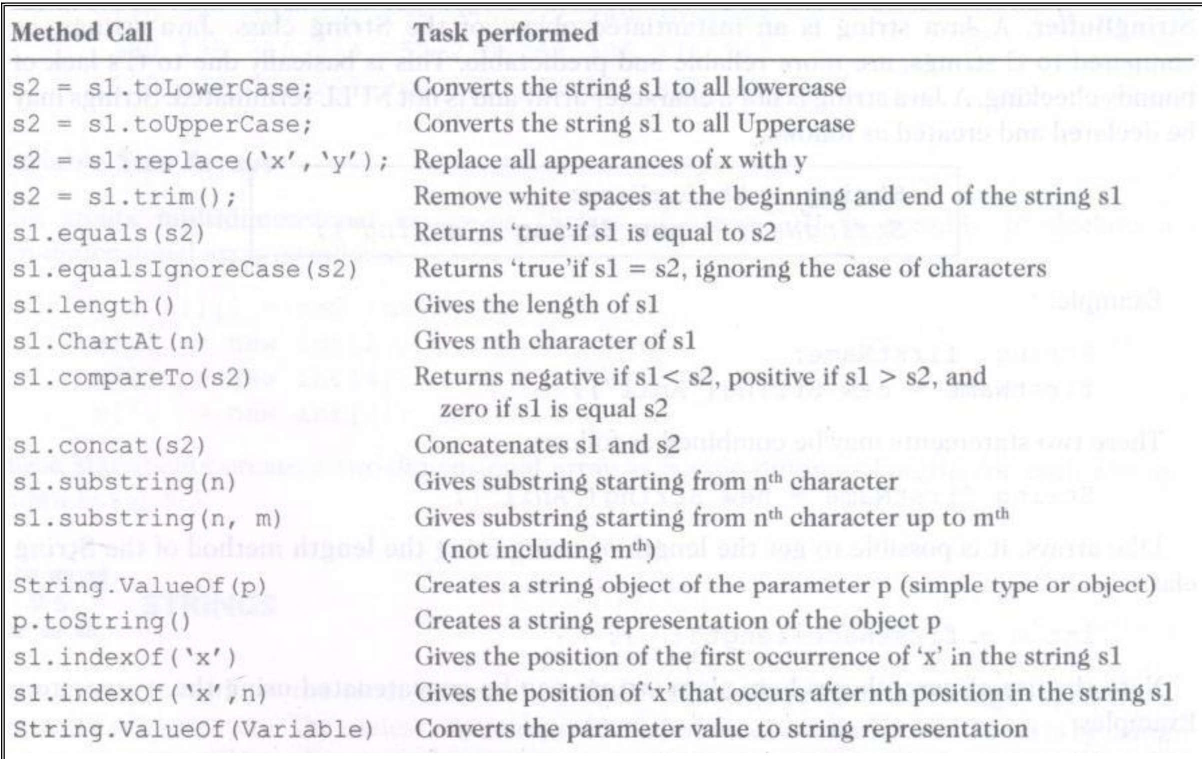
```
String fullName  =  name1 + name2;
String city1     =  "New" + "Delhi";
```

wherenamel and name2 are Java strings containing string constants.

String Arrays
We can also create and use arrays that contain strings. The statement
String itemArray[ ] = new String[3];
Will create an itemArray of size 3 to hold three string constants. We can assign the strings to the itemArray element by element using three different statements or more efficiently using a for loop

String Methods
The String class defines a number of methods that allow us to accomplish a variety of string manipulation tasks.

Some Most Commonly Used String Methods

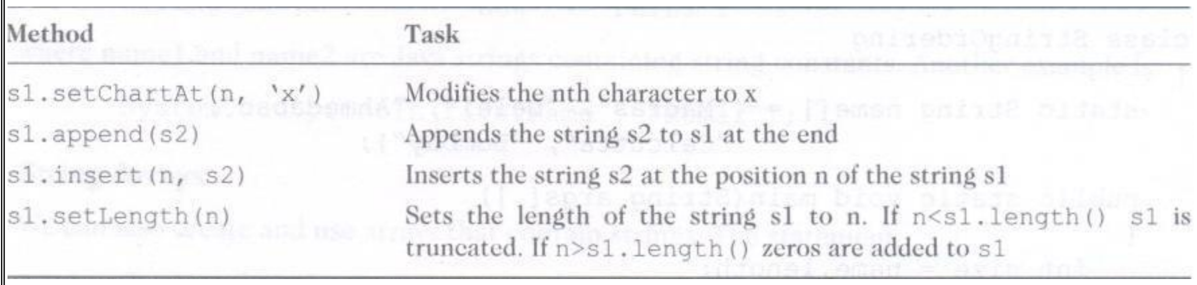


Method Call	Task performed
`s2 = s1.toLowerCase;`	Converts the string s1 to all lowercase
`s2 = s1.toUpperCase;`	Converts the string s1 to all Uppercase
`s2 = s1.replace('x', 'y');`	Replace all appearances of x with y
`s2 = s1.trim();`	Remove white spaces at the beginning and end of the string s1
`s1.equals(s2)`	Returns 'true'if s1 is equal to s2
`s1.equalsIgnoreCase(s2)`	Returns 'true'if s1 = s2, ignoring the case of characters
`s1.length()`	Gives the length of s1
`s1.ChartAt(n)`	Gives nth character of s1
`s1.compareTo(s2)`	Returns negative if s1< s2, positive if s1 > s2, and zero if s1 is equal s2
`s1.concat(s2)`	Concatenates s1 and s2
`s1.substring(n)`	Gives substring starting from nth character
`s1.substring(n, m)`	Gives substring starting from nth character up to mth (not including mth)
`String.ValueOf(p)`	Creates a string object of the parameter p (simple type or object)
`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
`s1.indexOf('x',n)`	Gives the position of 'x' that occurs after nth position in the string s1
`String.ValueOf(Variable)`	Converts the parameter value to string representation

StringBuffer Class

StringBuffer is a peer class of String. While String creates strings of fixed_length, StringBuffer
creates strings offlexible length that can be modified in terms of both length and content. We can insert characters and substrings in the middle of a string, or append another string to the end.

Commonly Used StringBuffer Methods



Method	Task
`s1.setChartAt(n, 'x')`	Modifies the nth character to x
`s1.append(s2)`	Appends the string s2 to s1 at the end
`s1.insert(n, s2)`	Inserts the string s2 at the position n of the string s1
`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

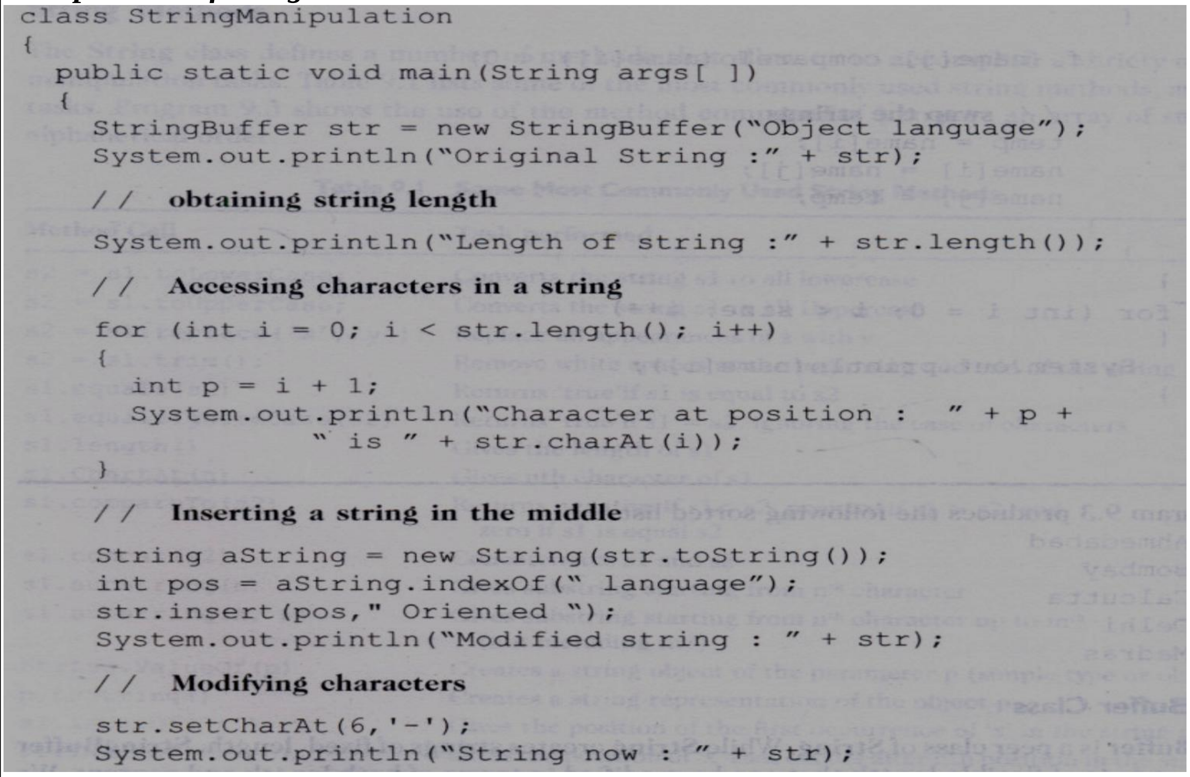
Alphabetical ordering **of strings**

```
class StringOrdering
{
   static String name[] = {“Madras”, “Delhi”, “Ahmedabad”,
                              “Calcutta”, “Bombay”};

   public static void main(String args[ ])
   {
      int size = name.length;
      String temp = null;
      for (int i = 0; i < size; i++)
      {

        for (int j = i+1; j < size; j++)
        {

          if (name[j].compareTo(name[i]) < 0)
          {
            //  swap the strings
            temp = name[i];
            name[i] = name[j];
            name[j] = temp;
          }
        }
      }
      for (int i = 0; i < size; i++)
      {
        System.out.println(name[i]);
      }
   }
}
```

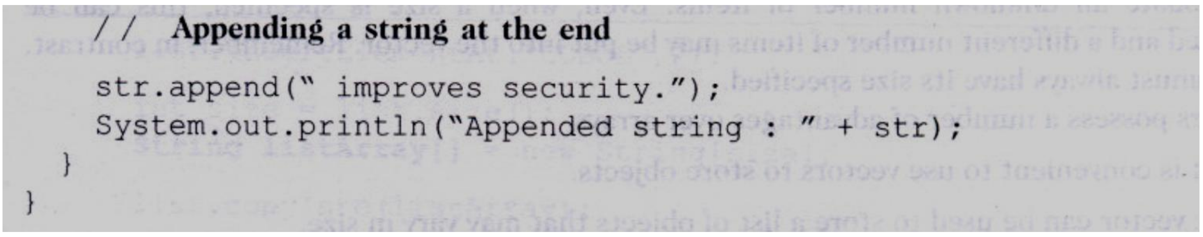
Program produces the following sorted list:

```
Ahmedabad
Bombay
Calcutta
Delhi
Madras
```

Manipulation of strings

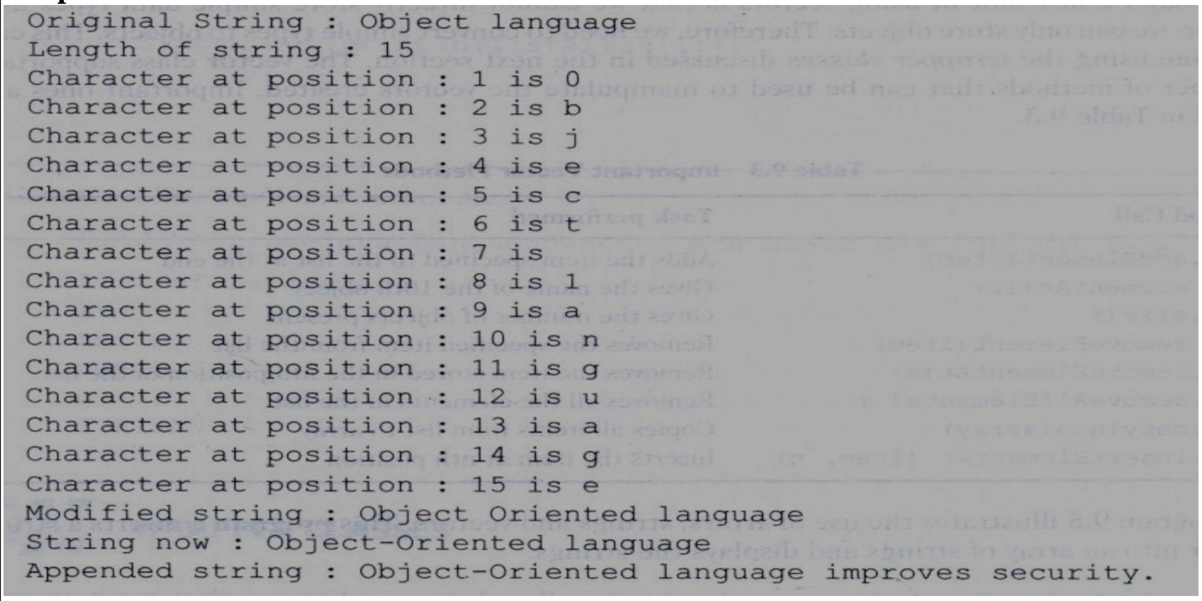


```
class StringManipulation
{
  public static void main(String args[ ])
  {
    StringBuffer str = new StringBuffer(“Object language”);
    System.out.println(“Original String :” + str);

    //  obtaining string length

    System.out.println(“Length of string :” + str.length());

    //  Accessing characters in a string

    for (int i = 0; i < str.length(); i++)
    {
      int p = i + 1;
      System.out.println(“Character at position :  ” + p +
             “ is ” + str.charAt(i));
    }

    //  Inserting a string in the middle

    String aString = new String(str.toString());
    int pos = aString.indexOf(“ language”);
    str.insert(pos," Oriented “);
    System.out.println(“Modified string : ” + str);

    //  Modifying characters

    str.setCharAt(6,'-');
    System.out.println(“String now : ” + str);
```



```
    //  Appending a string at the end

    str.append(“ improves security.”);
    System.out.println(“Appended string : ”+ str);
  }
}
```

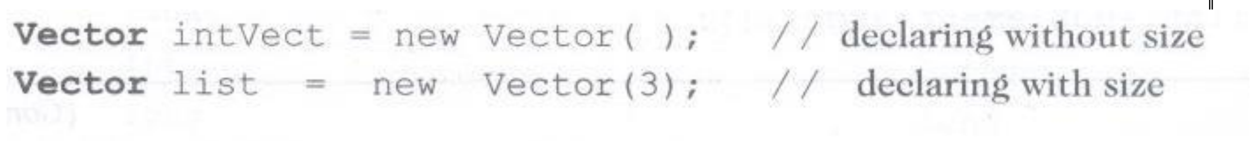
Output



```
Original String : Object language
Length of string : 15
Character at position : 1 is O
Character at position : 2 is b
Character at position : 3 is j
Character at position : 4 is e
Character at position : 5 is c
Character at position : 6 is t
Character at position : 7 is
Character at position : 8 is l
Character at position : 9 is a
Character at position : 10 is n
Character at position : 11 is g
Character at position : 12 is u
Character at position : 13 is a
Character at position : 14 is g
Character at position : 15 is e
Modified string : Object Oriented language
String now : Object-Oriented language
Appended string : Object-Oriented language improves security.
```

6.5 Vectors

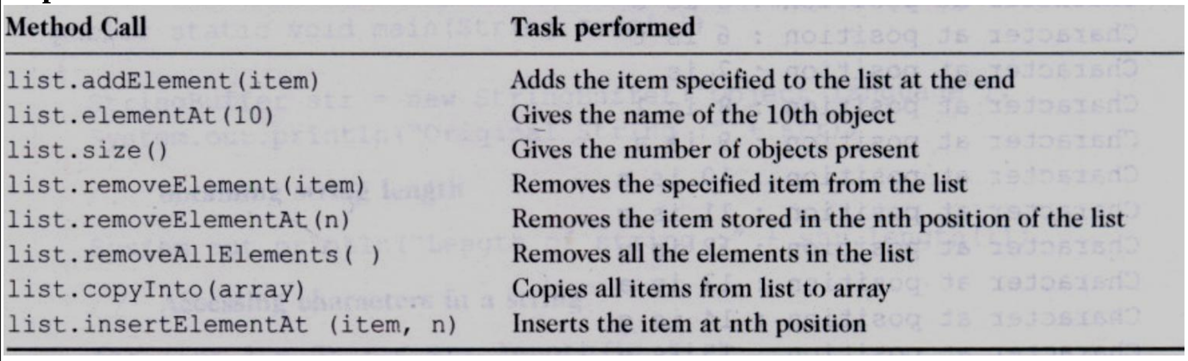
Java does not support the concept of variable arguments to a function. This feature can be achieved in Java through the use of the Vector class contained in the java.util package. This class can be used to create a generic dynamic array known as *vector* that can hold *objects of* *any type* and *any number*. The objects do not have to be homogenous. Arrays can be easily implemented as vectors. Vectors are created like arrays as follows:



```
Vector intVect = new Vector( );    // declaring without size
Vector list  =  new  Vector(3);    //  declaring with size
```

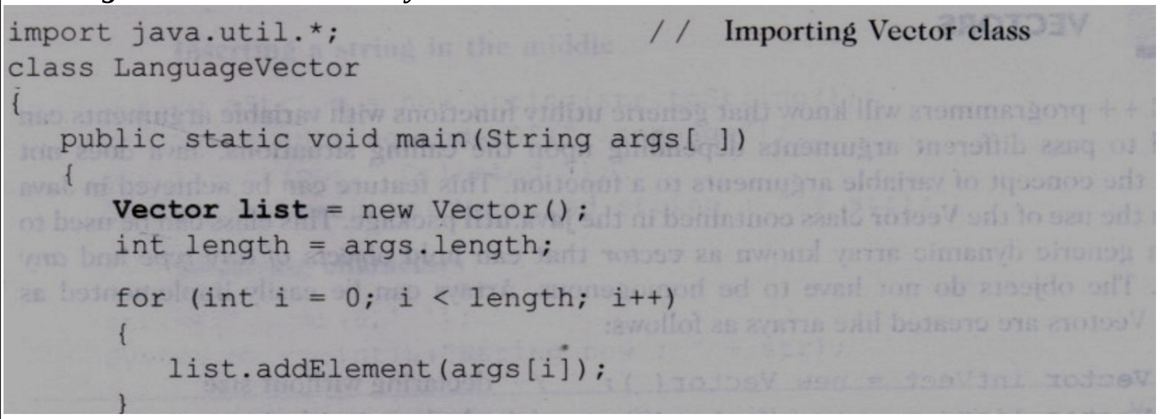
Note that a vector can be declared without specifying any size explicitly. A vector can accommodate an unknown number of items. Even, when a size is specified, this can be overlooked and a different number of items may be put into the vector. Remember, in contrast, an array must always have its size specified. Vectors possess a number of advantages overarrays.
1. It is convenient to use vectors to storeobjects.
2. A vector can be used to store a list of objects that may vary insize.
3. We can add and delete objects from the list as and whenrequired.
A major constraint in using vectors is that we cannot directly store simple data types in a vector; we can only store objects. Therefore, we need to convert simple types to objects. The vector class supports a number of methods that can be used to manipulate the vectors created.

Important Vector Methods



Method Call	Task performed
`list.addElement(item)`	Adds the item specified to the list at the end
`list.elementAt(10)`	Gives the name of the 10th object
`list.size()`	Gives the number of objects present
`list.removeElement(item)`	Removes the specified item from the list
`list.removeElementAt(n)`	Removes the item stored in the nth position of the list
`list.removeAllElements( )`	Removes all the elements in the list
`list.copyInto(array)`	Copies all items from list to array
`list.insertElementAt (item, n)`	Inserts the item at nth position

Working with vectors and arrays



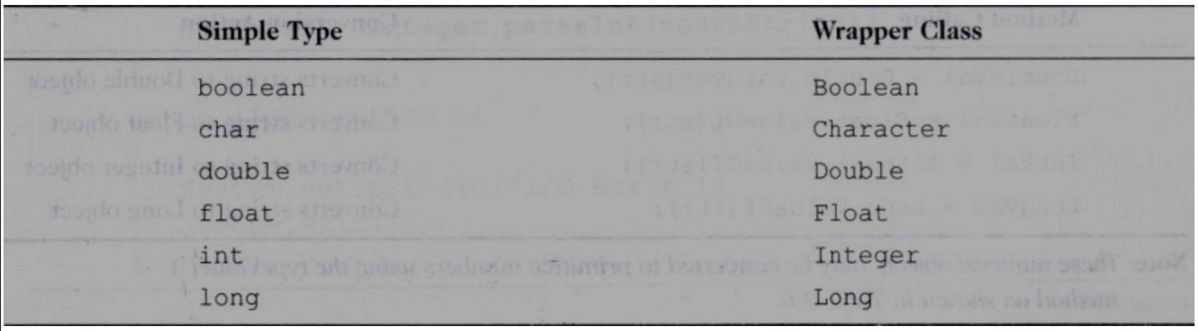
```
import java.util.*;                    //  Importing Vector class
class LanguageVector
{
   public static void main(String args[ ])
   {
      Vector list = new Vector();
      int length = args.length;
      for (int i = 0; i < length; i++)
      {
         list.addElement(args[i]);
      }
      list.insertElementAt("COBOL",2);

      int size = list.size();
      String listArray[] = new String[size];

      list.copyInto(listArray);

      System.out.println("List of Languages");
      for (int i = 0; i < size; i++)
      {
         System.out.println(listArray[i]);
      }
   }
}
```

Command line input and output are:

```
C:\JAVA\prog>java LanguageVector Ada BASIC C++ FORTRAN Java

List of Languages
Ada
BASIC
COBOL
C++
FORTRAN
Java
```

6.6 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 contained in the java. Lang package. Table shows the simple data types and their corresponding wrapper class types

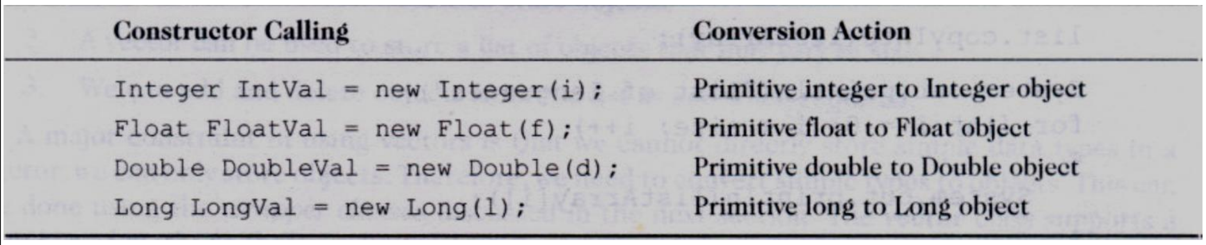
Wrapper Classes for Converting Simple Types



Simple Type	Wrapper Class
`boolean`	`Boolean`
`char`	`Character`
`double`	`Double`
`float`	`Float`
`int`	`Integer`
`long`	`Long`

The wrapper classes have a number of unique methods for handling primitive data types and objects. They are listed in the following tables.

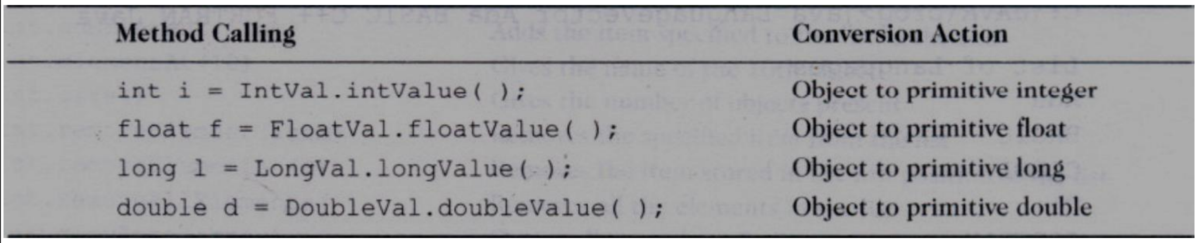
Converting Primitive Numbers to Object Numbers Using Constructor Methods



Constructor Calling	Conversion Action
`Integer IntVal = new Integer(i);`	Primitive integer to Integer object
`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

Note: *These numeric objects may be converted to primitive numbers using the typeValue()*

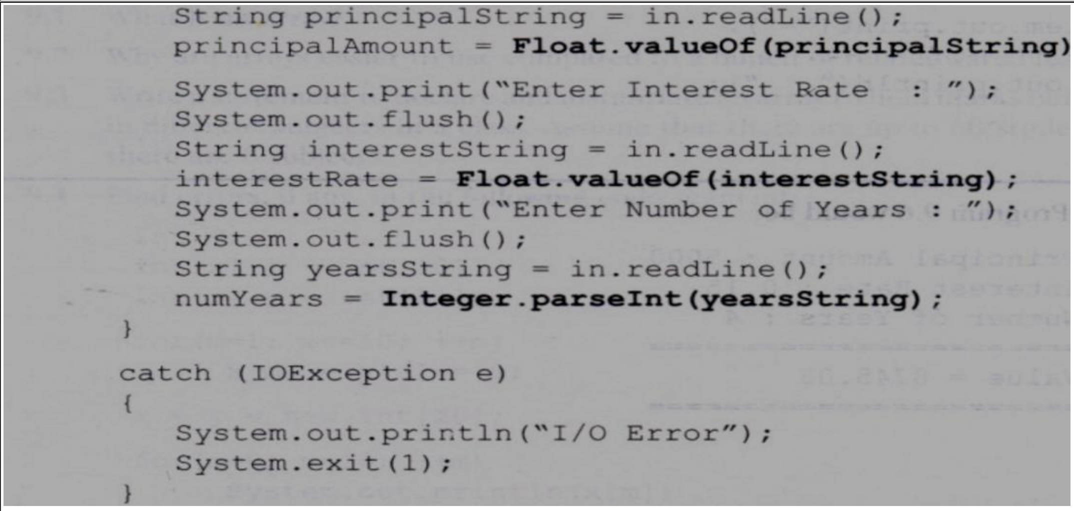
Converting Numeric Strings to Primitive Numbers Using Parsing Methods

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

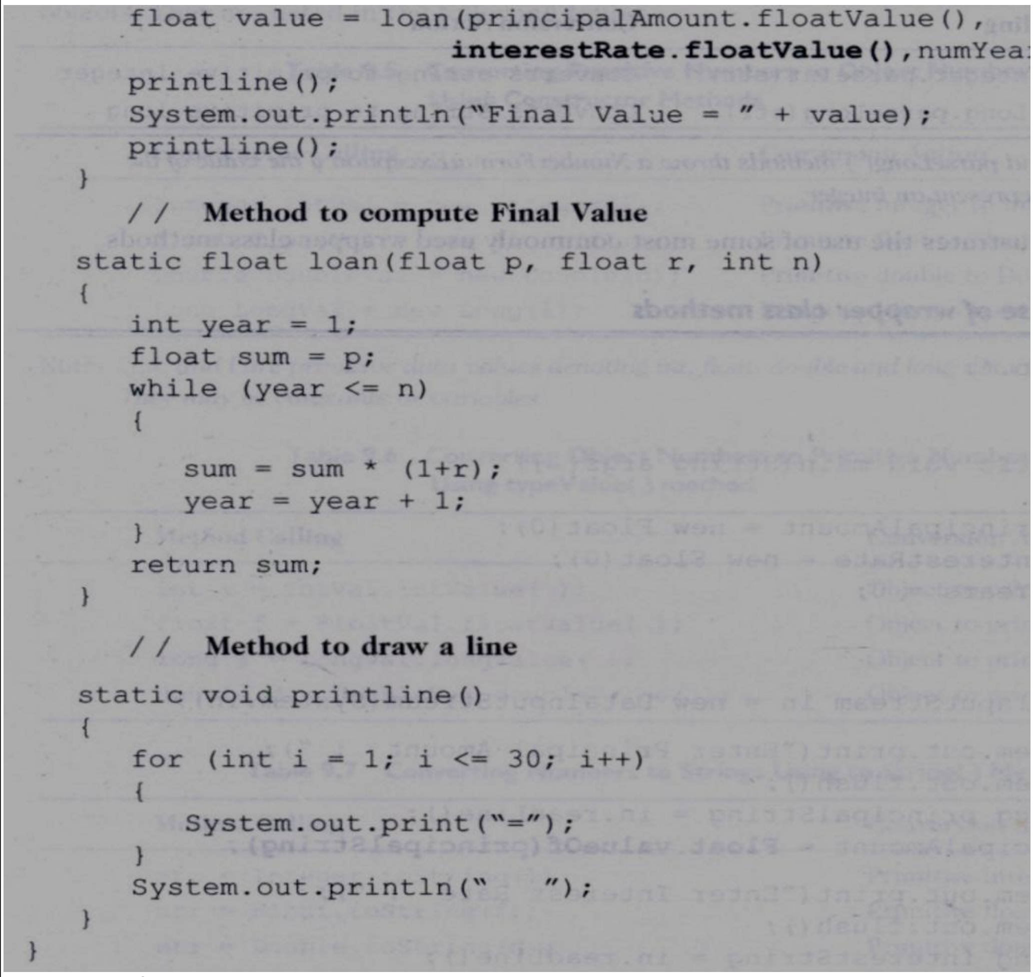
Note: *parseInt() and parseLong( ) methods throw a NumberFormatException if the value of the str does not represent an integer.*

Use of *wrapper* class *methods*

```
import java.io.*;
class Invest
{
   public static void main(String args[ ])
   {
      Float principalAmount = new Float(0);
      Float interestRate = new Float(0);
      int numYears = 0;
      try
      {
         DataInputStream in = new DataInputStream(System.in);
         System.out.print("Enter Principal Amount  : ");
         System.out.flush();
```

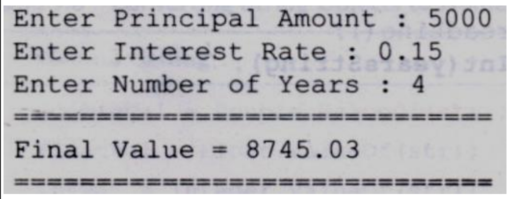


```
        String principalString = in.readLine();
        principalAmount = Float.valueOf(principalString)
        System.out.print("Enter Interest Rate   : ");
        System.out.flush();
        String interestString = in.readLine();
        interestRate = Float.valueOf(interestString);
        System.out.print("Enter Number of Years : ");
        System.out.flush();
        String yearsString = in.readLine();
        numYears = Integer.parseInt(yearsString);
    }
    catch (IOException e)
    {
        System.out.println("I/O Error");
        System.exit(1);
    }
```



```
    float value = loan(principalAmount.floatValue(),
                         interestRate.floatValue(),numYea
    printline();
    System.out.println("Final Value = " + value);
    printline();
  }
    // Method to compute Final Value
  static float loan(float p, float r, int n)
  {
    int year = 1;
    float sum = p;
    while (year <= n)
    {
      sum = sum * (1+r);
      year = year + 1;
    }
    return sum;
  }
    // Method to draw a line
  static void printline()
  {
    for (int i = 1; i <= 30; i++)
    {
      System.out.print("=");
    }
    System.out.println("   ");
  }
}
```

The outputof Program



```
Enter Principal Amount : 5000
Enter Interest Rate : 0.15
Enter Number of Years : 4
==============================
Final Value = 8745.03
==============================
```

Assignment 3

1. **In java arrays are**
   A. Objects
   B. Objectreferences
   C. Primitive datatype
   D. None of theabove
   Answer: Option A
2. **Which one of the following is a validstatement?**
   A. char[]c=newchar();
   B. char[]c=newchar[5];
   C. char[]c=newchar(4);
   D. char[]c=newchar[];
   Answer: Option B
3. **When you pass an array to a method, the methodreceives**
   A. A copy of thearray
   B. A copy of the firstelement
   C. The reference of thearray
   D. The length of thearray
   Answer: Option C
4. **Which is true**
   A. "X extends Y" is correct if and only if X is a class and Y is aninterface
   B. "X extends Y" is correct if and only if X is an interface and Y is aclass
   C. "X extend Y" is correct if X and Y are either both class or bothinterface
   D. "X extend Y" is correct for all combinations of X and Y being classes and/or interfaces
   Answer: Option C
5. **Tostring() method is definedin**
   A. Java.lang.String
   B. Java.lang.Object
   C. Java.lang.util
   D. None ofthese
   Answer: Option B
6. **The string methodcompareTo()returns**
   A. True
   B. False
   C. An int value
   D. 1
   E. -1
   Answer: OptionC
7. **The class stringbelongsto------- package**.
   A. Java.awt
   B. Java.lang
   C. Java.applet
   D. Java.string
   Answer: Option B
8. **How many constructor String classhave?**
   A. 2
   B. 7
   C. 13
   D. 11

E. None ofthese

Answer: Option C

9. **An interface is a blueprint of a class. It has static constants and abstract methods.An interface is a blueprint of a class. It has static constants and abstractmethods.**
   A. True
   B. False
   Answer: True
10. **Which variables are created when an object is created with the use of the keyword'new' and destroyed when the object isdestroyed?**
    A. Local variables
    B. Instance variable
    C. Staticvariable
    D. Classvariable
    Answer: Option B
11. **Breaking a string or stream into meaningful independent words is known as tokenization.**
    A. True
    B. False
    Answer: Option A
12. **What can be accessed or inherited without actual copy of code to eachprogram?**
    A. Browser
    B. Applet
    C. Package
    D. None of theabove
    Answer: Option C
13. **Which provides accessibility to classes andinterface?**
    A. Import
    B. Staticimport
    C. All theabove
    D. None of theabove
    Answer: Option A
14. **The following program is an exampleof?**

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

String s="Sachin";
System.out.println(s.length());//6
}
}
```

    A. Length()method
    B. Intern()method
    C. Trim()method
    D. charAt()method
    Answer: Option A
15. **If you are inserting any value in the wrong index as shownbelow,**

```
1. int a[]=new int[5];
2. a[10]=50;
```

    it wouldresultin________.
    A. NullPointerException
    B. ArrayIndexOutOfBoundsException

C. ArithmeticException
D. NumberFormatException

Answer: Option B

16. **Which interfaces provide methods for batch processing inJDBC?**
A. java.sql.Statement
B. java.sql.PreparedStatement
C. All theabove
D. None of theabove

Answer: Option D

17. **Sessions is a part of the SessionTracking and it is for maintaining the client stateat server side.**
A. True
B. False

Answer: Option A

18. **Interceptor can change the flow of the application by returning thestring.**
A. True
B. False

Answer: True

19. **Which method can set or change the text in aLabel?**
A. setText()
B. getText()
C. All theabove
D. None of theabove

Answer: OptionA

20. **Which methods are provided by the PrintStream class?**
A. Read data to anotherstream
B. Write data to anotherstream
C. Read data to samestream
D. Write data to samestream

Answer: Option B

21. **What is known as the classes that extend Throwable class except RuntimeExceptionand Error?**
A. CheckedException
B. UncheckedException
C. Error
D. None of theabove

Answer: Option A

22. **Java application uses an output stream to read data from a source, it may be a file,an array, peripheral device orsocket.**
A. True
B. False

Answer: False

23. **Which methods returns a stream that simply provides the raw bytes from thedatabase without anyconversion?**
A. getCharacterStream
B. getBinaryStream
C. getAsciiStream
D. getUnicodeStream

Answer: Option B

24. **tostring() method is definedin**
A. Java.lang.string
B. Java.lang.object
C. Java.lang.util
D. None ofthese

Answer: Option B

25. **The string methodcompareTo()returns**
    A. **True**
    B. **False**
    C. **An intvalue**
    D. **1**
    E. **-1**
    Answer: Option C

Long answer questions

1. Compare and contrast overloading and overriding methods in Java with examples?
2. Explain single level of inheritance with an example?
3. Explain Abstract methods and classes?
4. Write a note on finalizer()method?
5. How does the string class differ from string buffer class? Give the syntax and use of following method
   i) indexOf() ii) subString() iii)insert() iv)setCharAt()
6. What is a vector? Explain different methods used in implementing vectors in Java with an example
7. What is an array? How is it declared and initialized in java? Explain with an example?
8. What is a wrapper classes? Explain the usage of wrapper classes in Java?
9. Explain the different methods available in String class with an examples.
10. Explain the various types of inheritance with examples.
11. Explain multiple inheritance with an example.
12. Explain visibility control modes with examples