Strings

- Strings represent a sequence of character.
- In java, strings are class objects and implemented using two classes namely **String** & **StringBuffer.**
- A Java string is an instantiated object of **String** class.
- A string is not a character array and is not NULL terminated.

Syntax: String *stringname*;
stringname= new String("string");
or
stringname="string";

String Methods: The String class defines a number of methods that allow us to accomplish a variety of string manipulation tasks. Consider the table given below-

Method call	**Task performed**
str2=str1.toLowerCase();	Converts the string str1 to all lowercase
str2=str1.toUpperwerCase();	Converts the string str1 to all uppercase
str2=str1.replace('x','y');	Replace all appearances of x with y
str2=str1.trim();	Removes white spaces at the beginning and end of the string str1
str1.equals(str2)	Returns 'true' if str1=str2
str1.equalsIgnorCase(str2)	Returns 'true' if str1=str2, ignoring the case of characters
str1.length()	Gives the length of str1
str1.CharAt(i)	Gives i^{th} character of str1
str1.compareTo(str2)	Returns negative if str1<str2, positive if str1>str2, and zero if str1=str2
str1.concat(str2)	concatenates str1 and str2
str1.substring(n)	Gives substring starting from n^{th} character
str1.substring(n, m)	Gives substring starting from n^{th} character up to m^{th} (not including m^{th})
String.valueOf(p)	Creates a string object of the parameter p(simple type or objects)
p.toString()	Creates a string representation of the object p
str1.indexOf('x')	Gives the position of the first occurrence of 'x' in the string str1
Str1.indexOf('x', n)	Gives the position of 'x' that occurs after n^{th} position in the string str1
string.ValueOf(variable)	Converts the parameter value to string representation

str1.setCharAt(n, 'x')	Modifies the n^{th} character to x
str1.append(str2)	Appends the string Str2 to Str1 at the end
str1.insert(n, str2)	Inserts the string str2 at the position n of the string str1
str1.setLength(n)	Sets the length of the string str1 to n. If n<str1.length() str1 is truncated. If n> str1.length() zeros are added to str1.

Vectors

- Vector 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.
- Vector class contained in the **java.util** package.
- **Syntax:** Vector variable=new Vector(); //declaring without size
  or
  Vector variable=new Vector(size); // declaring with size
- A vector can accommodate an unknown number of items.
- Even when a size is specified, this can be overlooked.
- A major constraint in using vectors is that we cannot directly store simple data types in a vector; we can only store objects (we can convert simple data types into objects using wrapper classes).

Vector methods: The vector class supports a number of methods that can be used to manipulate the vectors created.

Method call	**Task performed**
variable.addElement(item)	adds the item specified to the list at the end
variable.elementAt(n)	gives the name of the n^{th} object
variable.size()	gives the number of objects present
variable.removeElement(item)	removes the specified item from the list
variable.removeElementAt(n)	removes the item stored in the nth position of the list
variable.removeAllElements()	removes all the elements in the list
variable.copyInto(array)	copies all items from the list to array
variable.insertElementAt(item, n)	inserts the item at n^{th} position

Wrapper classes

- Primitive data types (int, float, long, char, double) are converted into object types by using wrapper classes.
- They are contained in **java.lang** package.

Table-1 Wrapper classes for converting simple types-

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

Table-2 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

Table-3 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

Table-4 Converting numbers to string 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

Table-5 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

Table-6 Converting numeric string to primitive numbers using parsing method-

Method calling	**Conversion action**
int i=Integer.parseint(str);	Converts string to primitive integers
long l=Long.parseLong(str);	Converts string to primitive long