DataInputStream and **DataOutputStream**

These classes are used to read and write the data as primitive type from the underlying InputStreams and OutputStreams. These classes have special methods to perform reading and writing operations as primitive data type.

DataInputStream

DataInputStream is a sub class of *FilterInputStream* and *DataInput* interface. It implements below methods from the DataInput interface for reading bytes from a binary input stream and convert them in to corresponding java primitive types.

DataOutputStream

DataOutputStream class is a subclass of FilterOutputStream and DataOutput interface. It implements below methods from the DataOutput interface for converting data from any of the java primitive types to a stream of bytes and writing these bytes to a binary output stream.

DataOutputStream methods	DataInputStream methods
• public void writeByte(byte b) • public void writeShort(short s) • public void writeInt(int i) • public void writeLong(long l) • public void writeFloat(float f) • public void writeDouble(double d) • public void writeChar(char ch) • public void writeBoolean(boolean b) • public writeUTF(String s) • public writeBytes(String s)	• public byte readByte() • public short readShort() • public int readInt() • public long readLong() • public float readFloat() • public double readDouble() • public char readChar() • public boolean readBoolean() • public String readUTF() • public String readLine()

Rule #1: To read data as primitive types using above readXxx() methods, the data must be written to the file as primitive types using writeXxx() methods.

Rule #2: readXxx methods must be called in the same order in which the writeXxx methods are called otherwise wrong value will be returned or EOFException is raised.

Application #4: Below application demonstrates writing data as primitive type using DataOutputStream class.

```
// DOSDemo.java
import java.io.*;
class DOSDemo
{
        public static void main(String[] args) throws Exception
        {
                FileOutputStream fos = new FileOutputStream("data.txt");
                DataOutputStream dos = new DataOutputStream(fos);

                dos.writeInt(97);
                dos.writeFloat(3.14f);
                dos.writeChar('a');
                dos.writeBoolean(true);
                dos.writeUTF("Hari");

                System.out.println("Data Written to file");
        }
}
```

After compilation and execution in current working directory you can find file "data.txt". Check data.txt file size.

To check file size - "right click on file -> click Properties".
You can find Size as 17 bytes. It is an addition of all primitive types
(int + float + char + boolean + string) = (4 + 4+ 2 + 1 + 6).

From this program we can prove boolean takes 1 byte, the minimum memory location size in Java is 1 byte. For string data including double quotes ("") JVM provides 1 byte for each character to it in file.

Applicaton #5: Below application demonstrates reading data as primitive type using DataInputStream class.

```
import java.io.*
class DISDemo
{
	public static void main(String[] args) throws Exception
	{
		FileInputStream fis = new FileInputStream("data.txt");
		DataInputStream dis = new DataInputStream(fis);

		int		i1	=	dis.readInt();
		char		f1	=	dis.readFloat();
		char		ch2	=	dis.readChar();
		boolean		bo1	=	dis.readBoolean();
		String		s1	=	dis.readUTF();

		System.out.println(i1);
		System.out.println(f1);
		System.out.println(ch2);
		System.out.println(bo1);
		System.out.println(s1);
	}
}
```

Test case #1: Execute above application by changing readXxx() methods calling order. Then check whether you will get right values or not.

Test case #2: Execute above application by calling different readXxx() methods. Then check whether you will get right values or not.

Actually readXxx() method read number of bytes from file based on the data type. For example if we call readInt() method it reads 4 bytes from file. So if four bytes are available in file program executed fine, else program execution terminates with EOFException.

Limitation of DIS and DOS

Using DOS and DIS classes we cannot read and write objects from persistence media. They have only capability to read data up to only primitive data types.

Solution

To read and write objects we must use ObjectInputStream and ObjectOutputStream.

ObjectInputStream and **ObjectOutputStream**

These two classes are used to store object's state permanently in files or to send to remote computer via network.

ObjectOutputStream class is a subclass of ObjectOutput interface. It implements below method to write object to underline output stream

```
public void writeObject(Object obj) throws IOException
```

ObjectInputStream class is a subclass of ObjectInput interface. It implements below method to read object from underline input stream

```
public Object readObject()throws IOException
```

Rule: To write or send object to external world it must be of type **java.io.Serializable** interface. It means this object's class must be a subclass of java.io.Serializable interface. Else writeObject() method throws **java.io.NotSerializableException.**

Constructors to create above classes objects

Like DIS and DOS, OIS and OOS are also cannot connect to source and destination directly. So their constructors take other input and output stream class objects.

```
public ObjectInputStream(InputStream in)
public ObjectOutputStream(OutputStream out)
```

Applcation #6: Below application shows writing Bank class object to file

```
//Bank.java
public class Bank implements java.io.Serializable
{
	static double minBalance = 5000;

	private long accNo;
	private String accHName;
	private String username;
	private transient String password;
	private transient double balance;

	public void setAccNo(long accNo)
	{
		this.accNo = accNo;
	}
	public long getAccNo()
	{
		return accNo;
	}
	public void setAccHName(String accHName)
	{
		this.accHName = accHName;
	}
	public String getAccHName()
	{
		return accHName;
	}

	public void setUsername(String username)
	{
		this.username = username;
	}
	public String getUsername()
	{
		return username;
	}
```

```
    public void setPassword(String password)
    {
        this.password = password;
    }
    public String getPassword()
    {
        return password;
    }
    public void setBalance(double amt)
    {
        this.balance = this.balance + amt;
    }
    public double getBalance()
    {
        return balance;
    }

    public String toString()
    {
        return "accNo: "+accNo +"\n" +
                    "accHName: "+accHName +"\n" +
                    "username: "+username+"\n" +
                    "password: "+password+"\n"+
                    "balance: "+balance+"\n"+
                    "minBalance: "+minBalance+"\n";
    }
}
```

```
// BankTransaction.java  -> shows object writing
import java.io.*;

class BankTransaction {
	public static void main(String[] args) throws Exception
	{
		//creating bank object
		Bank acc1 = new Bank();

		//setting bank object state
		acc1.setAccNo(1);
		acc1.setAccHName("Hari");
		acc1.setUsername("Hari Krishna");
		acc1.setPassword("Naresh technologies");
		acc1.setBalance(999999999);

		//printing bank object state
		System.out.println(acc1);

		//creating OOS object
		ObjectOutputStream oos = new ObjectOutputStream(
				new FileOutputStream("BankAccountsinfo.ser"));

		//writing bank object state to file
		oos.writeObject(acc1);

		System.out.println("Object written to file");
	}
}
```

```
// BankUser.java -> shows object reading
import java.io.*;

class BankUser
{
	public static void main(String[] args) throws Exception
	{
		//creating OIS object
		ObjectInputStream ois = new ObjectInputStream(
				new FileInputStream("BankAccountsinfo.ser"));

		//casting returned object to Bank type
		Bank accDetails = (Bank) ois.readObject();

		//printing Bank object data
		System.out.println(accDetails.getAccHName() + " details");
		System.out.println(accDetails);
	}
}
```

Explain what is the actual functionality of OOS and OIS?
Performing Serialization and deserialization

Serialization
Serialization is the process of converting objects into stream of bytes and sending them to underlying OutputStream. Using Serialization we can store object state permanently in a destination, for example file or remote computer.

Serialization operation is performed by writeObject () method of ObjectOutputStream.

Deserialization
Deserialization is the process of converting stream of bytes into original object. Deserialization operation is performed by readObject () of ObjectInputStream.

Rule on Serialization

Only java.io.Serializable type objects are serialized. Serializable is a marker interface, it doesn't have any methods. It provides special permission or identity to JVM to serialize object.

If the given object isn't of type Serializable interface then writeObject() will throw an unchecked exception called java.io.NotSerializableException.

The below diagram explains serialization and deserialization operations.

How can readObject() method load class of the serialized object?
It gets that serialized object's class name from serialized file.

Q) What is the functionality of writeObject and readObject methods?
In Serialization writeObject() method stores object state in the file with the below information.

1. Its class name
2. Current .class file's serialVersionUID
3. Its non-transient variable names and their data type
4. Those variables current modified values.

In deserialization readObject() method creates object with the state available in the file as shown below

1. readObject() method first reads class name from serialized file and loads that class by using Class.forName(). While loading the class static variables get memory with originally assigned values and SBs are executed.
2. Then it compares serialVersionUID of current loaded class with the serialVersionUID that is stored in serialized file
   a. If both are not same it terminates deserialization process by throwing exception
   java.io.InvalidClassException
   b. If both values are same, it creates object with the current loaded class non-static variables without using new keyword and constructor.
3. Then readObject method populates values
   a. from serialized file for the non-transient variables and
   b. default values for transient variables and also for the non-static variables those are not avaiable in serialized file.

Rules in deserialization:

" In deserialization the object class's ".class" file must be available, else readObject() method throws java.lang.ClassNotFoundException.

" In deserialization we must use the same version of .class file that is used in serialization, else object is not deserialized.

Let us understand Serialization with *IS-A* and *HAS-A* relations

Serialization with IS-A relation

If super class is deriving from Serializable interface, then all its subclasses are also serializable types. They no need to be sub classing from Serializable interface again explicitly.

In this case, if we serialize subclass object, super class all non-transient instance variables are also serialized. In deserialization all subclass and super class instance variable's values are populated from serialized file.

For Example:

```
//A.java
class A implements java.io.Serializable{
        int x;

        A(){
                x = 50;
                System.out.println("x is initialized with 50");
        }
}
//B.java
class B extends A{
        int y;

        B(){
                y = 60;
                System.out.println("y is initialized with 60");
        }

        public String toString(){
                return "x: "+x + "\ny: "+y;
        }
}
```

```
//OOSDemo.java
import java.io.*;

public class OOSDemo{
        public static void main(String[] args)throws Exception{
                OOS oos = new OOS(new FOS("test.ser"));
```

```
            B b = new B();
            b.x = 70;
            b.y = 80;

            System.out.println("x,y values are changed to 70, 80");

            oos.writeObject(b);
            System.out.println("B object is written to file");
        }
    }
```

Compilation and execution

```
>javac A.java
>javac B.java
>javac OOSDemo.java
>java OOSDemo
    x variable is initialized with 50
    y variable is initialized with 60
    x, y variables are changed to 70,80
    B object is written to file
```

```
//OISDemo.java
import java.io.*;

public class OISDemo{
    public static void main(String[] args)throws Exception{
        OIS ois = new OIS(new FIS("test.ser"));
        Object obj = ois.readObject();
        System.out.println(obj);
    }
}
```

```
>javac OISDemo.java
>java OISDemo
    x: 70
    y: 80
```

As you can observe in the above output, in deserialization B class object is populated with modified x and y variables values.

Q) If we remove "implements java.io.Serializable" from A class declaration, will B object be written to file?
A) No, it leads to exception java.io.NotSerializableException

Q) If we remove "implements java.io.Serializable" from A class declaration, and we added to B class declaration, will B object be written to file?
A) Yes, but only B class non-transient variable's values are stored in serialized file.

In deserialization, super class A's instance variable x is initialized by constructor.
So, if we run OISDemo with above change the output will as shown below

```
> java OISDemo
        x variable is initialized with 50
        x: 50;
        y: 80;
```

FAQ: Is super class must be Serializable to serialize subclass object?
A) No.

Serialization with HAS-A relationship:
Q) To serialize an object, will its internal object also must be Serializable?
A) Yes, else object will not be serialized

For Example:

```
class Student implements java.io.Serializable{
        String name = "Hari Krishna, Naresh Technologies";
        Address add = new Address();
}
class Address{
        String city = "Hyd";
}

------
        Student s = new Student();
        oos.writeObject(s);
-----
```

In the above case, writeObject() method throws java.io.NotSerializableException, because Address is not sub classing from Serializable interface.

Solutions

There are three solutions for the above problem

1. Declare "add" as transient
   In this case, deserialized Student object's add variable will have value null, so Student will not have address (*not good*)
2. Create subclass of Address and use this subclass object
   In this case, deserialized Student object will have Address with null value, but it is not with current Student address value.
3. Customize serialization with below private methods
   private void writeObject(ObjectOutputStream oos)throws IOException
   private void readObject(ObjectInputStream ois)throws IOException

Customizing Serialization

If we want to write Student object with its current address object values we must customize serialization by using below two private methods

```
private void writeObject(ObjectOutputStream oos)throws IOException{
        // here you write address object values to file
        //using writeXxx(xxx) methods
}

private void readObject(ObjectInputStream ois)throws IOException{
        // here you read address object values from file
        // using readXxx() methods and store those values
        // in newly created Address object, and set this
        // Address object to Student's Address referenced variable "add"
}
```

Below is the changed Student class to write address object state

```
class Student implements java.io.Serializable {
    int rollNum;
    String sname;
    transient Address add;
    Student(int rollNum, String sname, Address add){
        this.rollNum    =    rollNum;
        this.sname      =    sname;
        this.add        =    add;
    }
```

```
    private void writeObject(ObjectOutputStream oos){
        try{
            oos.defaultWriteObject();

            //writing current Student's Address
            oos.writeInt(add.hno);
            oos.writeUTF(add.city);
        }
        catch(IOException e){
            e.printStackTrace();
        }
    }
    private void readObject(ObjectInputStream ois){
        try{
            ois.defaultReadObject();

            //reading current Student's Address state
            int    hno = ois.readInt();
            String city = ois.readUTF();

            add = new Address(hno, city);
        }
        catch(IOException e){
            e.printStackTrace();
        }
        catch(ClassNotFoundException e){
            e.printStackTrace();
        }
    }
    public String toString(){
        return      "rollNum:"  + rollNum              + "\n" +
                    " sname:    "      + sname         + "\n" +
                    " hno:"     + add.hno    + "\n" +
                    " city: "           + add.city;
    }
}
```

```
class Address{
        int hno;
        String city;

        Address(int hno, String city){
                this.hno = hno;
                this.city = city;
        }
}
```

Functionality of private writeObject and readObject methods:

- These two methods are automatically called by JVM
  - *private writeObject* method is called from *oos.writeObject()* method by passing current *OOS object* reference as argument. So that in this private writeObject method we can write data to the same underlying file (serialized file).
  - *private readObject* method is called from *ois.readObject()* method by passing current *OIS object* reference as argument. So that in this private readObject method we can read data from the same underlying file (serialized file).
- Inside these two methods we must call "*oos.defaultWriteObject()*"; and "*ois.defaultReadObject()*" methods to perform serialization and deserialization on the given object. Otherwise object's data written to the file is not read instead we get default values of that object's non-static variables.
- next to these two methods call we must call writeXxx() and readXxx() methods to write and read non-Serializable object data to the same file individually

Q) What is a marker interface?

An interface that is used to check / mark / tag the given object is of a specific type to perform a special operations on that object, is called marker interface. Maker interface will not have methods, they are empty interfaces.

Q) What is the need of marker interface?

It is used to check, whether we can perform some special operations on the passed object.

For example,
If class is deriving from java.io.Serializable, then that class object can be Serializable, else it cannot.If class is deriving from java.lang.Cloneable, then that class object can be cloned, else it cannot.

In side those particular methods, that method developer will check that the passed object is of type the given interface or not by using instanceof operator.

For example, in writeObject() method we will find below code

```
public class ObjectOutputStream extends OuputStream{
        public void writeObject(Object obj){
                if (obj instance of Serializable){
                                ------------
                                ------------
                                ------------
                }
                else{
                        throw new
                        NotSerializableException(obj.getClass().getName());
                }
        }
}
```

Q) Can we write custom marker interfaces?
A) Yes. Write an empty interface, and use in the method to check the passed object is of type that interface or not by using "instanceof" operator.

Q) Is marker interface an empty interface?Yes
Q) Is empty interface a marker interface?
Depends, if it is used in instanceof operator condition then it called marker else it is called empty interface.

List of know predefined marker interfaces

1. java.lang.Cloneable
2. java.io.Serializable
3. java.util.RandomAccess
4. java.rmi.Remote
5. javax.servlet.SingleThreadModel
6. java.util.EventListener

SequenceInputStream

This class is used to read data from multiple input streams in sequence from the first one until end of file is reached, whereupon it reads from the second one, and so on, until end of file is reached on the last of the contained input streams.

Basically it represents the logical concatenation of other input streams.
It has below two constructors to create its object

1. public SequenceInputStream(InputStream s1, InputStream s2)
   This constructor allows us to create SequenceInputStream class object only with two InputStreams.

2. public SequenceInputStream(Enumeration<? extends InputStream> e)
   This constructor allows us to create SequenceInputStream class object with 'n' number of InputStreams.

Below applications shows reading data from two InputStreams (two files)

```
//SequenceBufferdIOSDemo.java
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.SequenceInputStream;

/**
 * Demonstrates BufferedInputStream/BufferedOutputStream/SequenceInputStream.
 */
public class SequenceBufferdIOSDemo {
        public static void main(String args[]) throws IOException {

                FileInputStream f1 = new FileInputStream("abc.txt");
                FileInputStream f2 = new FileInputStream("bbc.txt");

                SequenceInputStream sis = new SequenceInputStream(f1, f2);
                int data;

                while( ( data=sis.read() ) != -1){
                        System.out.println((char)data);
                }
                bos.flush();
        }// End main()
}// End class
```

Q) Write a program to read data from three files by using SIS
Clue: In this application development we must use second constructor.

PrintStream class
This class is a filter output stream class used to write data *as it is in the given format*.
That means unlike *FileOutputStream*, this class writes 97 as 97 rather as 'a'.

It is the most convenient class in writing data to another output stream, because of below three reasons

1. Unlike other output stream's *write()* method, this class *write()* methods does not throw IOException
2. Calling *flush()* method is optional, it is called automatically.
3. In addition to default *write()* methods it defined new methods called *print* and *println*. These two methods are overloaded to print all types of java data type values including object *as it is in those format*. Below are the overloaded print and println methods list

```
public void print(int x);
public void print(long x);

public void print(float x);
public void print(double x);

public void print(char x);
public void print(boolean x);

public void print(char[] x);
public void print(String x);

public void print(Object x);
```

```
public void println();

public void println(int x);
public void println(long x);

public void println(float x);
public void println(double x);

public void println(char x);
public void println(boolean x);

public void println(char[] x);
public void println(String x);

public void println(Object x);
```

Note:
PrintStream class object is created in System class using a static variable called **out.** So we can access above methods from our class using the System class's PrintStream object as shown below.

```
System.out.print("abc");      System.out.print(30);
System.out.println("bbc");    System.out.println(40);
```

In the above first two statements we called *String* parameter print and println methods, and in the second two statements we called *int* parameter print and println methods

Q) How these methods are printing the given data *as it is in* the given format?
A) In side these methods the given data is converted into *String* data by calling "*String.valueOf*()" method, then the returned string data is printing on destination.

What is the difference between print and println methods?

print method places cursor in the same line after printing data, so that next coming output will be printed in the same line. But **println method** places cursor in the next line after printing data so that next coming output will be printed in the next line.

What is the output from below program?

```
class Test {
        public static void main(String[] args) {
                System.out.print("A");
                System.out.println("B");
                System.out.println("C");
        }
}
```

O/P
====

Rule #1: We cannot call print method without passing arguments because we do not have no-arg print() method it leads to *CE: cannot find symbol*. But we can call println method without passing arguments because we have no-arg println() method, it prints new line.

What is the output from below program?

```
class Test {
        public static void main(String[] args) {
                System.out.print("A");
                System.out.println();
                System.out.print("B");
                System.out.print();
                System.out.print("C");
        }
}
```

O/P
====

Rule #2: We cannot call print() or println() methods by passing ***null*** literal directly. It leads to CE: Ambiguous error. Because two methods are overloaded with String and Char[] parameters. Since these two parameters are siblings it leads to above compile time error. But we can pass null via referenced variable, then that referenced variable type parameter method is invoked. Object, String parameter methods prints null *where as* Char[] parameter method throws NPE.

What is the output from below program?

```
class Test {
        public static void main(String[] args){
                System.out.print(null);
                System.out.println(null);
        }
}
```

```
class Test {
        public static void main(String[] args) {
                String s = null;
                System.out.print(s);

                char[] ch = null;
                System.out.println(ch);
        }
}
```

What is the difference between println(Object) and writeObject(Object) methods?
writeObject(Object) method serializes the given object and stores its state in underlying output stream. But where as println(Object) method does not perform serialization instead it prints the passed object information that is returned by its *toString* method.

It means println(Object) method internally calls *toString()* on given object to print the given object information. If toString method is not defined in the passed object's class, then it is called from java.lang.Object, it is originally defined in Object class. toString method in Object class returns current object's "classname@hashcode in hexadecimal string format".

What is the output from below programs?

Case 1: toString method is not overridden in Example

```
class Example{
        int x = 10, y = 20;
}


class Test {
        public static void main(String[] args) {
                Example e = new Example();
                System.out.print(e);
        }
}
```

Case 2: toString method is overridden in Example

```
class Example{
        int x = 10, y = 20;
        public String toString(){
                return "Example class object" ;
        }
}
class Test {
        public static void main(String[] args) {
                Example e = new Example();
                System.out.print(e);
        }
}
```

Case 3: toString method is overridden with Example class object state

```
class Example{
        int x = 10, y = 20;
        public String toString(){
                return x + " ... " + y;
        }
}
class Test {
        public static void main(String[] args) {
                Example e = new Example();
                System.out.print(e);
        }
}
```

Case 4: Printing String class object

```
class Example{
        int x = 10, y = 20;
        public String toString(){
                return x +" ... " +y;
        }
}
class Test {
        public static void main(String[] args) {
                Example e = new Example();
                System.out.print(e);

                String s = "abc";
                System.out.print(s);
        }
}
```

Can developer create PrintStream class object explicitly?
Yes, it has below constructors to create its object.

```
public PrintStream(OutputStream out)
public PrintStream(OutputStream out,  boolean autoFlush)

public PrintStream(String fileName)
        throws FileNotFoundException

public PrintStream(File file)
        throws FileNotFoundException
```

These two constructors are given in Java 5 version to pass file name directly without creating FileOutputStream class object explicitly.

How can we print data in file using PrintStream class methods?
Yes, create PrintStream class object connecting to that file by using above constructors

Jdk1.4 based PrintStream object creation to connect to file

```
PrintStream ps = new PrintStream(new FileOutStream("abc.txt"));
```

Jdk1.5 based PrintStream object creation to connect to file

```
PrintStream ps = new PrintStream("abc.txt");
```

Below application shows storing data in file using PrintStream class

```
import java.io.*;
class PrintStreamDemo
{
        public static void main(String[] args) throws FileNotFoundException
        {
                //PrintStream object creation connecting to abc.txt file
                PrintStream ps = new PrintStream("abc.txt");

                //storing data in abc.txt file using explicit PrintStream object
                ps.print("A");
                ps.println("B");
                ps.println("C");

                System.out.print("Data written to abc.txt file");

                //printing data on console using implicit PrintStream object
                System.out.print("A");
                System.out.println("B");
                System.out.println("C");
        }
}
```

Difference between print(int) and write(int) methods?

print or println methods print the given data as it is in either on console or on file, it does not convert given number into *bytes*. But write method converts given number into bytes and writes those *bytes* into file. Then file editor showing those bytes as its corresponding ASCII character. Check below program and output

```
import java.io.*;
class PrintStreamDemo {
        public static void main(String[] args) throws Exception{

                FileOutputStream fos = new FileOutputStream("abc.txt");
                PrintStream ps = new PrintStream("bbc.txt");

                //storing data in abc.txt file using FileOutputStream object
                fos.write(97);
                fos.write(98);
                fos.write(99);
                System.out.print("Data written to abc.txt file");

                //storing data in bbc.txt file using explicit PrintStream object
                ps.print(97);
                ps.print(98);
                ps.print(99);
                System.out.print("Data written to bbc.txt file");

                //printing data on console using implicit PrintStream object
                System.out.println(97);
                System.out.println(98);
                System.out.println(99);
        }
}
```

Written to abc.txt file in same line

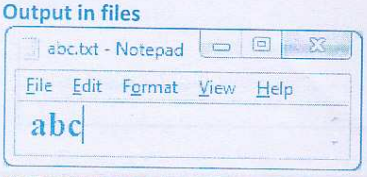
Printed in same line in bbc.txt file

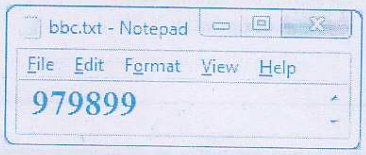
Printed in different lines on console

Output on console

```
d:\Nareshit\JavaHari\OCJP\IOStreams:javac PrintStreamDemo.java

d:\Nareshit\JavaHari\OCJP\IOStreams:java PrintStreamDemo
Data written to bbc.txt file
Data written to bbc.txt file
97
98
99

d:\Nareshit\JavaHari\OCJP\IOStreams:_
```

Output in files





Default Streams created in JVM

In JVM by default three streams are created by the class System to read data from keyboard and to write data to console (monitor). These stream objects are hold by static final referenced variables in System class. They are

- public static final InputStream **in**;
- public static final PrintStream **out**;
- public static final PrintStream **err**;

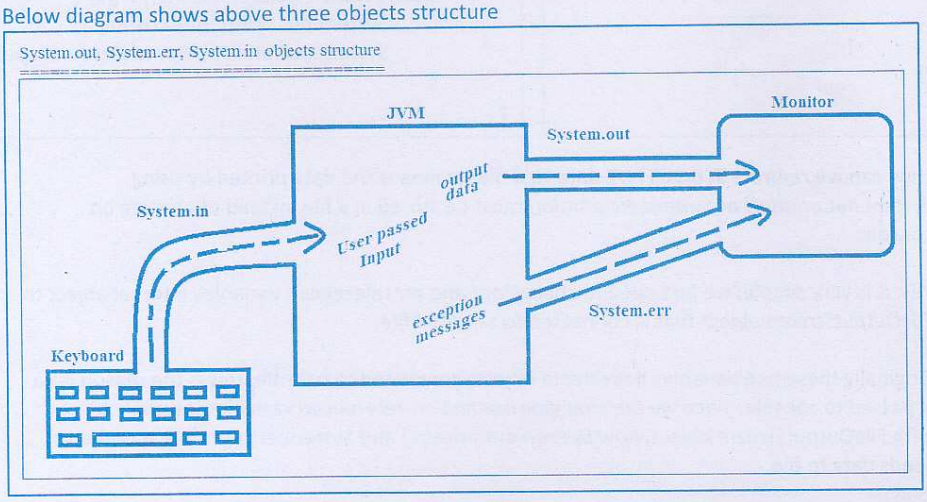
in variable holds BufferedInputStream class object that is connected to keyboard
out and **err** variables holds PrintStream class object that is connected to console.

Since these variables are defined as static variables we can access these objects hold by these variables from our class by using class name System like, **System.in**, **System.out**, **System.err**.

What is the difference between System.out and System.err?

JVM internally uses System.out to write output to console, and it uses System.err to write exceptions to console.

Below diagram shows above three objects structure



System.in object connects to keybord

Means to read data from keyboard we must use below statement

int data = System.in.read();

System.out and **System.err** objects connect to console.

Means to print data on console we must use below statements

System.out.println() or *System.err.println()*

Explain System.out.println() statement

- **System** is a predefined class defined in java.lang package
- **out** is a static referenced variable of type PrintStream class created in System class to hold PrintStream class object that is pointing to console.
- **println()** is a non static method defined in PrintStream class to print data.

So to call println() method from our class using PrintStream object created in System class we must use the statement **System.out.println()**;

Check below diagram, it shows the linking of all System, PrintStream and user class.

```
public class PrintStream
{
        public void print(String msg)
        {
                ----------
        }
        public void println(String msg)
        {
                ----------
        }
}
```

```
public class System{
        public static final InputStream in;
        public static final PrintStream out;
        public static final PrintStream err;
}
```

```
public class Test{
        public static void main(String[] args) {
                System.out.print("HariKrishna");
                System.out.println("Naresh Technologies");
        }
}
```

How can we redirect STDOUTPUT data to a file? It means the data printed by using *System.out.println()* or *System.err.println()* must be stored in a file instead of printing on console.

Yes, it is very simple; we just need to change *out* and *err* referenced variables internal object to *FileOutputStream* object that is connected to targeted file.

Originally these two variables has stream objects connected to console, this is the reason data is passed to console. Since we are changing *out* and *err* referenced variables internal objects with FileOutputStream object, now System.out.println() and System.err.println() statements sends data to file.

To set *out* and *err* referenced variables with new stream objects in System class below two methods are given

```
public static void setOut(PrintStream out)
public static void setErr(PrintStream err)
```

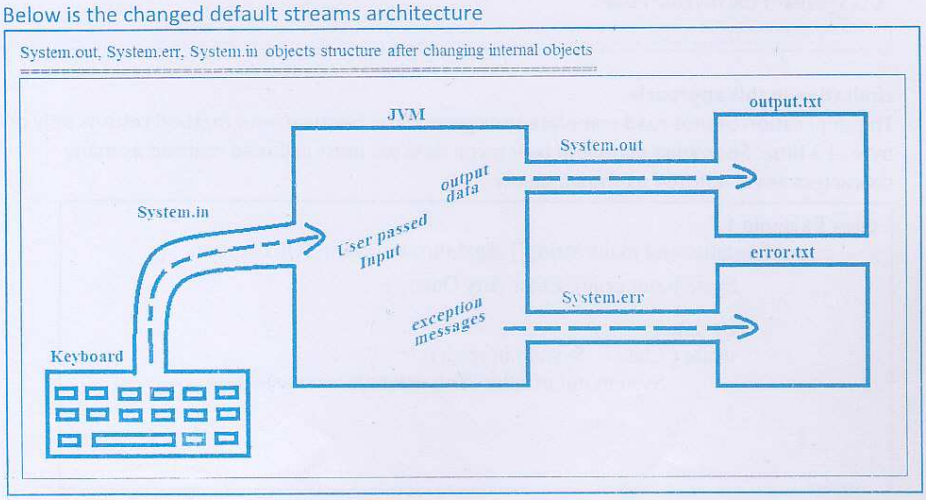
We also have below method to change *in* referenced variable internal object

```
public static void setIn(InputStream in)
```

Below program shows redirecting STDOUTPUT to a file.

```
import java.io.*;

class STDOutDemo
{
        public static void main(String[] args) throws Exception
        {
                System.out.println("Data before setOut() Method"); // printed on console
                System.err.println("Data before setErr() Method"); // printed on console

                //Creating PrintStream objects with targeted files
                PrintStream ps1 = new PrintStream(new FileOutputStream("output.txt"));
                PrintStream ps2 = new PrintStream(new FileOutputStream("error.txt"));

                //changing default streams
                System.setOut(ps1);
                System.setErr(ps2);

                System.out.println("Data after setOut() Method"); // passed to output.txt file
                System.err.println("Data after setErr() Method"); // passed error.txt
                int x = 10/0; //AE exception is passed error.txt
        }
}
```

Below is the changed default streams architecture



Reading Data from keyboard

To read data from keyboard we must use below statements in the program

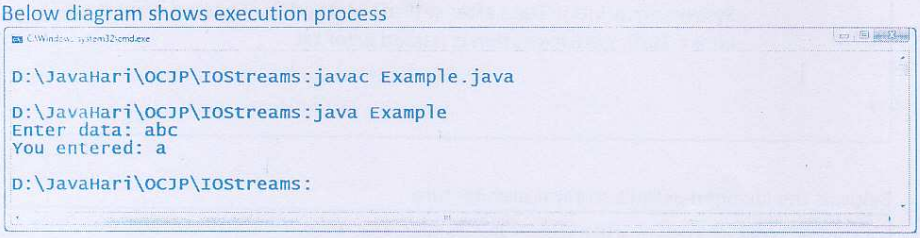
```
int i = System.in.read();
```

Below program shows reading data from keyboard using *System.in.read()*

```
class Example {
        public static void main(String[] args) throws java.io.IOException   {

                System.out.print("Enter data: ");

                int data = System.in.read();

                System.out.println("You entered: "+(char)data);
        }
}
```

Execution flow: when you execute this application, on console "Enter data:" message is printed and program execution is paused till user presses "Enter" key on keyboard. Then read method returns the user entered data and stores it in the variable *"data"*.

Below diagram shows execution process

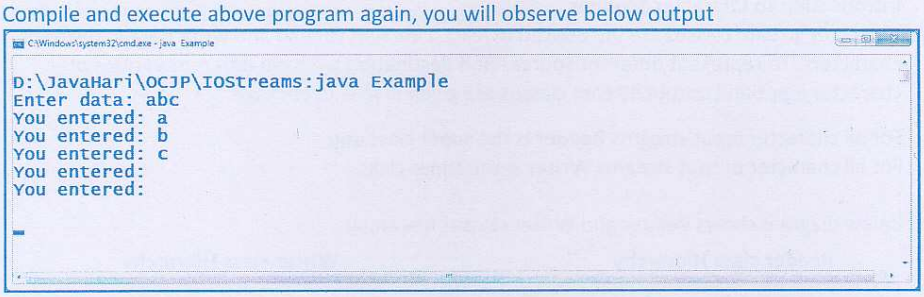


Limitation in this approach

This application cannot read complete user given data, because read method returns only one byte at a time. So to read complete user given data we must call read method as many characters as we entered as shown below

```
class Example {
        public static void main(String[] args) throws java.io.IOException   {
                System.out.print("Enter Any Data: ");

                int data ;
                while ( (data =  System.in.read()) != -1){
                        System.out.println("You entered: "+(char)data);
                }
        }
}
```

Compile and execute above program again, you will observe below output



The problem in this application is some extra unwanted characters are printed and more over application execution is not terminated.

When user presses "Enter" key on keyboard OS appends two more characters to the input data. These two characters are appended based on OS.

In Windows those two characters are **C**arriage **R**eturn (ASCII number 13) and **L**ine **F**eed (ASCII number 10).

So, above application prints user input and these special characters and still executing because while loop condition is not failed, means read() method never returns "-1".

To read only user given input characters, and further to terminate application we must prepare condition with value "**13**" as shown below.

```
class Example {
        public static void main(String[] args) throws java.io.IOException   {
                System.out.print("Enter Any Data: ");

                int data ;
                while ( (data =  System.in.read()) != 13){
                        System.out.println("You entered: "+(char)data);
                }
        }
}
```

This application only works in Windows OS, because other OS returns different value not 13.

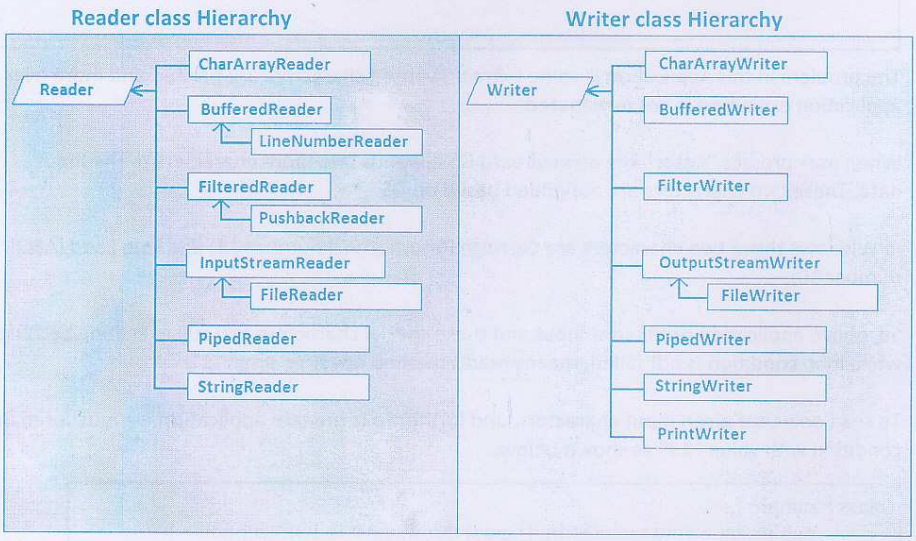
To solve these two problems in every java program developer must write manual validation code that is suitable to all OS. So, to solve this problem Sun has given a class called **BufferedReader** class as part of character streams. It has a method called **readLine** to return complete line as string at time.

Introduction to Character Streams

Character stream classes are introduced in jdk 1.1 version to read and write data in terms of characters. To represent different sources and destination we have different verities of character input and output stream classes are given in java.io package.

For all character input streams **Reader** is the super class and
For all character output streams **Writer** is the super class

Below diagram shows Reader and Writer classes hierarchy



Below are the Reader and Writer class methods

Reader class methods

```
public boolean ready() throws IOException;

public int read() throws IOException;
public int read(char[] ch) throws IOException;
public abstract int read(char[] ch, int off, int len)  throws IOException;

public long skip(long n) throws IOException;

public boolean markSupported();
public void mark(int readAheadLimit) throws IOException;
public void reset() throws IOException;

public abstract void close() throws IOException;
```

Writer class methods

```
==================
public void write(int i) throws IOException;
public void write(char[] ch) throws IOException;
public abstract void write(char[] ch, int off, int len) throws IOException;
public void write(String str) throws IOException;
public void write(String str, int off, int len) throws IOException;

public Writer append(char ch) throws IOException;
public Writer append(CharSequence seq) throws IOException;
public Writer append(CharSequence seq, int off, int len) throws IOException;

public Appendable append(char ch) throws IOException;
public Appendable append(CharSequence seq) throws IOException;
public Appendable append(CharSequence seq, int off, int len) throws IOException;

public abstract void flush() throws IOException;
public abstract void close()  throws IOException;
```

FileReader and FileWriter

These two classes are used to read and write data from a file as characters. It is recommended to use FileReader and FileWriter classes to read data from text file, where as FileInputStream and FileOutputStream classes are only recommended to use reading and writing image files.

Below are constructors available in these classes

```
FileReader(File f) throws FileNotFoundException
FileReader(FileDescriptor fd) throws FileNotFoundException
FileReader(String name) throws FileNotFoundException
```

```
FileWriter(File f) throws FileNotFoundException
FileWriter(FileDescriptor fd) throws FileNotFoundException
FileWriter(String name) throws FileNotFoundException

FileWriter(File f, boolean append) throws FileNotFoundException
FileWriter(String name, boolean append) throws FileNotFoundException
```

Below application shows reading data from a file "abc.txt" using FileReader

```
import java.io.*;

class FRDemo{
        public static void main(String[] args) throws FileNotFoundException,IOException{
                FileReader fr = new FileReader("abc.txt");

                int data;
                while ( (data = fr.read()) != -1)                {
                        System.out.print(data +" ... ");
                        System.out.println((char)data);
                }
        }
}
```

Limitation with FileReader

Using FileReader class we cannot read one line at time from the file. We can only read one character at time. It reduces the reading performance of the application.

To solve this problem and to read one line at a time we must use BufferedReader class.

BufferedReader and BufferedWriter

These two classes are used to improve reading and writing capability of other input and output stream.

Below are the constructors available in these classes to create its objects

```
public BufferedReader(Reader in)
public BufferedReader(Reader in, int size)

public BufferedWriter(Writer out)
public BufferedWriter(Writer out, int size)
```

BufferedReader class has below method to read character data one line at time.

```
public String readLine() throws IOException
```

Below statements shows creating BufferedReader class object for reading one line at time from the file "abc.txt"

```
BufferedReader br = new BufferedReader(new FileReader("abc.txt"));
```

We should call readLine() method as shown below to read one line

```
String line     = br.readLine();
```

It returns only first line, to read all line of text we must call it in while loop. It returns null if cursor reaches end of the file.

Below program shows reading file data one line at a time.

```
import java.io.*;
class BRFRDemo{
        public static void main(String[] args) throws FileNotFoundException,IOException{
                BufferedReader br = new BufferedReader(new FileReader("abc.txt"));

                String line;
                while ( (line = br.readLine()) != null){
                        System.out.println(line);
                }
        }
}
```

How can we read data from keyboard one line at a time?
It is only possible by using BufferedReader class object as this class has *readLine()* method.

We must create *BufferedReader* class object in connection with *System.in* as shown below
BufferedReader br = new BufferedReader(System.in); (***Wrong***)

But this statement leads to CE because BufferedReader class does not have InputStream parameter constructor. It has Reader parameter constructor.

So we need a Reader subclass that has InputStream parameter constructor. That class is *InputStreamReader*.

InputStreamReader and **OutputStreamWriter**
An InputStreamReader is a bridge from byte streams to character streams: It reads bytes and decodes them into characters using a specified charset.

It has below constructor to take InputStream type object

```
public InputStreamReader(InputStream in)
```

An OutputStreamWriter is a bridge from character streams to byte streams: Characters written to it are encoded into bytes using a specified charset.

It has below constructor to take OutputStream type object

```
public OutputStreamWriter(OutputStream out)
```

Now let us try to create BufferedReader object connecting with keyboard

- pass *System.in* as argument to *InputStreamReader* object creation
- pass *InputStreamReader* object as argument to *BufferedReader* object creation

Below is the complete object creation statement

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

Below application shows reading data from keyboard

```
import java.io.*;
class KeyboardReader{
        public static void main(String[] args) throws FileNotFoundException,IOException{
                BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

                System.out.print("Enter data:");

                String line = br.readLine();
                System.out.println("You entered: "+line);
        }
}
```

Write a program to add two numbers by reading its values from keyboard?

```
import java.io.*;
public class Addition {
        public static void main(String[] args) {

                try{
                        BufferedReader br = new BufferedReader(new
                                                        InputStreamReader(System.in));

                        System.out.print("Enter first integer value: ");
                        String fno = br.readLine();

                        System.out.print("Enter second integer value: ");
                        String sno = br.readLine();

                        //converring string form to int form
                        int i = Integer.parseInt(fno);
                        int j = Integer.parseInt(sno);

                        int k = i + j;

                        System.out.println("Result: "+k);
                }
                catch(NumberFormatException nfe){
                        System.out.println("Please pass only integer numbers");
                }
                catch(IOException ioe){
                        ioe.printStackTrace();
                }
        }
}
```

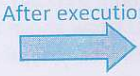
IOStreams real-time project based code

In project development sometimes it is necessary to change a number or a word in a file after our project installation. For example port number of tomcat server in server.xml file.

Below example shows replacing "xyz" with "abc" and storing changing content in the same file.

abc.txt

```
abc bbc xyz bbc abc
bbc xyz bbc xyz bbc
cbc bbc xyz bbc cbc
bbc xyz cbc xyz cbc
```



abc.txt

```
abc bbc abc bbc abc
bbc abc bbc abc bbc
cbc bbc abc bbc cbc
bbc abc cbc abc cbc
```

```
import java.io.*;

public class IOStreamProject {
        public void changeData()throws IOException{
                BufferedReader br = new BufferedReader(new FileReader("abc.txt"));

                //StringBuilder object creation to store file data
                StringBuilder fileDataBuffer = new StringBuilder();

                //reading complete file data and storing it in StringBuilder object
                while(br.ready()) //ready() method checks is stream has data to read or not
                {
                        fileDataBuffer.append( br.readLine() + "\n" );
                }

                //changing StringBuilder to String to call replace() method
                String fileData = fileDataBuffer.toString();

                //replacing "xyz" with "abc"
                fileData = fileData.replace("xyz","abc");

                //writting changed content to file
                FileWriter fw = new FileWriter("abc.txt"));
                fw.write(fileData);

                fw.flush();

                System.out.println("Data changed successfully");

                br.close();
                fw.close();
        }
        public static void main(String[] args) throws IOException  {
                IOStreamProject p = new IOStreamProject();
                p.changeData();
        }
}
```

How FileInputStream and FileOutputStream classes can check whether given file is existed or not? Also how FileOutputStream class can create file if the given file is not existed?
These two classes internally use **File** class.

File class
This class is used to represent files and directory paths (but not data of the file, file data is represented by FileInputStream and FileOutputStream). Basically this class is used to create, delete, rename files and directories and also used to know above the files and directory information like, given file name represents file or directory, type of the file read-only or writable, last modified, etc...

Instances of the File class are immutable; that is, once created, the abstract pathname represented by a File object will never change, because we do not have setter method to change the file name in this File object. But we have getter method to get file name.

File class *constructors*
File class has below constructors to create its object.

public File(String pathname)
Creates a new File instance with the given file name.

Ex:
```
File f = new File("abc.txt"); <- abc.txt file is used from the current working directory
```

public File(String parent, String child)
Creates a new File instance with the given parent file and child path.

Ex:
```
File f = new File("IOStreams", "abc.txt"); <- abc.txt file is used from IOStreams folder from
                                                the current working directory.
```

public File(File parent, String child)
Creates a new File instance with the given parent file and child path. To use this constructor parent file name must be passed as File class object.

Ex:
```
File f1 = new File("IOStreams");
File f2 = new File(f1, "abc.txt"); <- abc.txt file is used from IOStreams folder from the
                                      current working directory
```
Note: If we pass parent as null, abc.txt file is used from current working directory.

public File(URI uri)
Creates a new File instance by converting the given *file:* URI into an abstract pathname.

Ex:
```
File f1 = new File("abc.txt");
File f2 = new File(f1.toURI());
```

Rule: If we pass child argument as *null* all four constructors throws *NullPonterException*

Below program shows creating File class objects and printing those objects

```
import java.io.*;

class FileConstructors
{
        public static void main(String[] args)
        {
                File f1 = new File("abc.txt");
                File f2 = new File("test","abc.txt");
                File f = new File("test");
                File f3 = new File(f,"abc.txt");
                File f4 = new File(f1.toURI());

                System.out.println("f1: "+f1);
                System.out.println("f2: "+f2);
                System.out.println("f: "+f);
                System.out.println("f3: "+f3);
                System.out.println("f4: "+f4);
        }
}
```

Output

```
C:\Windows\system32\cmd.exe
D:\Nareshit\JavaHari\OCJP\IOStreams>javac FileConstructors.java

D:\Nareshit\JavaHari\OCJP\IOStreams>java FileConstructors
f1: abc.txt
f2: test\abc.txt
f:  test
f3: test\abc.txt
f4: D:\Nareshit\JavaHari\OCJP\IOStreams\abc.txt

D:\Nareshit\JavaHari\OCJP\IOStreams>
```

What happened when we create File **class object? Will JVM check whether a file** or **directory existed with given name? When executing above program if the file** abc.txt **and the folder** test **are not existed will JVM throws any exception else it creates files with given names?**
When File class object is created, JVM just creates File class object with the given file name as that object state. It will not check for file or directory with the given name.

While executing the above program, the file *abc.txt* and folder *test* are not existed in the current working directory, even though there is no exception as you can see in the above cmd prompt window, and after this program execution check current working directory you will not find the file *abc.txt* or the directory *test* there.

Conclusion: File class object creation does not create files.

File class *methods*

File class has below methods to perform different operations on regular files and directories including creating, deleting and renaming files and directories

Use below method to find is there any regular file or directory existed with given name

- public boolean exists()
  It checks whether file or directory is existed with given name or not.
  Return *true*, if file is existed else returns *false*

Use below methods to create file or directory with given name.

- public boolean createNewFile() throws IOException
  It creates an empty normal file with the given name in given path.
  Returns *true,* if the file is created with given file name.
- public boolean mkdir() throws IOException
  It creates a directory with given name in the given path.
  Returns *true,* if the directory is created with the given file.
- public boolean mkdirs() throws IOException
  It creates a both parent and child directories with given names in the given path. It creates child directory inside parent directory.
  Returns *true,* if the directory is created with the given file.

Use below methods to know given file name denotes normal file or directory

- public boolean isFile()
  It returns *true,* if the given file name denoted a normal file.
- public boolean isDirectory()
  It returns *true,* if the given file name denoted a directory.

Use below methods to know basic file properties

- public boolean canRead();
  returns *true*, if file is readable
- public boolean canWrite();
  returns *true*, if file is writable
- public boolean isHidden();
  returns *true*, if file is hidden
- public boolean setReadOnly();
  Set file is only readable, after this method call we cannot write data to this file.
- public long lastModified();
  Returns last modified time in milliseconds.
- public long length()
  Returns the length of the file denoted by this abstract pathname. The return value is unspecified if this pathname denotes a directory.

Use below methods to get file name and its path

- public String getName();
  Returns the relative path of the file, means name of the file.
- public String getAbsolutePath();
  Returns the absolute path of the file, means complete path of the file including name, in String object format.
- public File getAbsoluteFile();
  Returns the absolute path of the file in File object format.
- public String getCanonicalPath() throws IOException;
  Returns absolute path in String object format with system depended drive name.
- public File getCanonicalFile() throws IOException;
  Returns absolute path in File object format with system depended drive name.
- public boolean isAbsolute();
  Returns true, if file object is created with absolute path

Use below method to rename file or directory

- public boolean renameTo(File f);
  Renames current file with given name.

Use below methods to delete file or directory that is represented by this file object

- public boolean delete();
  It deletes file immediately
- public void deleteOnExit();
  It deletes file after JVM terminates.

Use below methods to list files and subdirectories of a given directory

- public String[] list();
  It returns all file and subdirectory names as String objects using String array.
- public File[] listFiles();
  It returns all file and subdirectory names as File objects using File array.
- public String[] list(FilenameFilter filter);
  It returns all file and subdirectory names whose names are matched with given filter as String objects using String array.
- public File[] listFiles(FilenameFilter filter);
  It returns all file and subdirectory names whose names are matched with given filter as File objects using File array.

Use below method to print file object state

- public String toString();
  It returns File object state in String format.

Use below method to convert file normal path to URL and URI

- public java.net.URL toURL() throws java.net.MalformedURLException;
- public java.net.URI toURI();

Below applications shows using all above methods

```
import java.io.*;

public class NormalFileCreation
{
	public static void main(String[] args) throws IOException
	{
		File f= new File("xyz.txt"); //just File object is created, xyz.txt file is not existed

		if(!f.exists())
		{
			System.out.println("f.createNewFile():"+f.createNewFile());
			System.out.println("File is Created");
		}
		else
		{
			if(f.isFile())
			{
				System.out.println("in else block");
				System.out.println("File Object is a normal file");
				System.out.println("f.getName():"+f.getName());
				System.out.println("f.length():"+f.length());

				System.out.println("f.canRead():"+f.canRead());
				System.out.println("f.canWrite():"+f.canWrite());
				System.out.println("f.getAbsolutePath():"+f.getAbsolutePath());
				System.out.println("f.getPath():"+f.getPath());
				System.out.println("f.deleteOnExit()");
				f.deleteOnExit();
				System.out.println("f.setReadOnly():"+f.setReadOnly());

				System.out.println("f.canWrite():"+f.canWrite());

			}
		}
	}
}
```

Run this application twice

In first run if block is executed, and file is created. In second run else block is executed.

Now answer below question

How FileInputStream and **FileOutputStream classes know whether file is existed** or **not with the given name?** *By using java.io.File class API*

These two classes in their constructors first they create File class object with given name then they call exist() method on that file object. If exist() method returns *true* means file is existed then they start reading and writing operation. If exist() method returns *false* then FileInputStream throws FileNotFoundException, and FileOutputStream creates file with the given name by using createNewFile() method. Below is the sample code of FIS and FOS

```
public class FileInputStream extends InputStream {
        private File name;
        public FileInputStream(String name) throws FileNotFoundException{
                this(new File(name));
        }
        public FileInputStream(File name) throws FileNotFoundException {
                if (!name.exists()){
                        throw new FileNotFoundException(name + " not existed")
                }

                this.name = name;
        }
}
```

```
public class FileOutputStream extends OutputStream {
        private File name;

        public FileOutputStream(String name) throws FileNotFoundException{
                this(new File(name));
        }
        public FileOutputStream(File name) throws FileNotFoundException        {

                if (name.exists()){
                        if(name.isDirectory()){
                                throw new FileNotFoundException(name + " not existed");
                        }
                }
                else{
                        boolean created = name.createNewFile();
                        if( !created ){
                                throw new FileNotFoundException(name + " not existed");
                        }
                }

                this.name = name;

        }
}
```

What happened in JVM when below statements are executed?

File f = new File("abc.txt");

- Just File class object is created in JVM. The file abc.txt is not created if it is not present.

FileInputStream fis = new FileInputStream("abc.txt");
FileReader fr = new FileReader("abc.txt");

- If abc.txt file is existed FIS or FR stream object is created to read data from that file,
- If it is not exited FIS or FR throws FileNotFoundException

FileOutputStream fos = new FileOutputStream("abc.txt");
FileWriter fw = new FileWriter("abc.txt");

- If abc.txt file is existed FOS stream object is created to write data into that file,
- If it is not exited FOS creates empty file with name abc.txt and then stream object is created.

Conclusion:

File class is only responsible of file creation, deletion and changing its properties
FileInputStream, FileReader are responsible of file data means in reading data from file
FileOutputStream, FileWriter are also responsible of file data means in writing to file

Below program shows creating directory, subdirectory and file inside a directory

```
import java.io.*;

class FileParentDirectoryDemo {
        public static void main(String[] args) throws IOException{
                File f1 = new File("abc.txt");
                f1.createNewFile(); //abc.txt is created as normal file in current working
                                                                        directory
                File f2 = new File("bbc.txt");
                f1.mkdir(); //bbc.txt is created as directory in current working directory.
                                We can create directories with extensions.

                File f3 = new File("xyz");
                f3.mkdir(); //xyz is created as directory in current working directory.

                File f4 = new File(f3, "1.txt");
                f4.createNewFile(); //1.txt is created as normal file in xyz directory.

                File f5 = new File(f3, "abc");
                f5.mkdir(); //abc is created as directory in xyz directory.

                File f6 = new File("pqr", "stv");
                f6.mkdirs(); //pqr is created as directory in current working directory, and stv is
                                                                        created in pqr as subdirectory.
```

```
                System.out.println("f1: "+f1);
                System.out.println("f2: "+f2);
                System.out.println("f3: "+f3);
                System.out.println("f4: "+f4);
                System.out.println("f5: "+f5);
                System.out.println("f6: "+f6);
        }
}
```

O/P:

```
f1: abc.txt
f2: bbc.txt
f3: xyz
f4: xyz\1.txt
f5: abc.txt\abc
f6: pqr\stv
```

```
//List directory files
import java.io.File;
public class ListFiles{
        public static void listFiles(String file)  {
                listFiles (new File(file));
        }
        public static void listFiles(File dir){
                try{
                        if(dir != null){

                                File dirList[] = dir.listFiles(); //File class method not current method

                                if(dirList != null ){
                                        for (int i = 0 ; i < dirList.length ; i++ ){
                                                File file = dirList[i];

                                                if(file.isFile())   {
                                                        System.out.println(file + " is a file");
                                                }
                                                else{
                                                        System.out.println(file + " is a directory");
                                                        listFiles(file);
                                                }
                                        }
                                }
                        }
                        else{
                                System.out.println("directory is null");
                        }
                }
                catch(Exception e){
                        e.printStackTrace();
                }
        }
}

class Test{
        public static void main(String[] args)  {
                //incurrent directory create a directory test with files and sub folders, then pass it
                as argument to this method
                ListFiles.listFiles("test");
        }
}
```

```
// This program demonstrates FileNameFilter.
import java.io.*;

class FileExtention implements FilenameFilter
{
        String extFile;

        public FileExtention(String extFile)
        {
                this.extFile = "." + extFile;
        }

        public boolean accept(File dir, String name)
        {
                return name.endsWith(extFile);
        }
}

public class FilterListFiles
{
        public static void main(String args[])
        {
                String strDir = "../IOStreams";
                File f = new File(strDir);

                FilenameFilter onlyFile = new FileExtention("txt");

                String strFile[] = f.list(onlyFile);

                System.out.println("\nThe JAVA files in the current directory are:\n");
                for(int i=0; i<strFile.length; i++)
                {
                        System.out.println(strFile[i]);
                }
        }
}
```

```
/*
Below application shows deleting normal files and directories
        Before executing this application, create normal empty files "1.txt, 2.txt" and a
        directory "xyz with a file abc.txt"
*/
import java.io.*;

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

                File f1 = new File("1.txt");
                f1.delete();
                if(!f1.exists()){
                        System.out.println("1.txt is deleted");
                }
                else{
                        System.out.println("1.txt is not deleted");
                }

                File f2 = new File("2.txt");
                f2.deleteOnExit();
                if(!f2.exists()){
                        System.out.println("2.txt is deleted");
                }
                else{
                        System.out.println("2.txt is not deleted");
                }

                File f5 = new File("xyz");
                f5.delete();
                if(!f5.exists()){
                        System.out.println("xyz is deleted");
                }
                else{
                        System.out.println("xyz is not deleted");
                }
        }
}
```

```
O/P
===
1.txt is deleted
2.txt is not deleted
xyz is not deleted
```

After this application execution check current working directory; you will discover "2.txt" is also deleted. Because deleteOnExit() method deletes file after JVM terminates.

The directory "xyz" will not be deleted, because it is not an empty directories, delete() method deletes only empty directories.

```
//Deleting directory files
import java.io.File;
public class DirectoryDelete{
	public static void directoryDelete(String file){
		directoryDelete(new File(file));
	}
	public static void directoryDelete(File dir){
		try{
			if(dir != null){

				File dirList[] = dir.listFiles();

				if(dirList != null ){
					for (int i = 0 ; i < dirList.length ; i++ ){
						File file = dirList[i];

						if(file.isFile())  {
							file.delete();
						}
						else{
							directoryDelete(file);
						}
					}
				}
				dir.delete();
			}
			else{
				System.out.println("directory is null");
			}
		}
		catch(Exception e){
			e.printStackTrace();
		}
	}
}

class Test{
	public static void main(String[] args) {
		//incurrent directory create a directory test with files and sub folders, then pass it
		as argument to this method
		DirectoryDelete.directoryDelete("test");
	}
}
```

File object creation with complete file or directory path

Q) How can we create File object by using normal file or directory that is existed in another directory let us say "D:\examples\test\abc.txt"?

A) We must create File class object by passing complete file path as shown below

```
import java.io.*;

class FileSeperators {
        public static void main(String[] args) {
                File f = new File("D:\examples\test\abc.txt");
                System.out.println("f: "+f);
        }
}
```

Compile above program you will get below compile time error

FileSeperators.java:7: illegal escape character

"\" represents escape sequence character.

After "\" the only allowed characters are "

	Character	usage
➢ Back slash	-> \	" \\ "
➢ Double quote	-> "	" \" "
➢ Single quote	-> '	" \' "
➢ Space character	->	" \ "
➢ Tab character	-> t	" \t "
➢ New line character	-> n	" \n "
➢ Carriage return character	-> r	" \r "
➢ Form feed character	-> f	" \f "

So to pass complete path of the file or directory we must use "\\" as file separator as shown below

```
import java.io.*;

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

                File f = new File("D:\\examples\\test\\abc.txt");

                System.out.println("f: "+f);
        }
}
```

File separators

File separators are platform dependent. In windows we must use "\\", and in Linux or Solaris we must use "/". So when we are moving project from Windows to Linux or Solaris or vice versa in all Java files we must change file separators to platform specific separator. Since it is a manual task, it requires lot of testing so it leads lot of maintenance cost.

So to solve this problem we must have a way to retrieve file separators dynamically specific to current platform.

In File class we have been given below two variables to retrieve and place path separators dynamically specific to current platform, they are

```
public static final char separatorChar
public static final String separator
```

Rewrite above application to place file separators dynamically

```
import java.io.*;

class FileSeperators {
        public static void main(String[] args) {
                String fs       = File.seperator;

                File f = new File("D:"+ fs +"examples"+ fs +"test"+ fs +"abc.txt");

                System.out.println("f: "+f);
        }
}
```

O/P

After executing above application we will get output

In windows
D:\examples\test\abc.txt

In Linux or Solaris
D:/examples/test/abc.txt

Chapter 29

Networking

- In this chapter, You will learn
  - Networking basics
  - Basic networking terminology
  - Types of protocols
  - How a request is sent to targeted server
  - How a response is sent back to same client
  - Socket programming
  - Understanding URLs and URLConnection
  - Understanding retrieving IP address and Host name dynamically
- By the end of this chapter- you will be in a position to develop networking based applications.

Interview Questions

By the end of this chapter you answer all below interview questions

- What is a Network?
- Types of Networks?
- Networking terminology
  - Request and response
  - Server and server system
  - Client and Client system
  - IP Address and host name
  - Port number
  - Protocol and Types of Protocols
  - URL, URI
  - Socket and ServerSocket
- How user can send request to only targeted Server when that client computer is connected to multiple other servers?
- And in the same way how server can send response back to same client computer when it is connected to multiple other client computers?
- How can we perform network operations using Java Application?
- How can we read / Update a resource available in remote / server computer?
- How can we get IP Address or host name of a computer dynamically?

Networking

Definition
Networking is the process of connecting multiple remote or local computers together.

Types of Networks
In the world we have below three types of networks

1. LAN
2. MAN
3. WAN

LAN – Local Area Network, used to connect computers within the building

MAN – Metropolitan Area Network, used to connect computers within a city.

WAN – Wide Area Network, used to connect computers throughout world.

Networking Terminology

Request and Response:
An *input data* sending via network to an application that is running in a remote computer is called *request*.

The *output* coming out from that application back to this client program is called *response*.

Client and Client System:
An *application* that allows us to send request is called *client*, and the person who is inputting the required data is called *end-user*, and the person who pays money to develop that entire project is called *customer*.

Client System is a *computer* in which client programs are executed, and on which end-user works.

Server and Server System:
Server is a *set of programs* (software) that takes request via network, process that request, generates response and sends it back to client application.

Server System is also a *computer* that has capability to receive network calls and handovers those calls to appropriate server software those are installed in that computer.

What is an IP Address and host name?
IP Address, Internet Protocol Address, is an identification number of computer in the network.

Rule: In a network two computers should not contain same IP Address. In this case network will prompt an error message **"IP Address conflict"**.

Host Name is the alias name of the computer. As human being we cannot remember computers with numbers. So each computer IP Address is mapped with a string called hostname.

Note:
In projects it is not recommended to use IP Address in sending request, instead we should use its hostname.

In order to use host name in place of IP Address we should configure server system IP Address and hostname in our local system in *hosts* file

This file path is "C:\Windows\System32\drivers\etc"

The mappings placed in this file are called host entries.

- Each entry should be kept on an individual line.
- The IP address should be placed in the first column followed by the corresponding host name with at least one space gap.

Example:

192.168.2.19	NareshIT19
192.168.2.20	NareshIT20
192.168.2.30	NareshIT30

Q) How network can get IP Address of the host name used in the request?
A) From hosts file of the local computer.

The default IP address of a computer is ***127.0.0.1***, and its mapped host name is local host.

Range of the IP Address

0.0.0.0 to 255.255.255.255

This entire range is divided into five classes
Class A, Class B, Class C, Class D, Class E

Class C range IP Address is used to build LAN, the range starts with "192.168.-.-"
Ex: 192.168.2.14

Below are the DOS commands we use to know

IP Address of the computer – **ipconfig**
Hostname of the computer – **hostname**

Sometimes computer cannot be connected to network even though network cable is plugged. In this situation use below command to establish / reestablish the connection with the server system (with network) – "**ipconfig /renew**".

What is a port number?
Port number is an identification number of server software.
It is a 32-bit positive integer number, having range 0 to 65535.

In a computer how many servers can be installed?
N number of servers can be installed and all of them can be start at a time if they have different port number.

Rule: Two servers cannot have same port number (PORT), as they cannot be run at same time. It leads to ambiguous problem for server system in redirecting request.

Note:
We can install more than one server with same port number, but we cannot start / use them at same time. If we start more than one server with same port number or if we start same server again while it is running, we experience an exception java.net.BindException: Address already in use

Note:
The port numbers 0 to 1024 are already registered for public servers. Hence we cannot release new server software with the port number in this range.

What is Protocol?
Protocol is a set of instructions to send request and receive response via network.

Types of Protocols:
There are two types of protocols
1. TCP/IP – Transmission Control Protocol / Internet Protocol
2. UDP – User Datagram Protocol

TCP/IP is a connection-oriented and secured protocol, for every request we will get response.
UDP is a connectionless protocol, and it is non-secured protocol. Using this protocol we cannot guarantee data transfer.
Ex: Mobile and radio communications.

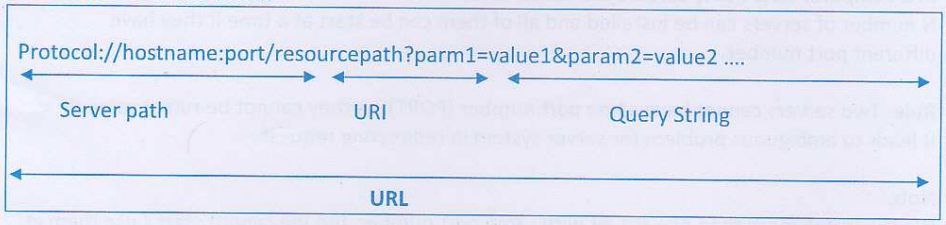
TCP/IP protocol is further divided into four protocols based on type of data we transfer
1. HTTP - Hyper Text Transfer Protocol
2. HTTPS - Hyper Text Transfer Protocol
3. FTP - File Transfer Protocol
4. SMTP - Simple Mail Transfer Protocol

What is a URL, URI?

URL is Uniform Resource Locator, which is an absolute path of remote file, used to locate that file in server.

It contains protocol, IP Address / host name of server system, Port number of the server, resource path and optionally its required input values (called query String).

URL format is as like below



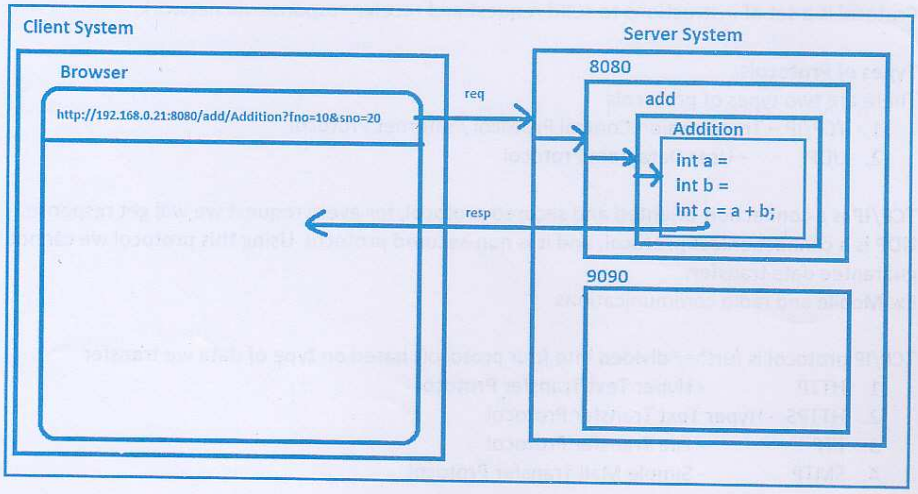
URI- is Uniform Resource Identifier is the path of the resource in server system.

URL – absolute path of remote file
URI – relative path of remote file

Ex:

The URL for sending request to below application is

http://192.168.0.21:8080/add/Addition?fno=10&sno=20

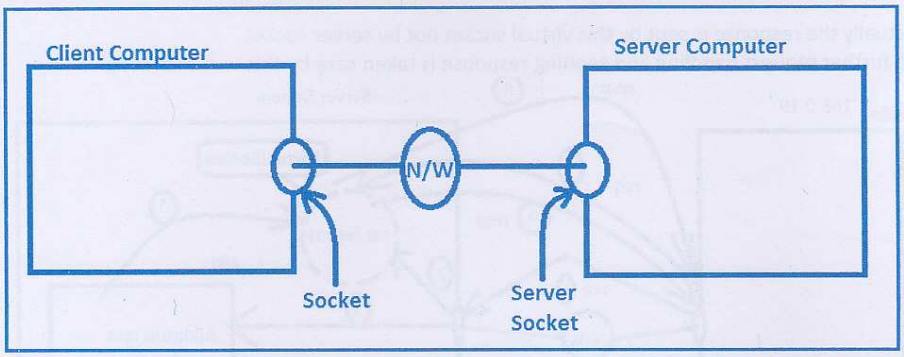


What is Socket, Server Socket?

Socket is a listener through which computer can receive requests and send responses. Using this listener actually computer connected to network and communicate to other computers.

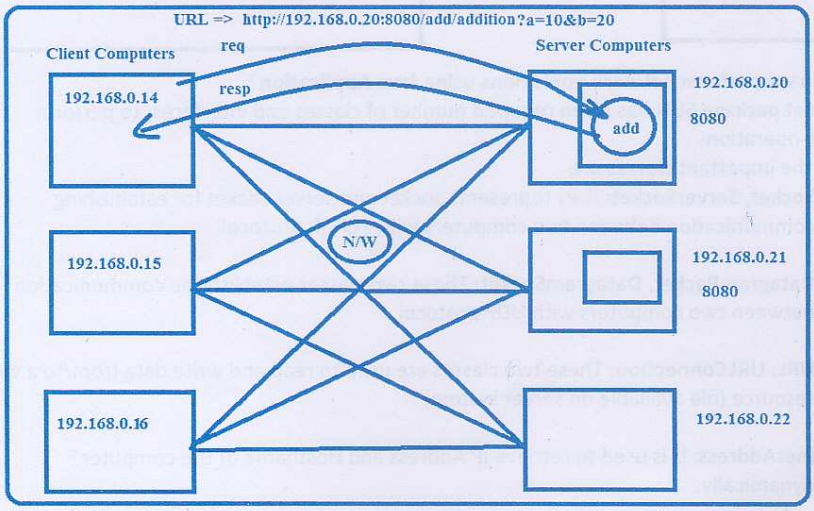
The listener of client system is called socket
The listener of server system is called server socket

It shows in below diagram



How user can send request from client to Server when client computer is connected with multiple servers?

It is possible due to URL format, which contains server IP Address / hostname, port of the server and resource path.

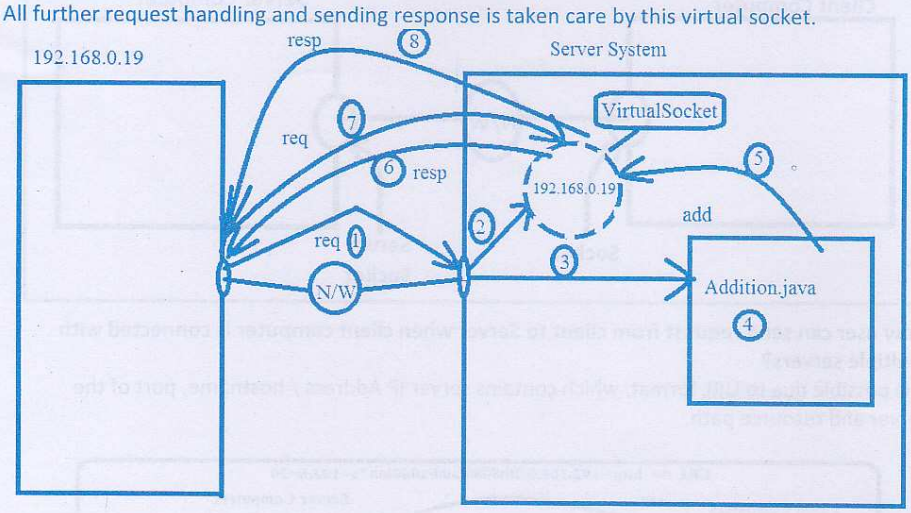


How server can send response back to same client when it is connected to multiple clients?
Due to the virtual socket created in server system for this socket.

When a request is sent from a client, in server system server socket creates a virtual socket (logical socket) for this client and stores this client information such as IP Address of the computer, client application details, etc...

Then server socket sends response back to same client program by using this virtual socket.

Actually the response is sent by this virtual socket not by server socket.
All further request handling and sending response is taken care by this virtual socket.



How can we perform network operations using Java Application?
In java.net package SUN has given required number of classes and interfaces, to perform network operations
Among the important classes are

1. **Socket, ServerSocket**: they represents socket and server socket for establishing communication between two computers with TCP/IP protocol

2. **DatagramPacket, DatagramSocket**: These two classes establish the communication between two computers with UDP protocol.

3. **URL, URLConnection:** These two classes are used to read and write data from/to a web resource (file available on server system)

4. **InetAddress**: It is used to retrieve IP Address and Hostname of the computer dynamically.

Networking programming

```
/*
 Client1.java
This program demonstrates printing a message on to the client, whenever client gets
connected to the server (One-way communication).
 */

import java.io.InputStream;
import java.io.DataInputStream;
import java.net.Socket;

class Client1
{
        public static void main(String args[])
        {
                try
                {
                        Socket s = new Socket("localhost", 4444);

                        InputStream     in = s.getInputStream();
                        DataInputStream dis1 = new DataInputStream(in);

                        String msg1 = dis1.readLine();
                        System.out.println("Server message is: " + msg1);
                        dis1.close();
                        s.close();
                }
                catch(Exception e)
                {
                        System.out.println("Exception: " + e);
                }
        }
}
```

```
/* Server1.java */

import java.io.OutputStream;
import java.io.DataOutputStream;
import java.net.ServerSocket;
import java.net.Socket;

class Server1 {
        public static void main(String args[]){
                try {
                        ServerSocket ss = new ServerSocket(4444);
                        System.out.println("Server is ready...");

                        Socket s = ss.accept();

                        System.out.println("Connection is accepted...");

                        System.out.println("Sent a message to the client...");
                        OutputStream out = s.getOutputStream();
                        DataOutputStream dos1 = new DataOutputStream(out);

                        dos1.writeBytes("Hello");

                        dos1.close();
                        s.close();
                        ss.close();
                }
                catch(Exception e)
                {
                        System.out.println("Exception: " + e);
                }
        }
}
```

Compilation:

Compile above two programs in any order.

Execution:

To execute we need two command prompts, because we are executing two applications at a time. To open new instance of cmd prompt with same folder path we must execute the dos command "start". In first cmd prompt execute server application, it waits for client application to be executed. Then execute client application.

```
>javac *.java
>start  >java Server1  >java Client1
```

```
/** This is an online chatting application. */

//ChatClient.java
import java.io.*;
import java.net.Socket;

public class ChatClient {
        public static void main(String[] args) {
                Socket s1;
                OutputStream os;
                InputStream is;
                DataOutputStream dos;
                DataInputStream dis;
                String sendMsg, receiveMsg;

                try
                {
                        s1      = new Socket("localhost",5555);
                        os      = s1.getOutputStream();
                        is      = s1.getInputStream();
                        dos = new DataOutputStream(os);
                        dis = new DataInputStream(is);

                        BufferedReader br = new BufferedReader(new
                                                InputStreamReader(System.in));

                        while(true)
                        {
                                sendMsg = br.readLine();
                                dos.writeUTF(sendMsg);
                                receiveMsg = (String) dis.readUTF();
                                System.out.println(receiveMsg);

                                if(receiveMsg.equals("bye")) {
                                        break;
                                }
                        }
                        dis.close();    dos.close();    os.close(); is.close();  s1.close();
                }
                catch(Exception e) {
                        e.printStackTrace();
                }
        }
}
```

```
//ChatServer.java
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;

public class ChatServer {
        public static void main(String[] args) {
                ServerSocket ss;
                Socket s1;
                OutputStream os;
                InputStream is;
                DataOutputStream dos;
                DataInputStream dis;
                String sendMsg, receiveMsg;

                try
                {
                        ss      = new ServerSocket(5555);
                        System.out.println("This is Online Chatting done on  Java");
                        s1      = ss.accept();
                        os      = s1.getOutputStream();
                        is      = s1.getInputStream();
                        dos     = new DataOutputStream(os);
                        dis = new DataInputStream(is);

                        BufferedReader br = new BufferedReader(new
                                                InputStreamReader(System.in));
                        while(true)
                        {
                                receiveMsg = (String) dis.readUTF();
                                System.out.println(receiveMsg);
                                if(receiveMsg.equals("bye")) {
                                        break;
                                }
                                sendMsg = br.readLine();
                                dos.writeUTF(sendMsg);
                        }
                        dis.close(); dos.close(); os.close(); is.close();  ss.close(); s1.close();
                }
                catch(Exception e){
                        e.printStackTrace();
                }
        }
}
```

```
/*
This program demonstrates displaying the server date on to the client,
using DatagramPacket and DatagramSocket classes.

UDPClient.java
*/

import java.net.DatagramPacket;
import java.net.DatagramSocket;

public class UDPClient
{
        public static void main(String args[])
        {
                try
                {
                        String strDate;
                        DatagramSocket ds     = new DatagramSocket(5555);
                        byte rdata[]                  = new byte[64];
                        DatagramPacket packate = new DatagramPacket(rdata,
                                                                rdata.length);

                        while(true)
                        {
                                ds.receive(packate);
                                strDate = new String(packate.getData());
                                System.out.println("Server Date and Time is: " +
                                                                strDate);
                        }
                }
                catch(Exception e)
                {
                        System.out.println("Exception: " + e);
                }
        }
}



//UDPServer.java
/*
```

```
 This program demonstrates displaying the server date on to the client,
 using DatagramPacket and DatagramSocket classes.
 */

import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Date;

public class UDPServer {
        public static void main(String args[]) {
                try
                {
                        Date date;
                        String strDate;
                        DatagramSocket ds = new DatagramSocket(4444);
                        int i = 0;

                        while(true)
                        {
                                date            = new Date();
                                strDate         = date.toString();
                                byte dbyte[]    = strDate.getBytes();

                                InetAddress address   =
                                                InetAddress.getByName("localhost");

                                DatagramPacket packate      = new
                                DatagramPacket(dbyte, dbyte.length, address, 5555);

                                ds.send(packate);
                                System.out.println((++i)+" packate sent");
                                Thread.sleep(1000);
                        }
                }
                catch(Exception e)
                {
                        System.out.println("Exception: " + e);
                }
        }
}
```

```
//URLDemo.java
/* This program demonstrates URL class.*/
import java.net.URL;
public class URLDemo {
        public static void main(String args[]) {
                try {
                        URL u = new URL("http://system1:1024/NetDemo2.java");

                        System.out.println("\nProtocol: " + u.getProtocol());
                        System.out.println("\nHost: " + u.getHost());
                        System.out.println("\nPort: " + u.getPort());
                        System.out.println("\nFile: " + u.getFile());
                        System.out.println("\nPath: " + u.toExternalForm());
                }
                catch(Exception e){
                        System.out.println("Exception: " + e);
                }
        }
}
//FileURLDemo.java
/*  This program demonstrates URL class.  */
import java.io.*;
import java.net.*;

public class FileURLDemo{
        public static void main(String args[]) {
                try {
                        File file       = new File("URLDemo.java");
                        String filePath = "file:///" + file.getAbsolutePath();
                        URL fileURL             = new URL(filePath);

                        InputStream in          = fileURL.openStream();
                        int data;

                        while((data = in.read()) != -1) {
                                System.out.print((char)data);
                        }
                        in.close();
                }
                catch(Exception e){
                        System.out.println("Exception: " + e);
                }
        }
}
```

```
//URLConnectionDemo.java
/* This program demonstrates URLConnection class. */

import java.io.*;
import java.net.URL;
import java.net.URLConnection;
import java.util.Date;
public class URLConnectionDemo {
        public static void main(String args[]) {
                try {
                        int c;
                        File file               = new File("Example.txt");
                        String filePath         = "file:///" + file.getAbsolutePath();
                        URL fileURL             = new URL(filePath);

                        URLConnection ucon  = fileURL.openConnection();

                        System.out.println("\nDate: " + new Date(ucon.getDate()));
                        System.out.println("Content-Type: " + ucon.getContentType());
                        System.out.println("Expires: " + ucon.getExpiration());
                        System.out.println("Last Modified: " + ucon.getLastModified());

                        int len = ucon.getContentLength();
                        System.out.println("Content Length: " + len + " bytes");

                        if(len > 0) {
                                System.out.println("\n= = = = = CONTENT = = = = = = ");
                                InputStream in = ucon.getInputStream();

                                int i = len;
                                while(((c = in.read()) != -1)) {
                                        System.out.print((char)c);
                                }
                                in.close();
                        }
                        else{
                                System.out.println("No content available.");
                        }
                }
                catch(Exception e)              {
                        System.out.println("Exception: " + e);
                }
        }
}
```

```
//InetAddressDemo.java
/* This program demonstrates InetAddress class. */

import java.net.InetAddress;

public class InetAddressDemo
{
        public static void main(String args[])
        {
                try
                {
                        InetAddress add = InetAddress.getLocalHost();
                        System.out.println("\nLocal Host Details      : " + add);
                        System.out.println("The Host IP Address is   : " +
                                                        add.getHostAddress());
                        System.out.println("The Host name is                 : " +
                                                                add.getHostName());

                }
                catch(Exception e)
                {
                        System.out.println("Exception: " + e);
                }
        }
}
```

Chapter 30

Collections and Generics

- In this chapter, You will learn
  - Definition and Need of collections
  - Problems of array object and its solution
  - Legacy collection classes
  - Collection Framework hierarchy
  - Types of collections
  - Advantage and disadvantage of every collection
  - Storing and retrieving elements from every collection
  - Use of Iterator, ListIterator, and Enumeration.
  - Differences between Iterator and Enumeration
  - Differences between Iterator and ListIterator
  - How collection uses hashCode() and equals() methods
  - Differences between Comparable and Comparator interfaces
  - Adding custom objects as keys to Map objects.
  - Need of utility classes – Collections, Array, BitSet
  - Reading properties from a properties file
  - Internationalization
  - Tokenizing string
  - Working with Date and TimeZone
  - Different timezones
  - Creating Timer and TimerTask (scheduling threads)
  - Real-time project development to show collection need with MVC and LC-RP architectures
- By the end of this chapter- you will understand collecting homogeneous and heterogeneous objects without size limitation.

i

Interview Questions

By the end of this chapter you answer all below interview questions

Collection prerequisites

- In how many ways we can store data in JVM?
- Memory location structure of variable, array object and class object.
- How can we pass single and multiple values and objects as arguments to methods
- Java Bean design pattern - DB and Java Object mapping
- What is effect upon primitive values and objects when they are passed as arguments and if they are modified using that method's parameter?
- Runtime polymorphism
- Use of toString(), equals() and hashCode() methods
- Generics
- Enhanced for-loop
- Autoboxing and unboxing

Introduction to collection framework

- Definition and need of collection
- Introduction to java.util package
- Why these classes are given when we have Object[] to group heterogeneous objects?
- Definition of framework, why this package classes are called Collections Framework?
- In how many formats we can group objects using collection? **Collection** and **Map**
- What are the benefits of collection classes?
- Legacy classes to group objects
- Introduction to Collection framework and its hierarchy.
- Common terminology used in this chapter
- Introduction to core collection interfaces and their inheritance relationship
- Collection
  - |<- List
  - |<- Set <- SortedSet <- NavigableSet (1.6)
  - |<- Queue (1.5)
- Map
  - |<- SortedMap <- NavigableMap (1.6)
- Introduction to **Collection interface class hierarchy**
- Situation force you to use Collection hierarchy classes.
- Similarities and Differences between ArrayList and Vector.
- Similarities and Differences between HashMap and Hashtable.
- Collection and its sub interface's methods.
- Different ways for retrieving elements from the Collection objects

- Introduction to Iterator, ListIterator, and Enumeration
- Their rules and methods usage
- Enhanced *for* loop (1.5)
- Similarities and Differences between Iterator and Enumeration.
- Similarities and differences between Iterator and ListIterator
- Understand the operations performed by addAll(), removeAll(), containsAll() and retainAll() methods in the Collection interface.
- Understanding equals() and hashCode() methods usage in collection objects List and HashSet, LinkedHashSet.
- Storing elements in sorting order using TreeSet
- Use of Comparator interface and difference between Comparable and Comparator interfaces.
- Inserting user defined objects in TreeSet using Custom Comparator
- Introduction to **Map interface class hierarchy**
- Three views of Map.
- Difference between Collection and Map
- Similarities and differences between HashMap and Hashtable.
- Map.Entry
- Storing entries with custom objects as keys
- Identify methods in the **Collections** class those can be used to produce immutable and thread-safe versions of various types of collections also method for searching, sorting, swapping elements in collection.
- How can you sort elements of unsorted collection like ArrayList?
- Identify methods in the **Arrays** class those can be used to perform searching and sorting operations of array elements.
- How can you print all elements of array without using for loop explicitly?
- Introduction to Properties class and the way of loading the properties into properties object from the file and storing the properties from the properties object to a file.
- Tokenizing the given string using **StringTokenizer**
- Implementing I18N application in JAVA using ResourceBundle and Locale
- Use of Currency class.
- Retrieving date and timestamp of the current computer using **Date** and **TimeStamp**.
- Introduction to ***Generics***, need of Generics and rules while using Generics.

Collections Framework and Generics

Definition and Need of Collection

Collection is a Java object that is used to group homogeneous & heterogeneous, duplicate & unique objects without size limitation for carrying multiple objects at a time from one application to another application among multiple layers of MVC architecture as method arguments and return type.

Introduction to java.util package

java.util package contains several classes to group / collect *homogeneous* and *heterogeneous* objects *without size limitation*. These classes are usually called collection framework classes.

Why collection classes are given when we have Object[] to group heterogeneous objects? What is the problem with Object[] to collect objects?

All array objects has below two problems

1. It allows us to store only same type of elements.
2. It is fixed in size

The first problem can be solved using java.lang.Object[]. Using Object[] array we can collect all types of objects. But the second problem cannot be solved automatically. We should develop below code to solve this problem

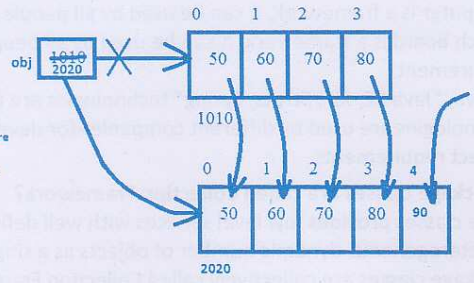
- Create array object with initial size and store elements
- Once array size is reached its capacity, execute below steps to store new element.
  - *Step #1:* Create another temporary array with required size
  - *Step #2:* Copy old array values to new array
  - *Step #3:* Add new element to new array at the end of all its elements
  - *Step #4:* Assign new array object reference to old array referenced variable

Check below program for more clarity

Program

```
Array object creation
Object[] obj = new Object[4];

Storing elements
obj[0] = "50";
obj[1] = "60";
obj[2] = "70";
obj[3] = "80";

Size reached its Capacity so further we can not store
any more elements,
1. So creating new array object with required size
   Object[] tempObj = new Object[10];
2. Copying old array elements to new array
   int oldArraySize = obj.length;
   int i = 0;
   for( ; i < oldArraySize; i++)
   {
       tempObj[i] = obj[i];
   }
3. storing new element at end of new array object
   tempObj[i] = "90";

4. Assinging new array object reference to old array
   referenced variable
   obj = tempObj;
```

Memory Location Structure



The above array object creation logic should be developed in every java project, which is treated as code redundancy because in addition to business logic development in every project and in every company developer must spent time to develop this repeated code. It leads to lot of code maintains problems and also takes more time to complete project development.

So, to solve all above problems SUN decided to implement collection classes common to every java project with high performance.

All collection classes are defined in java.util package.

Sun developed many collection classes among them 15 are important, they are

- LinkedList
- ArrayList
- Vector
- Stack
- PriorityQueue
- HashSet
- LinkedHashSet
- TreeSet
- HashMap
- LinkedHashMap
- TreeMap
- Hashtable
- Properties

In addition to collection classes SUN also defined some helper classes
They are

- Collections
- Arrays
- Random
- Scanner
- StringTockenizer
- Locale
- ResourceBundle
- Date
- Calendar
- GregorianCalendar

Why the name collection framework?

Let us first understand what is a framework?

Framework is a semi finished reusable application which provides some common low level services for solving reoccurring problems and that can be customized according to our requirement.

For example

- Computer is a framework, it can be used by all people according to their requirement
- Switch board is a framework, it can be used by all people according to their requirement
- In Java, "Java EE, JSF, Struts, Spring" technologies are frameworks, all these technologies are used by different companies for developing projects according to the project requirements.

Why this package classes are called Collection Framework?

This package classes provides low level services with well defined data structures to solve collecting heterogeneous dynamic number of objects as a single entity. Due to these reason java.util package classes are collectively called Collection Framework.

All collection classes are implementing from *java.io.Serializable* so the collection object along with all its internal objects can be stored in local file system or can be sent across network to remote computer. Here **Rule is** *objects stored in collection are also should be Serializable types to store collection in file*.

Need of Collection framework classes

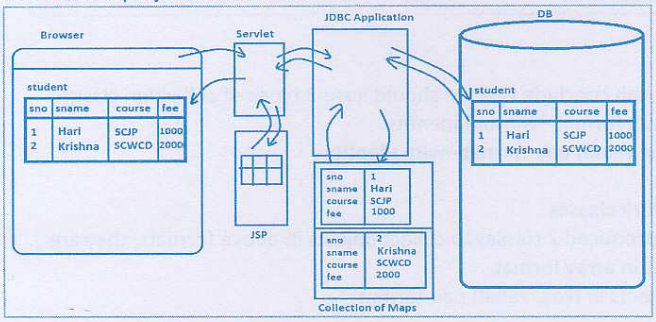
In Java projects Collection Framework classes are used to store and transport objects are of same and different types without *size limitation*.

Project Code design with collection classes

As you know, every project has three layers Model, View, Controller.

- ➔ Model layer application collects data from DB using ResultSet and store it collection object, and return this collection object to Controller layer application.
- ➔ Then Controller layer application read data from collection object and fill it in View layer required code using HTML components
- ➔ Finally this HTML code is passed to client then browser displays the result to end-user.

Below is the project architecture



In this project

- ➔ First we retrieve student's records using JDBC API
- ➔ Then we read all records from ResultSet and store them in Collection object
- ➔ Finally we send that collection object to Servlet
- ➔ Then Servlet by using JSP displays student's records on browser.

Check *Projects* section for the complete code of this project.

Collection classes are Java data structures

Basically collection framework classes are Java data structures. These classes internally use several standard data structures to collect objects such as growable array, vector, stack, queue, linked list, doubly linked list, hash table, tree etc. So, collection object size is automatically incremented and decremented.

The primary advantages of collections framework are that it

- **Reduces programming effort** by providing useful data structures and algorithms so you don't have to write them yourself.
- **Increases performance** by providing high-performance implementations of useful data structures and algorithms. Because the various implementations of each interface are interchangeable, programs can be easily tuned by switching implementations.
- **Provides interoperability between unrelated APIs** by establishing a common language to pass collections back and forth.
- **Reduces the effort required to learn APIs** by eliminating the need to learn multiple ad hoc collection APIs.
- **Reduces the effort required to design and implement APIs** by eliminating the need to produce ad hoc collections APIs.
- **Fosters software reuse** by providing a standard interface for collections and algorithms to manipulate them.

In how many formats can we collect objects?

We can collect objects in 2 ways

1. In **array format** – in this format object does not have identity
2. In **(key, value) pair format** - in this format object has identity

Storing Employee data in different format

Array format

7279	Hari Krishna	Naresh Technologies	Java

Key, Value pair format

eno	7279
ename	Hari Krishna
working with	Naresh Technologies
teaches	Java

In the first format data does not have identity, so user may interpret it wrongly. In the second format data has identity, so it is very clear for the user to understand.

Types of collection classes

Based on above discussion we can conclude that we should have 2 types of collection classes

1. To collect objects in array format –without identity.
2. To collect object in (key, value) pair format –with identity.

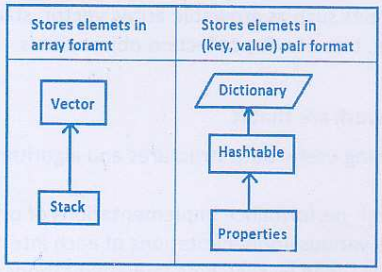
Legacy and collection framework classes

In Java 1.0 version SUN only introduced 2 classes to collect objects in above formats, they are

a. **Vector**: It stores objects in array format
b. **Hashtable**: It stores objects in (key, value) pair format

In addition to above 2 classes we have one more class called **Stack,** it is the subclass of Vector class. It is used to store objects and to retrieve them in **L**ast **I**n **F**irst **O**ut (**LIFO**) fashion.

Below diagram shows relation between these classes.



The drawback of Vector and Hashtable classes

These two classes were created as thread-safe classes, means all the methods in these two classes are declared as synchronized. Hence for every method call on these two objects either for adding or removing elements, JVM locks and unlocks these two objects. So in Single thread model application it leads to performance issue. In Single thread model application execution is sequential so there is no data corruption, hence no need locking. To solve this performance issue problem in Java 1.2 SUN introduced non-thread safe classes as an alternative to above two classes.

Collection Framework

In *Java 1.2 version* more number of collection classes are added. These classes are added to include more features and also to solve the limitations of the above classes – Vector and Hashtable. So in Java 1.2 version new classes were added as alternative to these two classes they are – ArrayList and HashMap respectively.

Note:

From Java 1.2 version all collection classes are collectively called as collection framework. And the collection classes available from Java 1.0 version are called as legacy classes.

Types of Collection Framework Hierarchy

The collection framework is divided into two hierarchies to store objects in array format and (key, value) pair format, they are:

1. **Collection** hierarchy
2. **Map** hierarchy

Collection hierarchy classes collect object in **array format**, it is the root interface of all those classes. **Map** hierarchy classes collect object in **(key, value) pair format**, it is the root interface of all those classes.

Common terminology used in this chapter

Before learning more points on collection framework let us first learn common terminology used in this chapter

collection of objects

The collection object that contains other normal class objects is called collection of objects.

collection of collections

The collection object that contains other collection objects is called collection of collections.

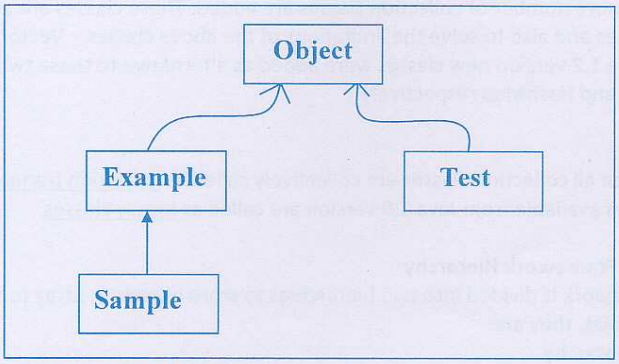
collection of maps

The collection object that contains map objects is called collection of maps.

Element	- means **object**
Entry	- means **(key, value) pair**
Homogeneous elements	- same class objects
Heterogeneous elements	- different class objects
Unique elements	- different class objects or same class objects with different state / reference
Duplicate elements	- same class objects with same state /reference

Note: Unique and duplicate elements is actually decided by "==" operator and equals() method returned value. If equals method returns true, those two objects are considered as duplicate (same objects), else they are considered as unique (different objects).

Consider below hierarchy



In the below list find out all above four types of objects
+ Same type of objects are called homogeneous objects / elements
+ Different type of objects are called heterogeneous objects / elements.

Homogeneous objects

```
Example e1 = new Example();
Example e2 = new Example();
Sample  s1 = new Sample();
Sample  s2 = new Sample();
```

Heterogeneous objects

```
Example e = new Example();
Test    t = new Test();
String  s = new String();
Integer i = new Integer();
```

+ If objects have same state/reference, they are considered as duplicate objects.
+ Else they are considered as unique objects.
+ They can be homogeneous or heterogeneous.

Based on *state* you decide the objects type

Homogeneous **duplicate** objects

```
Example e1 = new Example();
Example e2 = new Example();
```

Homogeneous **unique** objects

```
Example e1 = new Example(5, 6);
Example e2 = new Example(7, 8);
```

Heterogeneous **unique** objects

```
Example e = new Example();
Test    t = new Test();
```

In the below list, find out the above type of objects

```
new String("a");
new String("a");
new String("a");
```

```
new String("a");
new String("b");
new String("c");
```

```
new Integer(10);
new Double(10);
new String("a");
```

In the next sections you will learn Collection **and** Map **hierarchy classes**

Collection hierarchy

Collection hierarchy classes are divided into three categories - **Set, List,** and **Queue.**

Using Collection hierarchy classes we can collect **unique** and **duplicate objects** in array format.

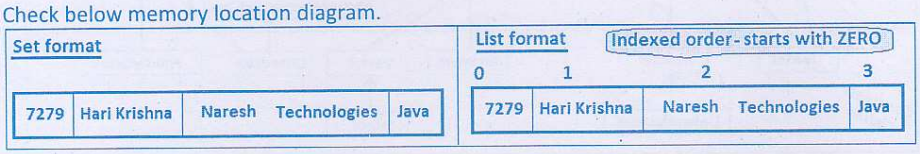
Set is a **unique collection,** it does ***not*** allow duplicate elements.
Set is the super interface for all unique collection classes.

List is a **duplicate collection,** it allows duplicate elements.
List is the super interface for all duplicate collection classes.

Unique collection – Set – is an **unsorted** and **unordered**, means it stores elements without index, where as

Duplicate collection – List – is **unsorted** and **indexed ordered,** means it stores each element with index exactly same as array in insertion order.

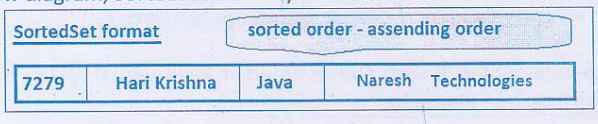
Check below memory location diagram.



As you can noticed in the above diagram, Set does not store elements in indexed or sorted order, whereas List stores elements in indexed order but not in sorted order.

SortedSet is a sub interface of Set that **stores elements in sorted order** either in ascending or descending order based on the object's natural sorting order.

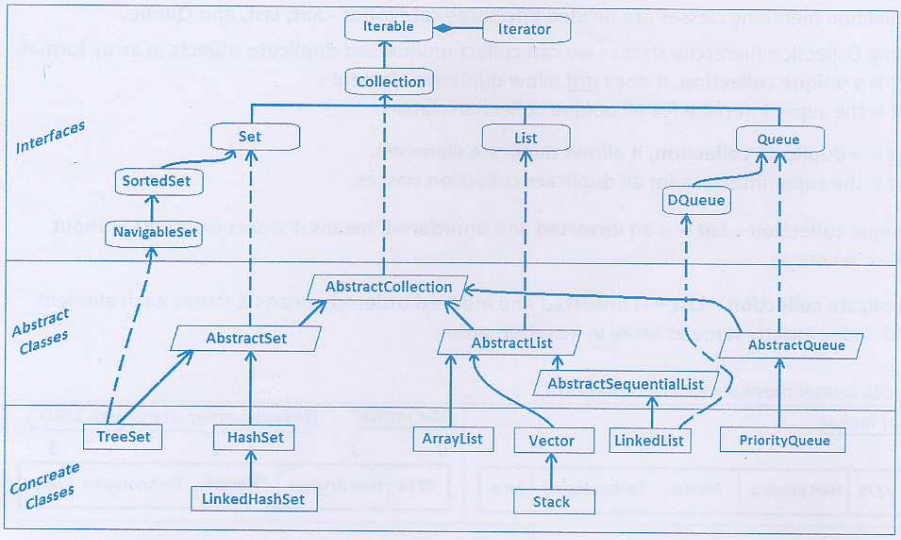
Check below diagram, SortedSet memory location format



NavigableSet is a sub interface of SortedSet. It is added in Java 6 to add more navigation methods to sorted set. The methods are *lower, floor, ceiling*, and *higher* return elements respectively less than, less than or equal, greater than or equal, and greater than a given element, returning *null* if there is no such element.

Queue is a root interface of all types of queues. Besides basic Collection operations, queues provide additional insertion, extraction, and inspection operations. Queues typically, but do not necessarily, order elements in a FIFO (first-in-first-out) manner. We can create different types of queues like **1. *Priority Queues***, which order elements according to a supplied comparator, or the elements' natural ordering, and **2. *LIFO queues*** (or stacks) which order the elements LIFO (last-in-first-out). In a **3. FIFO queue**, all new elements are inserted at the tail of the queue. Other kinds of queues may use different placement rules.
Every Queue implementation must specify its ordering properties.

Collection interface classes' hierarchy

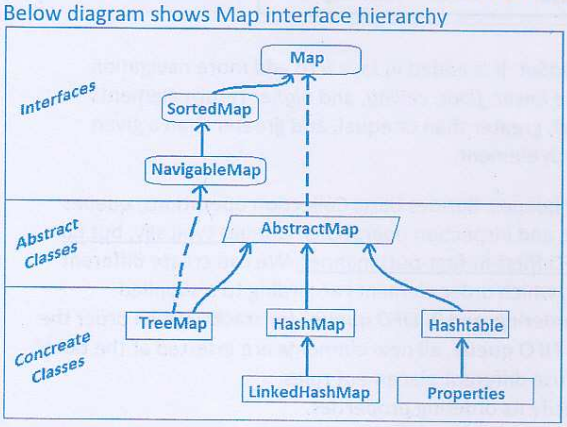


In the above hierarchy, *Vector and Stack* classes are available since Java 1.0, *LinkedHashSet* class is available since Java 1.4, *Queue* is available since Java 5, and *NavigableSet* is available since Java 6, and all other remaining classes and interfaces are available since Java 1.2 version.

Map hierarchy

Map hierarchy classes are used to collect elements in (key, value) pair format.
In a map, *keys* should be unique and *values* can be duplicated.
Each (*key, value*) is called an entry. In map by default entries are not sorted.

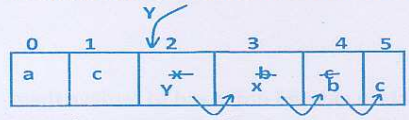
SortedMap is the sub interface of Map. It sorts entries based on keys natural sorting order.
Below diagram shows Map interface hierarchy



In this hierarchy, *Hashtable and Properties* classes are available since Java 1.0, *LinkedHashMap* class is available since Java 1.4, and *NavigableMap* is available since Java 6, and all other remaining classes and interfaces are available since Java 1.2 version.

Different project Scenarios- we choose above collection classes

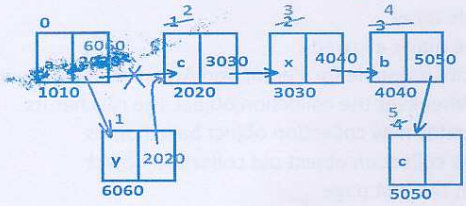
- To collect objects in array format we choose *Collection hierarchy* classes.
- To collect objects in (key, value) pair format we should choose Map hierarchy classes.
- To collect unique elements we must choose *Set implemented* classes.
- To collect unique and/or duplicate elements in indexed order we must choose *List implemented* classes.
- To retrieve elements in FIFO manner we choose *Queue implemented* classes.
- To store heterogeneous unique elements with no order we must choose *HashSet*.
- To store heterogeneous unique elements in insertion order we must choose *LinedHashSet*.
- To store elements in their natural sorting order we must choose *TreeSet*. TreeSet allows only Homogeneous Comparable unique elements.
- To store and retrieve duplicate elements in random access we must choose *ArrayList* or *Vector*. These two classes functionality is same except, Vector is thread-safe. So, in Single Thread model application we should choose ArrayList, in multithreading model application if list object is modifying concurrently we should choose Vector.
- The ***drawback of ArrayList and Vector*** is "They give less performance if we perform insertions and deletions in middle", because for every insertion of new element, all remaining elements must be moved right hand side, and for every element deletion, elements must be moved to left hand side. As shown in the below diagram.



ArrayList or Vector object

- To solve this performance issue we must choose LinkedList. It gives high performance in inserting or deleting elements in middle. It internally uses linked list data structure, so there will not be any data movements, instead we will have only changing links, and the indexes.

  Check below diagram for its working functionality



- To store and retrieve elements in FILO manner we should choose *Stack*.

- To store unique entries, we must choose *HashMap*.
- To store unique entries in insertion order, we must choose *LinkedHashMap*.
- To store unique entries in sorted order, we must choose *TreeMap*. It only allows homogeneous unique entries with Comparable type keys.

- Hashtable and HashMap both works similar, the only difference is Hashtable is thread-safe.

- Properties class is used to store properties permanently.

Let us understand each class separately also let us understand important interview questions
Interview Questions you will face from this chapter

1. What is the implemented data structure?
2. What is the default capacity, and incremental capacity?
3. What type of elements allowed to add?
4. In which order elements are preserved and retrieved?
5. Is null allowed, if so how many?

List implemented classes
We should choose list implemented classes to store heterogeneous unique or duplicate elements in indexed order. It has four concrete classes, they are
ArrayList, Vector, Stack, LinkedList.

To store duplicate objects Vector class is already available, then why ArrayList and LinkedList classes are given?
Read below points.

ArrayList, Vector
We should choose these two classes to store elements in indexed order and to retrieve them randomly, it means retrieving nth element directly without retrieving (n-1) elements, because these two classes are sub classes of ***RandomAccess*** interface.

Specific Functionalities of these two classes

1. duplicate objects are allowed in indexed order
2. heterogeneous objects are allowed
3. insertion order is preserved
4. implemented data structure is Growable array
5. null insertion is possible, more than one null is allowed.
6. Initial capacity is 10, incremental capacity is double for Vector and ArrayList uses below formula (currentCapacity * 3 / 2 + 1). Whenever the collection object size reaches its max capacity, that class internal API creates new collection object based on its incremental capacity value. Then in new collection object old collection object elements are copied, as we discussed in the first page.

The **main difference** between these two classes is:

Vector object is thread-safe it means synchronized object – so multiple threads cannot modify this object concurrently. It is best suitable in multithreaded model application to get desired result. But in single thread model application it gives poor performance, because for every operation this object should be locked and locked, which useless activity in single thread model application. To solve this performance issue in collection framework ArrayList class is given.

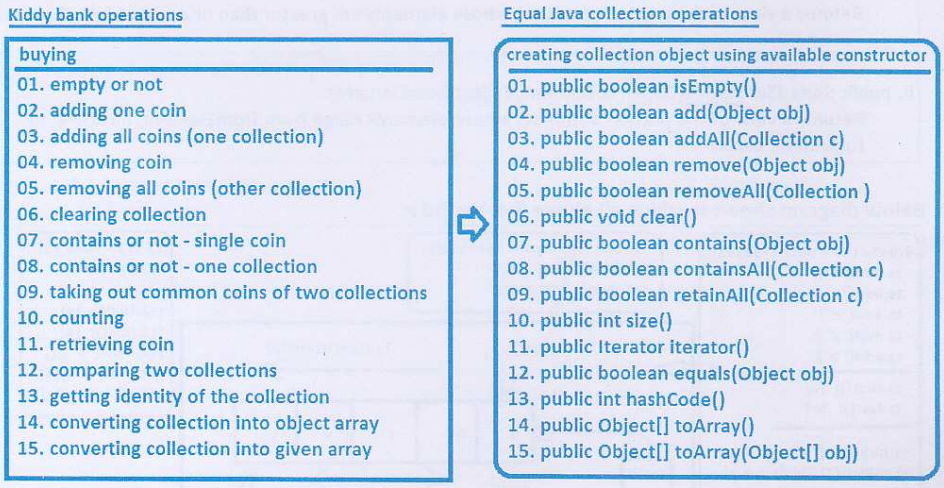
ArrayList object is not thread-safe it means it is not synchronized object. It is best suitable in multithreaded model application, also in multithreaded model application if you ensure there is not data corruption.

Limitation of both classes

Inserting and removing elements in middle is costlier operation. Because for every insert operation elements must move to right and for every delete operation elements must move to left from that location.

Collection interface methods - 15:

In collection interface below methods are defined commonly for all collections to perform different operations



Set interface methods - 15:

Why add(Object) method return type is boolean?

In Collection interface **add()** method is defined commonly for both **Set** and **List.** So its return type should be boolean return **false** if duplicate element is added.

Answer my question, is Set interface has any specific methods?

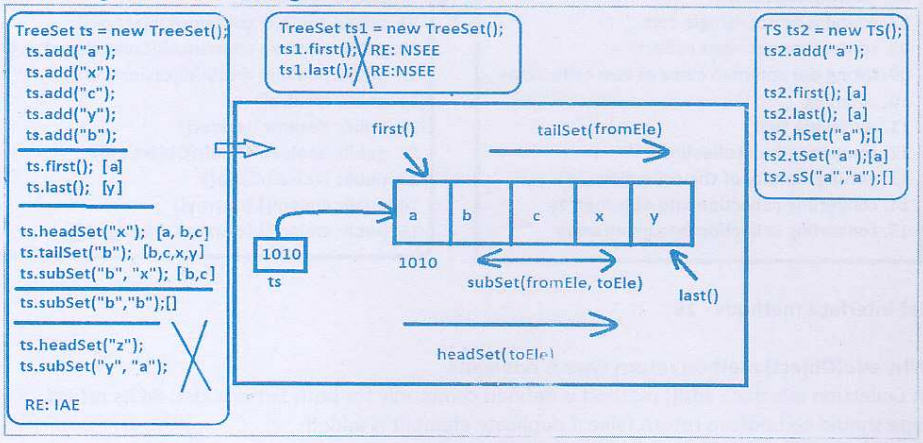
No, Set interface have same above 15 methods because its specific functionality – maintaining unique elements – is implemented with the same add(Object) method as explained above.

SortedSet interface methods -21:

In addition to above 15 methods this interface has specific 6 methods based on sorting functionality, they are:

1. public Comparator comparator()
   Returns the comparator used to order the elements in this set, or null if this set uses the natural ordering of its elements
2. public Object first()
   retruns first (lowest) element
3. public Object last()
   retruns last (highest) element
4. public SortedSet headSet(Object toElement)
   Returns a view of the portion of this set whose elements are strictly less than toElement
5. public SortedSet tailSet(Object fromElement)
   Returns a view of the portion of this set whose elements are greater than or equal to fromElement.
6. public SortedSet subSet(Object toElement, Object fromElement)
   Returns a view of the portion of this set whose elements range from fromElement, inclusive, to toElement, exclusive.

Below diagram shows working all above five methdos



List interface methods – 25

In addition to Collection interface 15 methods List interface has 10 more specific methods based in indexed order.

```
1. public void add(int index,Object element)      => inserts element at given index
2. public boolen addAll(int index,Collection c)   => inserts Collection at given index
3. public Object get(int index)                   => returns element at given index, it will not remove that element
                                                     from collection
4. public Object set(int index,Object element)    => replaces element at given index, and return old element
5. public Object remove(int index)                => deletes element at given index, and return old element

6. public int indexOf(Object element)             => retruns first occurence index of given object
7. public int lastIndexOf(Object element)         => retruns last occurence index of given object

8. public List subList(int start, int end)        => retruns sub list in the given range

9. public ListIterator listIterator()             => retruns ListIterator on this collection
10. public ListIterator listIterator(int index)   => retruns ListIterator on this collection from given index
```

The more specific operations we can do on List object are: insert, retrieve, replace, and remove at the given index.

ArrayList class Constructors

ArrayList class has below constructors to create its object

```
public class ArrayList extends AbstractList implements List RandomAccess, Cloneable, Serializable
{
  public ArrayList();
      Constructs an empty list with an initial capacity of ten.
  public ArrayList(int capacity)
      Constructs an empty list with the specified initial capacity.
  public ArrayList(Collection c)
      Constructs a list containing the elements of the specified collection, in the order they are returned by the
      collection's iterator.

  //Specific methods
  public void trimToSize()
      Trims the capacity of this ArrayList instance to be the list's current size. An application can use this operation to
      minimize the storage of an ArrayList instance.
  public void ensureCapacity(int minCapacity)
      Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of
      elements specified by the minimum capacity argument.
  protected void removeRange(int fromIndex, int toIndex)
      Removes from this list all of the elements whose index is between fromIndex, inclusive, and toIndex, exclusive.
      Shifts any succeeding elements to the left (reduces their index). This call shortens the list by (toIndex - fromIndex)
      elements.
}
```

Below program shows adding, retrieving and removing elements from ArrayList.

```
//ArrayListDemo.java
import java.util.ArrayList;

public class ArrayListDemo
{
        public static void main(String args[])
        {
                // Creating an ArrayList
                ArrayList al = new ArrayList();
                System.out.println("Initial size of ArrayList: " + al.size());

                //Add elements to the ArrayList
                al.add("Red");
                al.add("Green");
                al.add("Blue");
                al.add("Pink");
                al.add("Orange");

                System.out.println("\nSize of ArrayList after additions: " + al.size());

                //Display the ArrayList
                System.out.println("\nContents of ArrayList After add: " + al);

                // Remove 4th index element from ArrayList
                al.remove(4);
                System.out.println("\nContents of ArrayList after remove index: " + al);

                // Remove "Pink" element from ArrayList
                al.remove("Pink");
                System.out.println("\nContents of ArrayList after remove object: " + al);
                System.out.println("\nSize of ArrayList after deletions: " + al.size());

                //retrieving 1st index element
                String alElement        = (String)(al.get(1));
                System.out.println("alElement :"+alElement);

                //inserting at 2nd index
                al.add(2, alElement +" rose" );
                System.out.println(al);
        }
}
```

Vector class constructors and methods

As being a Legacy class Vector class has below specific methods and constructors

```
public class Vector extends AbstractList implements List, RandomAccess, Cloneable,
Serializable
{
        public Vector()
                Constructs an empty vector so that its internal data array has size 10.

        public Vector(Collection c)
                Constructs a vector containing the elements of the specified collection, in the
                order they are returned by the collection's iterator.

        public Vector(int initialCapacity)
                Constructs an empty vector with the specified initial capacity.

        public Vector(int initialCapacity, int incrementalCapacity)
                Constructs an empty vector with the specified initial capacity and capacity
                increment.

        //specific methods
        public synchronized boolean isEmpty()
        public synchronized void addElement(Object o)
        public synchronized Object elementAt(int index)
        public synchronized Object firstElement()
        public synchronized Object lastElement()
        public synchronized void insertElementAt(Object o, int index)
        public synchronized void setElementAt(Object o, int index)
        public synchronized boolean removeElement(Object o)
        public synchronized void removeAllElements()
        public synchronized void removeElementAt(int index)
        protected synchronized void removeRange(int index, int index)
        public synchronized int size()
        public synchronized int capacity()
        public synchronized void ensureCapacity(int capacity)
        public synchronized void trimToSize()
        public synchronized void setSize(int size)

        public Enumeration elements()

        public synchronized void copyInto(Object[])

        public synchronized Object clone()
}
```

Below program shows storing and removing elements from Vector class

```
//VectorDemo.java
import java.util.Vector;
public class VectorDemo{
        public static void main(String args[]){
                Vector v = new Vector();
                for(int i = 0 ; i <= 9 ; i++){
                        v.addElement(new Integer(i*10));
                }

                v.removeElementAt(0);
                v.removeElementAt(1);

                for(int i = 0 ; i < v.size() ; i++){
                        System.out.println(v.elementAt(i));
                }
        }
}
```

Below program shows changing Vector capacity

```
//Address.java
class Address {

        String hno, street, city, ph;

        Address(String hno, String street, String city, String ph) {

                this.hno = hno;
                this.street = street;
                this.city = city;
                this.ph = ph;
        }
}
//Customer.java
class Customer {

        String name;   int age;   Address address;

        Customer(String name, int age, Address address)    {
                this.name       = name;
                this.age        = age;
                this.address    = address;
        }
```

```
        public String toString(){
                return (        "Name: " + name + "\nAge: " + age +
                                "\nH.No: " + address.hno + "\nStreet: " + address.street +
                                "\nCity: " + address.city+"\nph"+address.ph
                        );
        }
}

//VectorCapacitySizeDemo.java
import java.util.Enumeration;
import java.util.Vector;

public class VectorCapacitySizeDemo {
        public static void main(String[] args)  {
                Vector v = new Vector(3);

                System.out.println("Intial Capacity and Size of Vector is..");
                System.out.println("Capacity: " + v.capacity());
                System.out.println("Size       : " + v.size());

                System.out.println();

                Customer c1 = new Customer("Newton", 30 ,
                                                new Address("2-3-102/1",
                                                "Ameerpet","Hyderabad","12345"));
                Customer c2 = new Customer("Einstene",23,
                                                new Address("1-10-1022/3",
                                                "Kukatpally","Hyderabad","23345"));
                Customer c3 = new Customer("Clinton",31,
                                                new Address("1-3-32",
                                                        "Amberpet","SecBad","33345"));
                Customer c4   =new Customer("Ken Thomposn",31,
                                                new Address("1-3-32",
                                                        "Amberpet","SecBad","33345"));
                Customer c5 = new Customer("Edward Shepered",31,
                                                new Address("1-3-32",
                                                        "Amberpet","SecBad","33345"));
                Customer c6 = new Customer("Michael Slater",31,
                                                new Address("1-3-32",
                                                        "Amberpet","SecBad","33345"));
                Customer c7 = new Customer("Justin Langer",31,
                                                new Address("1-3-32",
                                                        "Amberpet","SecBad","33345"));
```

```
                v.add(c1);
                v.add(c2);

                System.out.println("Reached  its Capacity....Or not?");
                System.out.println("Capacity: " + v.capacity());
                System.out.println("Size       : " + v.size());
                System.out.println();

                v.add(c3);
                System.out.println("Reached  its Capacity....Or not?");
                System.out.println("Capacity: " + v.capacity());
                System.out.println("Size       : " + v.size());
                System.out.println();

                v.add(c4);
                System.out.println("Reached  its Capacity....Or not?");
                System.out.println("Capacity: " + v.capacity());
                System.out.println("Size       : " + v.size());
                System.out.println();

                v.add(c5);
                v.add(c6);
                System.out.println("Reached  its Capacity....Or not?");
                System.out.println("Capacity: " + v.capacity());
                System.out.println("Size       : " + v.size());
                System.out.println();

                v.add(c7);
                System.out.println("Capacity Increased...and Size of Vector Altered..");
                System.out.println("Capacity: " + v.capacity());
                System.out.println("Size       : " + v.size());
                System.out.println();

                System.out.println("The customer labels are: ");
                System.out.println();

                Enumeration e = v.elements();
                while(e.hasMoreElements())
                {
                        System.out.println(e.nextElement());
                        System.out.println();
                }
        }
}
```

Stack class constructors and methods

The Stack class represents a last-in-first-out (LIFO) stack of objects. It extends class Vector with five operations that allow a vector to be treated as a stack. The usual *push* and *pop* operations are provided, as well as a method to *peek* at the top item on the stack. When a stack is first created, it contains no items.

public class Stack extends Vector{

public Stack()
Creates an empty Stack.

public boolean empty()
Tests if this stack is empty.

public Object push(Object o)
Pushes an item onto the top of this stack.

public synchronized Object pop()
Removes the object at the top of this stack and returns that object as the value of this method.

public synchronized Object peek()
returns the object at the top of this stack without removing it from the stack.

public synchronized int search(Object o)
Returns the 1-based position where an object is on this stack. If the object o occurs as an item in this stack, this method returns the distance from the top of the stack of the occurrence nearest the top of the stack; the topmost item on the stack is considered to be at distance 1. The equals method is used to compare o to the items in this stack.

}

Below program shows java based stack operations

```
//StackDemo.java
import java.util.Stack;
public class StackDemo{
    public static void main(String args[]){
        Stack st        = new Stack();
        st.push(new Integer(10));          st.push(new Integer(20));
        st.push(new Integer(30));          st.push(new Integer(40));

        System.out.println(st);
        System.out.println(st.pop());
        System.out.println(st);
        System.out.println(st.peek());
        System.out.println(st);
        System.out.println(st.search(new Integer(10)));
    }
}
```

Set Implemented Classes:

public class HashSet extends AbstractSet implements Set, Cloneable, Serializable{

public HashSet()
Constructs a new, empty set; the backing HashMap instance has default initial capacity (16) and load factor (0.75).

public HashSet(Collection c)
Constructs a new set containing the elements in the specified collection. The HashMap is created with default load factor (0.75) and an initial capacity sufficient to contain the elements in the specified collection.

public HashSet(int initialCapacity)
Constructs a new, empty set; the backing HashMap instance has the specified initial capacity and default load factor (0.75).

public HashSet(int initialCapacity, float loadFactor)
Constructs a new, empty set; the backing HashMap instance has the specified initial capacity and the specified load factor.

}

public class LinkedHashSet extends HashSet implements Set, Cloneable, Serializable{

public LinkedHashSet()
public LinkedHashSet(Collection c)
public LinkedHashSet(int initialCapacity)
public LinkedHashSet(int initialCapacity, float loadFactor)

}

public class TreeSet extends AbstractSet implements SortedSet, Cloneable, Serializable{

public TreeSet()
Constructs a new, empty tree set, sorted according to the natural ordering of its elements. All elements inserted into the set must implement the Comparable interface.

public TreeSet(Collection c)
Constructs a new tree set containing the elements in the specified collection, sorted according to the natural ordering of its elements

public TreeSet(SortedSet s)
Constructs a new tree set containing the same elements and using the same ordering as the specified sorted set.

public TreeSet(Comparator comparator)
Constructs a new, empty tree set, sorted according to the specified comparator. It should be used to supply custom sorting order for comparable objects and to store non-comparable objects.

}

```
// Below program explains all Set implemented concrete classes, and their functionality

//it is a dummy class, created to store its objects in collection
//Example.java
public class Example{
        int x = 10;
        int y = 20;
}

//SetClassesDemo.java
import java.util.HashSet;
import java.util.LinkedHashSet;
import java.util.TreeSet;

public class SetClassesDemo  {
        public static void main(String[] args) {
                //set objects creation
                HashSet         hs      = new HashSet();
                LinkedHashSet   lhs     = new LinkedHashSet();
                TreeSet         ts      = new TreeSet();
                //printing initial size of collection objects
                System.out.println("hs      length: "+hs.size());
                System.out.println("lhs     length: "+lhs.size());
                System.out.println("ts      length: "+ts.size());
                System.out.println();

                //adding elements to hs
                System.out.println("is 20 added: "+  hs.add( Integer.toString(20) ) );
                System.out.println("is a added: "+   hs.add( new Character('a') )   );
                System.out.println("is b added: "+   hs.add( new Character('b') )   );
                System.out.println("is abc added: "+ hs.add( "abc"  )               );
                System.out.println("is Abc added: "+ hs.add( "Abc" )                );
                System.out.println("is abc added: "+ hs.add( "abc" )                );
                System.out.println("is abc added: "+ hs.add( new String("abc") )    );
                System.out.println("is Example added: "+hs.add( new Example() ) );
                System.out.println("is Example added: "+hs.add( new Example() ) );

                //adding nulls to hs
                System.out.println("is null added: "+hs.add(null));
                System.out.println("is null added: "+hs.add(null));
                System.out.println();

                //printing hs modified size and elements
                System.out.println(hs.size());
                System.out. println("hs : " + hs);
```

```
        //adding elements to lhs
        System.out.println("is 20 added: "+lhs.add( Integer.toString(20)));
        System.out.println("is a added: "+lhs.add(new Character('a')));
        System.out.println("is b added: "+lhs.add(new Character('b')));
        System.out.println("is abc added: "+lhs.add("abc"));
        System.out.println("is Abc added: "+lhs.add("Abc"));
        System.out.println("is abc added: "+lhs.add("abc"));
        System.out.println("is abc added: "+lhs.add(new String("abc")));
        System.out.println("is Example added: "+lhs.add(new Example()));
        System.out.println("is Example added: "+lhs.add(new Example()));

        //adding nulls to lhs
        System.out.println("is null added: "+lhs.add(null));
        System.out.println("is null added: "+lhs.add(null));
        System.out.println();

        //printing lhs modified size and elements
        System.out.println(lhs.size());
        System.out. println("lhs : " + lhs);

        //adding homogeneous elements to ts
        System.out.println("is abc added: "+ts.add("abc"));
        System.out.println("is xyz added: "+ts.add("xyz"));
        System.out.println("is bbc added: "+ts.add("bbc"));
        System.out.println("is pqr added: "+ts.add(new String("pqr")));

        //adding heterogeneous element
        System.out.println("is Integer added: "+ts.add(new Integer(10)));

        //adding null to ts
        System.out.println("is null added: "+ts.add(null));
        System.out.println("is null added: "+ts.add(null));

        //printing ts modified size and elements
        System.out.println(ts.size());
        System.out. println("ts : " + ts);

        //Q) when can we add null to TreeSet?
        //If it is solo TreeSet we can add null, check the below code
        TreeSet solots = new TreeSet();
        solots.add(null);

        System.out.println(solots.size());
        System.out. println("solots : " + solots);
    }
}
```

Note: From Java 7 onwards *we cannot* add *single null also.*

This solots code leads to NPE.

Three types of cursors to retrieve elements from all types of collection objects

We have three types of cursor objects for retrieving elements they are:

1. Enumeration
2. Iterator
3. ListIterator

All above three are interfaces. Their subclasses are created by SUN as inner classes inside collection classes. These subclasses objects are returned via factory methods. These factory methods are also defined inside collection classes. So these factory methods must be called with collection object from which we want to retrieve elements.

Those Factory methods are:

- to retrieve Enumeration object
  public Enumeration ***elements()***
  - This method is defined inside both Vector and Hashtable classes

- to retrieve Iterator object
  public Iterator ***iterator()***
  - It is defined in Collection interface. So it can be called on all collection objects

- to retrieve ListIterator object
  public ListIterator ***listIterator()***
  public ListIterator listIterator(int index)
  - These methods are defined in List interface. So these methods can be called only List type collection objects.

Q) What are the operations we must perform using above three cursor objects?

1. Checking is element available in the underlying collection
2. If element available retrieve that element reference.

So to perform above two tasks these three cursor objects must have minimum two methods

1. For checking element available or not
2. For returning the element reference from collection and moving cursor to next location

All above cursor objects cursor is initially pointing to before first location of the collection. Below is the cursor object structure with given collection object "c".

We must call nextElement()/next() methods till cursor reaches last element.

Understanding *Enumeration* interface

Enumeration is a legacy interface defined in Java 1.0 to retrieve elements from legacy collections Vector and Hashtable. It has below two methods to retrieve elements:

```
public interface Enumeration{
        public boolean hashMoreElements()
          - It checks whether elements exist or not. If not existed returns false.

        public Object nextElement()
          - It returns the next element. If no element is available it throws an exception
                                                    "java.util.NoSuchElementException"
}
```

Q) Who implement above two methods?

SUN in Vector and Hashtable classes as inner class.

Q) How can we obtain Enumeration object on Vector and Hashtable?

- Vector class has below factory method
  public Enumeration **elements**()
- Hashtable class has below two factory methods
  1. public Enumeration **keys**()
     - it returns enumeration on all keys
  2. public Enumeration **elements**()
     - it returns enumeration on all values

Q) How can we obtain Enumeration on collection framework classes?

A) We cannot obtain Enumeration on collection framework classes directly, we must use "*Collections*" class static method "*enumeration()*". Its prototype is:

public static Enumeration **enumeration(Collection c)**

Q) Write a program to retrieve elements from *Vector* object by using Enumeration.

```
Vector v = new Vector();
v.add(10);
v.add("a");

Enumeration e = v.elements();

while( e.hasMoreElements() ){
     Object obj = e.nextElement();
     System.out.println(obj);
}
```

Q) Write a program to retrieve elements from *ArrayList* object by using Enumeration.

```
ArrayList  al = new ArrayList();
al.add(10);
al.add("a");

Enumeration e = Collections.enumeration( al );

while( e.hasMoreElements() ){
     Object obj = e.nextElement();
     System.out.println(obj);
}
```

Execution flow of above two programs

- Enumeration object is created, stored in e variable, it is pointing to before first element.
- While loop is iterated till *e.hasMoreElements()* returns false. It returns false when cursor reaches end of the collection elements.
- In every iteration *e.nextElement()* returns the next element in the collection and moves the cursor to this next element location..

Understanding *Iterator* interface

Iterator is a Collection Framework interface defined in Java 1.2. It is a cursor object used to retrieve elements from all Collection type objects List, Set and Queue including Vector. It is an alternative of Enumeration object and replacement of "for" loop on collection objects.

List objects has index so we can retrieve elements by using for loop as shown below

Retrieving elements from List object

```
ArrayList al = new ArrayList();

al.add("a");
al.add("b");
al.add("c");
al.add("a");
```

Since ArrayList stores objects with **index** we can retrieve elements using **for loop** as shown below,

```
for(int i = 0 ; i < al.size() ; i++){
        SopIn(al.get(i));
}
```

But how can we retrieve elements from Set objects since they do not have index?
We do not have *get* method in *Set*, check below code

Retrieving elements from Set object

```
LinkedHashSet lhs = new LinkedHashSet();

lhs.add("a");
lhs.add("b");
lhs.add("c");
lhs.add("a");
```

Size() method is available but get() method is not available, then how can we retrieve elements from Set

```
for(int i = 0 ; i < lhs.size() ; i++){
        SopIn( ? );
}
```

Iterator is only the option we have to retrieve elements from Set objects

Iterator interface provides below three methods for retrieving elements from collection

```
public interface Iterator{

  public boolean hasNext()
        Returns true if the iteration has more elements.

  public Object next()
        Returns the next element in the iteration, and moves cursor to next location.

  public void remove()
        Removes from the underlying collection the last element returned by the Iterator.
}
```

Q) Who does implement above three methods?
SUN in all Collection subclasses as inner class.

Q) How can we obtain Iterator object?
In Collection interface we have below method to obtain Iterator object

public Iterator iterator();

Below statement we should use to obtain Iterator object on a given collection object **c**.

Iterator itr = c.iterator();

Q) Write a program to retrieve elements from *ArrayList* object by using Iterator.

```
ArrayList al = new ArrayList();

al.add("abc");
al.add("bbc");
al.add("cbc");

Iterator itr = al.iterator();

while(itr.hasNext()){
      Object ele = itr.next();
      System.out.println(ele);
}
```

Q) Write a program to retrieve elements from *LinkedHashSet* object by using Iterator.

```
LinkedHashSet lhs = new LinkedHashSet();

lhs.add("abc");
lhs.add("bbc");
lhs.add("cbc");

Iterator itr = lhs.iterator();

while(itr.hasNext()){
      Object ele = itr.next();
      System.out.println(ele);
}
```

ClassCastException while using collection elements:
next() method returns every element as java.lang.Object type from collection. So we must cast the returned objects to their own type by using cast operator to call that returned object's specific members. So in this code we should use **instanceof** operator to avoid **CCE**.

Below program shows retrieving elements from Set implemented classes using Iterator.
This program shows adding string and integer objects to LHS and retrieving and printing all string objects in uppercase and remaining objects as it is.

```
//IteratorDemo.java
import java.util.LinkedHashSet;
import java.util.Iterator;

public class IteratorDemo  {

        public static void main(String[] args) {

                LinkedHashSet lhs = new LinkedHashSet();

                lhs.add("abc");
                lhs.add(10);
                lhs.add(20);
                lhs.add(40);
                lhs.add("bbc");

                Iterator lhsIterator = lhs.iterator();

                while(lhsIterator.hasNext()) {

                        Object obj = lhsIterator.next();

                        if(obj instanceof String)  {

                                String str = ( (String)obj ).toUpperCase();
                                System.out.println("Modified String :"+ str );

                        }
                        else{
                                System.out.println("Object :"+obj);
                        }

                }//while closed
        }
}
```

Iterator rules

Rule #1: While retrieving elements, in while loop **collection object structure should not be modified** by either adding element or by removing element using collection methods. It leads to **java.util.ConcurrentModificationException.** This behavior of Iterator is called ***fail-fast***.

Rule #2: next() method should not be called on empty collection or empty location, it leads RE: **java.util.NoSuchElementException**

Rule #3: remove() method must be called only after next() method call and also only once, violation leads to **java.lang.IllegalStateException**

Below program shows Iterator rules

```
//IteratorMethodsRulesDemo.java
import java.util.LinkedHashSet;
import java.util.Iterator;

public class IteratorMethodsRulesDemo {
        public static void main(String[] args) {
                LinkedHashSet lhs = new LinkedHashSet();
                lhs.add("a");
                lhs.add("b");
                lhs.add("c");

                Iterator itr = lhs.iterator();

                //itr.remove();  RE: java.lang.IllegalStateException

                Object o1 = itr.next();
                System.out.println(o1);//a

                itr.remove(); //a is removed

                //lhs.add("d");  // this statement causes CME when next() is called

                Object o2 = itr.next(); // RE: java.util.ConcurrentModificationException
                System.out.println(o2);//b

                Object o3 = itr.next();
                System.out.println(o3);//c

                //Object o4 = itr.next(); //RE: java.util.NoSuchElementException

                itr.remove();//c is removed

                lhs.add("d"); //it is correct and d is added

                System.out.println(lhs);//[b, d]
        }
}
```

Q) What are the differences between *Enumeration* and *Iterator*?

Iterator takes the place of Enumeration in the Java collections framework. Iterators differ from enumerations in two ways:

- Iterators allow the caller to remove elements from the underlying collection during the iteration with well-defined semantics.
- Method names have been improved.

Below is the complete list of differences

Diff #1:

- Enumeration is legacy interface.
- Iterator is collection framework interface

Diff #2:

- Enumeration method names are big
- Iterator method names are short and it has an additional method remove()

Diff #3:

- Enumeration cannot remove elements
- Iterator can remove elements with well defined semantics

Diff #4:

- Enumeration is not fail-fast, so we can modify collection while enumerating.
- Iterator is fail-fast, so we cannot modify collection while iterating.

Diff #5:

- Enumeration cannot be applied on collection framework classes directly
- Iterator can be applied on all types of collection classes including legacy classes as these classes are also subclass of Collection and Map.

ListIterator

It is a bidirectional cursor object used to retrieve elements only from List implemented collection objects- ArrayList, Vector, Stack, LinkedList. It is a sub interface of Iterator.

Obtaining its object

In "List" interface we have below two methods to obtain its object

- public ListIterator listIterator()
- public ListIterator listIterator(int index)

➔ first method iterates starts from begin to end
➔ second method iterates starts from given index to end

Q) What are the differences between Iterator and ListIterator?

The main difference is:

Diff #1:

- Iterator is unidirectional – means elements are retrieved only in forward direction.
- ListIterator is bidirectional – means elements are retrieved in both directions.

The other differences are:

Diff #2:

- Iterator is a super interface
- ListIterator is a sub interface of Iterator

Diff #3:

- Iterator can be applied on both Set and List implemented classes
- ListIterator can only be applied on List implemented classes, because it works with index.

Diff #4:

- Iterator can only allow us to retrieve and remove elements
- ListIterator can allow us to retrieve, remove, insert, and also replace elements.

ListIterator methods

It has below special methods to perform above operations:

It has below methods to retrieve elements in forward direction.
These methods are inherited from Iterator interface

1. public boolean **hasNext**()
2. public Object **next**()

It has below methods to retrieve elements in backward direction

3. public boolean **hasPrevious**()
4. public Object **previous**()

For removing current returned element

5. public void **remove**()

For inserting element

6. public void **add**(Object obj)
   - It inserts the given element in the next location of current returned element.

For replacing current element

7. public void **set**(Object obj)
   - It replaces current returned element with given element.

Write a program to create ArrayList object with
- **3 String objects and 3 Integer objects**
- **while iterating replace all String objects with their uppercase**
- **Insert the number 20 after the first Integer object**

```
//ListIteratorDemo.java
import java.util.*;

class ListIteratorDemo {
        public static void main(String[] args) {
                ArrayList al = new ArrayList();
                al.add("a");
                al.add("b");
                al.add("c");
                al.add(5);
                al.add(6);
                al.add(7);

                System.out.println("AL elements before iteration: "+ al);
                int count = 1;

                ListIterator litr = al.listIterator();
                while (litr.hasNext()){
                        Object obj = litr.next();

                        if (obj instanceof String){
                                String s = (String)obj;
                                litr.set(s.toUpperCase());
                        }
                        else if (obj instanceof Integer){
                                if (count == 1){
                                        litr.add(20);
                                        count++;
                                }
                        }//else if
                }//while

                System.out.println("AL elements after iteration: "+ al);

                //retrieving elements in reverse order
                while (litr.hasPrevious()){
                        Object obj = litr.previous();
                        System.out.println(obj);
                }
        }//main
}//class
```

Understanding the operations performed by addAll(), removeAll(), containsAll() and retainAll() methods in the Collection interface.

- addAll() - performs Union operation
- containsAll() - performs subset operation
- removeAll() - performs subtraction operation
- retainAll() - performs intersection operation

Assume we have two collections c1 and c2
If we call

➔ **c1.addAll(c2)** - all elements of c2 are added to c1.

If c1 is **Set** collection duplicates are *not* added.
If c1 is **List** collection duplicates are also added.

➔ **c1.containsAll(c2)** - It returns true if all elements of c2 available in c1. else it returns false.

➔ **c1.removeAll(c2)** - It removes all elements of c2 from c1. Duplicate elements are also removed.

➔ **c1.retainAll(c2)** - It removes all elements of c1 except c2 elements. Duplicate elements are also retained.

Conclusion:
addAll(), retainAll(), removeAll() methods modifies current collection object based on argument collection object elements.

Difference between Comparable and Comparator interfaces
Comparable and Comparator both are interfaces. They are used to specify elements sorting order in TreeSet and keys in TreeMap.

The difference between these two interfaces
Comparable used to specify natural sorting order of the object.
Comparator is used to specify custom sorting order of the object.

To specify the sorting order by developer these two interfaces have below methods

Comparable method
```
public int compareTo(Object o1)
```

Comparator method
```
public int compare(Object o1, Object o2)
```

Adding custom objects to TreeSet

- TreeSet internally uses Comparable interface method *compareTo()* to compare and sort objects.
- So we can only add Comparable objects to TreeSet.
- Compiler allows to add non-comparable objects to TreeSet because add () method parameter is Object, but JVM throws ClassCastException. In TreeSet class add() method the argument object is cast to Comparable to call compareTo() method on that object.

  So to add custom object we must follow below procedure
  - its class must be subclass of Comparable interface
  - then should override compareTo() method with required sorting order logic. This logic is used as default sorting order to sort that class objects.

TreeSet class sample add() method logic

```
public boolean add(Object obj)
{
        if (! (this.isEmpty()) )
        {
                Comparable c = (Comparable) obj;
                c.compateTo(tsItr.next()); //passing TreeSet elements.
        }
}
```

For more details check running notes.

Below program shows adding Example objects to TreeSet

Case #1: Example class is not deriving from Comparable

```
class  Example {
        int x;
        Example(){}
        Example(int x){this.x = x;}
}

class AddingCustomObjects{
        public static void main(String[] args){
                TreeSet ts = new TreeSet();
                ts.add(new Example());
                //ts.add(new Example());       RE: ClassCastException: Example cannot
                                                    be  cast to java.lang.Comparable
        }
}
```

Case #2: Example class is deriving from Comparable and overriding compareTo().

```
class  Example implements Comparable {

        int x;
        Example(){}
        Example(int x){this.x = x;}

        public int compareTo(Object obj) {
                Example e = (Example)obj;
                return e.x - this.x;
        }

        public String toString(){
                return ""+x;
        }
}

class AddingCustomObjects{

        public static void main(String[] args){

                TreeSet ts = new TreeSet();

                ts.add(new Example());
                ts.add(new Example(20));
                ts.add(new Example());
                ts.add(new Example(5));
                ts.add(new Example(30));
                ts.add(new Example(20));
                ts.add(new Example());

                System.out.println(ts.size());
                System.out.println(ts);
        }
}
```

O/P

Changing default sorting order of objects

```
/*
 * This program demonstrates developing a custom Comparator to show you
 * changing natural order of predefined objects like String and wrapper classes.
 * call this ComparatorDemo.java
 */
import java.util.Comparator;
```

```
import java.util.TreeSet;

class StringComparator implements Comparator {
        public int compare(Object a, Object b) {
                String aStr, bStr;
                aStr = (String) a;
                bStr = (String) b;

                // Reverse the comparison
                return bStr.compareTo(aStr);
        }
}

public class ComparatorDemo {
        public static void main(String args[]) {
                /*
                  - Creating a TreeSet object using no-arg constuctor,
                  - So, elements are sorted according to natural sorting order of adding object
                 */
                TreeSet ts = new TreeSet();
                // Add elements to the TreeSet
                ts.add("C");
                ts.add("A");
                ts.add("B");
                ts.add("E");
                ts.add("F");
                ts.add("D");

                System.out.println("ts object with default comparator :"+ts);

                /*
                  - Creating a TreeSet object using Comparator parameter constructor,
                  - to sort elements according to custom comparator sorting order
                 */

                TreeSet tsc = new TreeSet(new StringComparator());
                tsc.add("C");
                tsc.add("A");
                tsc.add("B");
                tsc.add("E");
                tsc.add("F");
                tsc.add("D");

                System.out.println("tsc object with custom comparator :"+tsc);
        }
```

```
}
```

Adding *non-Comparable* objects to TreeSet by using custom Comparator. StringBuffer is a non-comparable object. Below program shows adding StringBuffer objects to TreeSet.

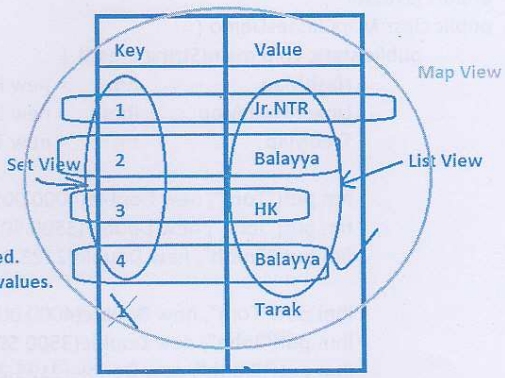
```
//SBComparator.java
class SBComparator implements java.util.Comparator {

        public int compare(Object obj1, Object obj2){
                StringBuffer sb1 = (StringBuffer)obj1;
                StringBuffer sb2 = (StringBuffer)obj2;

                String s1 = sb1.toString();
                String s2 = sb2.toString();

                return s2.compareTo(s1);
        }
}

//AddingSBtoTS.java
import java.util.*;

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

                TreeSet ts = new TreeSet(new SBComparator());
                ts.add(new StringBuffer("a"));
                ts.add(new StringBuffer("b"));
                ts.add(new StringBuffer("x"));
                ts.add(new StringBuffer("c"));
                ts.add(new StringBuffer("y"));

                System.out.println(ts);
        }
}
```

Map architecture

Map object has three views
- Set view - collection of keys
- List view - collection of values
- Map view - collection of entries

Check the diagram.

we can retrive value by passing key its associated.
So key should be unique, but we can duplicate values.



Map interface methods

```
public interface Map{

    public abstract boolean isEmpty();

    public abstract Object put(Object key, Object value);
    public abstract void putAll(Map m);

    public abstract Object get(Object key);

    public abstract Object remove(Object key);
    public abstract void clear();

    public abstract boolean containsKey(Object key);
    public abstract boolean containsValue(Object value);

    public int size();

    public abstract Set keySet();
    public abstract Collection values();
    public abstract Set entrySet();

    public abstract boolean equals(Object obj);
    public abstract int hashCode();
}
```

```
//MapClassesDemo.java
import java.util.*;
public class MapClassesDemo {
        public static void main(String args[]) {
                HashMap         hm      = new HashMap();
                LinkedHashMap   lhm     = new LinkedHashMap();
                TreeMap         tm      = new TreeMap();

                hm.put("Tom", new Double(4000.00));
                hm.put("John", new Double(3500.50));
                hm.put("Smith", new Double(2125.25));

                lhm.put("Tom", new Double(4000.00));
                lhm.put("John", new Double(3500.50));
                lhm.put("Smith", new Double(2125.25));

                tm.put("Abc", new Double(4000.00));
                tm.put("Cbc", new Double(3500.50));
                tm.put("Bbc", new Double(2125.25));

                System.out.println("hm  elements : "+hm);
                System.out.println("lhm elements : "+lhm);
                System.out.println("tm  elements : "+tm);

                //Getting the set of keys and getting the iterator
                Set             set             = hm.keySet();
                Iterator        hmitr           = set.iterator();

                System.out.println("\nThe Account balance hm Account holders:");
                while(hmitr.hasNext()) {
                        Object key      = hmitr.next();
                        Object value    = hm.get(key);
                        System.out.println(key +" : "+ value);
                }
                System.out.println();

                System.out.println("\nThe Account balance of lhm Account holders:");
                //Getting the collection of values and getting Iterator
                Collection lhmcol       = lhm.values();
                Iterator   lhmitr       = lhmcol.iterator();

                while(lhmitr.hasNext()) {
                        System.out.println(lhmitr.next());
                }
```

```
                System.out.println();
                System.out.println("\nThe Account balance of tm Account holders:");

                //Getting the set of Entries and obtaining Iterator
                Set tmset       = tm.entrySet();
                Iterator tmitr  = tmset.iterator();
                while(tmitr.hasNext()) {
                        Map.Entry me = (Map.Entry)tmitr.next();

                        System.out.print(me.getKey() + ": ");
                        System.out.println(me.getValue());
                }
                double balance = ((Double)hm.get("John")).doubleValue();
                hm.put("John", new Double(balance + 1000));

                System.out.println("John's new balance: " + hm.get("John"));
        }
}

//HashTableDemo.java
import java.util.*;
public class HashTableDemo {
        public static void main(String[] args) {
                Hashtable ht = new Hashtable();

                ht.put("Tom",new Double(10.50));
                ht.put("Henery",new Double(12.25));
                ht.put("Scott",new Double(8.75));

                System.out.println("ht elements :"+ht);

                Enumeration e = ht.keys();
                System.out.println("Employee Name \t\t Employee Sal");

                while(e.hasMoreElements()) {
                        String key = (String) e.nextElement();
                        System.out.println(key + "\t\t" + ht.get(key));
                }
                System.out.println();

                double sal = ((Double)ht.get("Scott")).doubleValue();
                sal += 11.25;

                ht.put("Kean",new Double(sal));
                ht.put("Scott",new Double(sal));

                e = ht.keys();
                System.out.println();
```

```
                System.out.println("The changed values are: ");
                while(e.hasMoreElements()) {
                        String key = (String) e.nextElement();
                        System.out.println(key + "\t\t" + ht.get(key));
                }
        }
}

//Understanding hashcode and equals() methods usage with collection objects
public class Student implements java.io.Serializable{
        int sno;
        String name;
        int whichClass;

        public Student(int sno, String name, int whichClass) {
                this.sno        = sno;
                this.name       = name;
                this.whichClass = whichClass;
        }

        public int hashCode() {
                return whichClass;
        }

        public boolean equals(Object obj){
                //below condition should be presented; else there is a chance of getting
                                                        ClassCastException
                if(obj instanceof Student)
                {
                        Student s = (Student)obj;
                        if (    this.sno == s.sno               &&
                                this.name.equals(s.name)        &&
                                this.whichClass == s.whichClass
                          )
                        {
                                return true;
                        }
                }
                return false;
        }
        public String toString(){
                return whichClass+": "+name;
        }
}
```

```
//School.java – storing and removing custom elements from Set
import java.util.*;
class School {
        public static void main(String[] args) {
                HashSet hs = new HashSet();

                hs.add(new Student(1, "Balayya", 1));
                hs.add(new Student(1, "Harikrishna", 1));

                hs.add(new Student(2, "Balayya", 2));
                hs.add(new Student(2, "Harikrishna", 2));
                hs.add(new Student(2, "Jr.NTR", 1));
                hs.add(new Student(2, "Jr.NTR", 1));

                hs.add(new Integer(8));
                hs.add(new Integer(10));
                hs.add(new String("a"));

                System.out.println(hs);

                hs.remove(new Student(2, "Jr.NTR", 1));
                System.out.println("\nafter removing");
                System.out.println(hs);
        }
}


// – using custom elements as keys to store, retrieve and remove values from Map
//Address.java
class Address implements java.io.Serializable{
        int hno;
        int streetNo;
        String city;
        Address(int hno, int streetNo, String city){
                this.hno        = hno;
                this.streetNo   = streetNo;
                this.city       = city;
        }
        public String toString(){
                return  "hno: "+ hno +"\n" +
                        "streetNo: "+ streetNo +"\n" +
                        "city: "+ city;
        }
}
```

```
//College.java
import java.util.*;
import java.io.*;
public class College {
        public static void main(String[] args) throws Exception       {
                LinkedHashMap       lhm     = new LinkedHashMap();

                lhm.put(new Student(11, "Rama", 11), new Address(1, 2, "Hyd"));
                lhm.put(new Student(11, "Raju", 12), new Address(1, 3, "Hyd"));

                lhm.put(new Student(12, "Sita", 11), new Address(12, 30, "Sec"));
                lhm.put(new Student(12, "Rani", 12), new Address(14, 30, "Sec"));

                //storing students information in file
                FileOutputStream      fos      = new FileOutputStream("studentinfo.ser");
                ObjectOutputStream oos      = new ObjectOutputStream(fos);

                oos.writeObject(lhm);
                System.out.println("\n Student info is saved ");
        }
}

//CollegeWebSite.java
import java.io.*;
import java.util.*;
public class CollegeWebSite {
        public static void main(String[] args) throws Exception{
                //reading students information from file
                FileInputStream       fis     = new FileInputStream("studentinfo.ser");
                ObjectInputStream     ois     = new ObjectInputStream(fis);

                LinkedHashMap       lhm     = (LinkedHashMap)ois.readObject();
                Set keySet                =       lhm.keySet();
                Iterator keyItr           =       keySet.iterator();

                while (keyItr.hasNext()){
                        Object key      = keyItr.next();
                        Object value    = lhm.get(key);

                        System.out.println(key + " student Address");
                        System.out.println(value);
                        System.out.println();
                }
        }
}
```

Collections class

This class consists exclusively of static methods that operate on or return collection objects. It contains polymorphic algorithms that operate on collections, "wrappers", which return a new collection backed by a specified collection, and a few other odds and ends.

This class is a member of the Java Collections Framework available since 1.2

This class has methods for performing below different operations on collection objects

1. searching element - only work for List
2. sorting elements - only work for List
3. swapping two elements - only work for List
4. shuffling elements - only work for List
5. reverse the order - only work for List
6. rotating elements at given distance - only work for List
7. copy elements - only work for List
8. checking whether two collections are disjoint - work for Set, List and Queue
9. Finding max, min elements - work for Set, List and Queue
10. Finding number of frequencies of a given element - work for Set, List and Queue
11. Obtaining Enumeration object - work for Set, List and Queue

12. Obtaining *synchronized* Set, List, Queue, Map, SortedSet, SortedMap collections
13. Obtaining *unmodifiable* Set, List, Queue, Map, SortedSet, SortedMap collections
14. Creating *immutable* empty Set, List, Map collections
15. Creating *singleton* Set, List, Map
16. Obtaining *reverse natural sorting* Comparable object

Very important methods for 1.5 collection API enhancement with Generics

Q) How can we convert Non-generics collections to Generics based collections?

A) We should use below methods.

17. Creating *checked* Set, List, Queue, Map, SortedSet, SortedMap

Arrays class

This class contains various methods for manipulating arrays (such as sorting and searching). This class also contains a static factory that allows arrays to be viewed as lists.

The methods in this class all throw a NullPointerException if the specified array reference is null.

Q) How can we print given array data directly without using for loop?

A) *Arrays* class has static parameterized *toString* method to return array content in String format.

For method prototypes and more details on above two classes please refer its API documentation. By following API documentation try to develop your own programs to check all above operations.

Properties class:
This class is used to store properties permanently in a file.
Property means (key, value) in string form.

Rule: To store properties in a file both key and value type should be java.lang.String type. Because file can only display string data, and more over any data that we type in file is of type string.

Procedure to create Properties file

- The file that contains properties is called properties file, and as per coding standards its extension should be ".properties".
- Every property's name and value must be separated by "=" or ":"
- Every property must be placed in new line.
- The properties file of Employee is look like as below

Employee.properties

```
eid=7279
ename=HariKrishna
desig=JavaFaculty
company=NareshTechnologies
```

It has below methods to read and store properties from a properties file

1. public void **load**(InputStream in)
2. public String **getProperty**(String name)
   If the given name is not existed in the file it returns null
3. public String **getProperty**(String name, String msg)
   If the given name is not existed in the file it returns the given message not *null*
4. public Enumeration **propertyNames**()
   Returns all property names as Enumeration object
5. public Object **setProperty**(String name, String value)
   It sets this new property to Properties object
   It returns Object because it internally calls "put" method to add this property
6. public void **list**(PrintStream out)
7. public void **store**(OutputStream out, String msg)
   Both methods are used to store current properties object to file
   "store" method allows to store properties with given message comment, where as it is not possible with "list" method.

```
//PropertiesDemo.java
import java.io.*;
import java.util.*;

public class PropertiesDemo
{
        public static void main(String[] args)  throws Exception
        {
                Properties p = new Properties();
                p.load(new FileInputStream(args[0]));

                System.out.println("name:"+p.getProperty("eid"));
                System.out.println("sal:"+p.getProperty("ename"));
                System.out.println("desg:"+p.getProperty("desig"));
                System.out.println("company:"+p.getProperty("company"));

                System.out.println("abc:"+p.getProperty("abc"));
                System.out.println("abc:"+p.getProperty("abc","abc key is not found"));

                p.setProperty("exp","Since 2004");
                Enumeration e          = p.propertyNames();

                while(e.hasMoreElements()){
                        System.out.println(p.getProperty((String)e.nextElement()));
                }

                p.list(new PrintStream(new FileOutputStream("
                                                        List Result.properties")));
                p.store(new PrintStream(new FileOutputStream("Store
                Result.properties")),"Emp details");

        }
}

emp.properties
eid=7279
ename=Harikrishna
desig=JavaSCJPFaculty
company=NareshiTechnologies
```

//Internationalization – I18N

+ To execute the Java application in different countries with that country specific language we must develop I18N.

+ It means if we want to display a GUI components names - for example, Username, Password, SignIn - in a country specific language we must develop this feature.

+ In Java we have two classes given in java.util package to develop I18N.
They are:
1. Locale
2. ResourceBundle

+ Locale represents country
+ ResourceBundle loads the country's properties file dynamically based on the Locale object.

Procedure to develop I18N application:
1. We must develop a properties file one per each country with that country lanaguage. For all these properties file we must create a base file in the default language in which we want to display names.

2. Then we must create ResourceBundle object by using its factory method getBundle() by passing the properties file base name with out extension.

3. ResourceBundle loads the current country properties file by appending its locale to the base name.

4. ResourceBundle gets the current country locale from Locale object, it gets that current country locale from the OS from its environment variable "lang" and "country".

Methods:
public ResourceBundle getBundle(String fileName);
public ResourceBundle getBundle(String fileName, Locale locale);
- for creating resource bundle object
- If given properties file is not existed it throws java.util.MissingResourceException

public String getString(String pn)
- for retriving property value
- If pn is not found in the properties file it also throws MRE
- If pn is null it throws NPE
- If pn value is not String it throws CCE

Locale class contains many fields to represent each country language and country code.
For US
language code is: en

counrty code is: US

Below are the variables defined in Locale class for above US codes
public static final Locale ENGLISH
public static final Locale US

Refer Locale API for more details

Q) How can we get System properties?
A) By calling System.getProperties() method.

```
import java.util.*;

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

                //loaded properties file based on current locale
                ResourceBundle rb     = ResourceBundle.getBundle("wish");
                //ResourceBundle rb  = ResourceBundle.getBundle("wish",
                                                        Locale.getDefault());

                //loaded properties file based on passed Locale
                //ResourceBundle rb  = ResourceBundle.getBundle("wish",
                                                new Locale(Locale.FRENCH.toString(),
                                                        Locale.FRENCH.toString()));

                String wish      = rb.getString("hellowish");

                System.out.println(wish);
        }
}
```

```
#wish.properties        => it is the default properties file for all languages and countries
hellowish=hello
```

```
#wish_fr_FR.properties   => it contains France country language wish
hellowish=hello in French
```

```
#wish_de_DE.properties   => it contains German country language wish
hellowish=hello in German
```

StringTokenizer:
This class allows us to break a string into tokens. To break the string we must pass a delimiter, it is a character or substring available in the tokenizing string.

The default delimiters are
1. the space character,
2. the tab character (\t),
3. the newline character (\n),
4. the carriage-return character (\r), and
5. the form-feed character (\f).

In constructing StringTokenizer object if we do not pass delimiter, it uses above delimiters to break the string. From Java 1.4 onwards it is recommended to use String class "*split*" method rather StringTokenizer class.

Constructors:
It has three constructors

a. **public StringTokenizer(String str)**
It breaks given string with above default delimiters

For example

```
String s = "Hari Krishna";

StringTokenizer st = new StringTokenizer (s);
```

It returns two tokens
1. Hari
2. Krishna

b. **public StringTokenizer(String str, String delim)**
It breaks given string with given delimiter

For example

```
String s = "Hari Krishna";

StringTokenizer st = new StringTokenizer (s, "r");
```

It returns 3 tokens
1. Ha
2. i K
3. ishna

c. **public StringTokenizer(String str, String delim, boolean returnDelims)**
It breaks given string with given delimiter and also includes delimiter as token

For example

```
String s = "Hari Krishna";
```

StringTokenizer st = new **StringTokenizer** (s, "r", true);
It returns 5 tokens

1. Ha
2. **r**
3. i K
4. **r**
5. ishna

Delimiter "r" is also included in the list of tockens

Q) What is the direct subclass of Enumeration interface?
StringTokenizer

Methods:
It has below 5 methods

1. **To get tokens count**
   public int countToken()

2. **To check is token available or not**
   public boolean hasMoreElements()
   public boolean hasNextToken()

3. **To retrieve token**
   public Object nextElement()
   public String nextToken()

//StringTokenizerDemo.java

```
import java.util.StringTokenizer;

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

                String str              = "Hari Krishna, Naresh i Technologies";

                //StringTokenizer st    = new StringTokenizer(str);

                //StringTokenizer st    = new StringTokenizer(str, ",");

                StringTokenizer st      = new StringTokenizer(str,",",true);

                System.out.println("Number of tockens: "+st.countTokens());
                while(st.hasMoreElements()) {
                        System.out.println(st.nextElement());
                }
        }
}
```

Introduction to Generics

This feature is introduced in Java 5 to provide the type for method's parameters and return types dynamically by achieving compile-time type checking for solving ClassCastException.

It means this feature allows us to pass the type for method parameter and return type at runtime when this class object is created.

If we define "class / interface / enum" with generic type that

- class is called parameterized type or generic type.
- method is called generic type method
- constructor is called generic type constructor

Syntax to create a class with generic parameter:

```
class Example<T>{

    void m1(T t){}
    <T> m2(){ --- }

    Example<T> (){}
    Example<T> (T t){}
}
```

object creation of above class

We must pass class name as argument for creating object of this class

```
Example<String> e1 =  new Example<String>();
```

Rule: Then using e1 we should call m1() method only by passing "String" objects, because m1 method parameter for e1 object is String. If we pass Integer objects it leads to CE

```
e1.m1("a");  ✓
e1.m1(10);   ✗
```

Example class object to accept only Integer objects in calling m1() method

```
Example<Integer> e2 = new Example<Integer>();
e2.m1("a");  ✗
e2.m1(10);   ✓
```

The main reason of introducing Generics

Basically Generic Types feature provides compile-time type safety for calling methods as shown above. The main reason of introducing this feature is to solve collection coding problem *ClassCastException*. To solve this exception compiler should only allow homogeneous objects to add to collection object. So generics feature is introduced to create homogeneous collection that means a collection with only same type objects.

So by using this feature we can tell to compiler what type of elements only must be added to a collection object, as shown below

```
ArrayList<String> al = new ArrayList<String>();
```

From the above statements compiler allows us to add only String objects to this collection object. If we add Integer objects compiler throws compile time error, because add method parameter is String.

```
al.add("a");
al.add(10);
```

Below are the collection classes' definitions as per Java 1.4 and Java 5 with generics

Java 1.4 ArrayList and Iterator

```
public class ArrayList{

   public boolean add(Object obj){}
   public Object get(int index)

   Iterator iterator(){}

}
```

```
public interface Iterator{

   public boolean hasNext();
   public Object next();
   public void remove();
}
```

Java 5 ArrayList and Iterator

```
public class ArrayList<E>{

   public boolean add(E e){}
   public E get(int index)

   Iterator<E> iterator(){}

}
```

```
public interface Iterator<E>{

   public boolean hasNext();
   public E next();
   public void remove();
}
```

Java 1.4 LinkedHashMap

```
public class LinkedHashMap{

   public Object put(Object key, Object value){}
   public Object get(Object key){}

   public Set keySet(){}
   public Collection values(){}
   public Set entrySet(){}

}
```

Java 5 LinkedHashMap

```
public class LinkedHashMap<K, V>{

   public V put(K key, V value){}
   public V get(K key){}

   public Set<K> keySet(){}
   public Collection<V> values(){}
   public Set<Map.Entry<K,V>> entrySet(){}

}
```

Q) Write a program to collect only String objects using ArrayList. Retrieve and print all elements by using get, iterator and for-each loops.

Java 1.4 based code

```
import java.util.*;

class NonGenericAL{

    public static void main(String[] args){

        ArrayList al = new ArrayList();

        al.add("a");
        al.add("b");
        al.add("c");

        Iterator itr = al.iterator();

        while( itr.hasNext() ){

            Object obj = itr.next();
            String s = (String)obj;
            System.out.println(
                    s.toUpperCase()
            );

        }

    }
}
```

Java 5 based code

```
import java.util.*;

class GenericAL{

    public static void main(String[] args){

        ArrayList<String> al =
                new ArrayList<String>();

        al.add("a");
        al.add("b");
        al.add("c");

        Iterator<String> itr = al.iterator();

        while( itr.hasNext() ){

            String s = itr.next();
            System.out.println(
                    s.toUpperCase()
            );

        }

//if we do not add <String> parameter to
Iterator then next() method returns elements
as java.lang.Object type, then again we should
downcast it String class as shown below

        Iterator itr2 = al.iterator();

        while( itr.hasNext() ){

            Object obj = itr.next();
            String s = (String)obj;
            System.out.println(
                    s.toUpperCase()
            );

        }
    }
}
```

SCJP Questions – Generics Rules

Rule #1: we can create generic class object without passing parameter, for that object the implicit parameter is java.lang.Object.

```
ArrayList al = new ArrayList();

//for this al object add method parameter is java.lang.Object.
//So we can add all heterogeneous objects

al.add(10);
al.add("a");
```

Note: for these two methods call compiler shows a "Note" saying unchecked operation. Note is not a CE, so for the current class ".class" file is generated.

Q) How can we create generic type collection object to collect heterogeneous elements.

A) The parameter type should be java.lang.Object

```
ArrayList<Object> al = new ArrayList<Object>();
```

Rule #2: Upcasting is not possible for generic type, it means "List<String>" is not a subtype of "List<Object>"

Find CEs

```
List<String> ls = new ArrayList<String>();
List<Object> ls = new ArrayList<String>();
```

Why not allowed?

Because using "ls" we can call add method by passing heterogeneous objects, which is illegal.

```
Ex: ls.add(10);
```

Q) Is below method definition and its call allowed?

```
interface Shape{}
class Rectangle implements Shape{}
class Test{
        static void m1(List<Shape> ls){

        }
        pubic static void main(String[] args){
                m1(new ArrayList<Rectangle>())
        }
}
```

Method call leads to CE, because ArrayList<Rectangle> is not a subclass of List<Shape>

Rule #3: to solve above problem we should use wildcard character (?).

- ? means unknown type, it creates super type for generics.

- Collection<Object> is not a super generic type of all collection objects.
- Collection<?> is a super type of all collection objects.

```
//List<Object> ls1 = new ArrayList<String>();

List<?> ls1 = new ArrayList<String>();
List<?> ls2 = new ArrayList<Integer>();
```

Problem #1 with wildcard: method parameters are not replaced with type, but return types are replaced with java.lang.Object. Due to this we cannot call parameterized methods (mutator methods) but we can call return type method (accessor methods)
So, with this ? syntax we cannot call add method to add elements to collection object because compiler does not know add() method parameter.

```
ls1.add("a"); CE:
```

but we can call get() method on this unknown type, because every added object is of type java.lang.Object.

Find out CE in the below lines

```
Object obj = ls1.get(0);
String s = ls1.get(0);
```

Q) In which case should we use wildcard character syntax based generic type?
A) To develop runtime polymorphism based method to take all generic type objects, we must use wildcard generic type parameter. But in this method we can only read elements but we can add, or update or remove elements.

```
void m1(List<?> ls){
        for(Object obj : ls){
                System.out.println(obj);
        }
}

m1(new ArrayList<String>());
m1(new ArrayList<Integer>());
m1(new ArrayList<Object>());
m1(new ArrayList<Rectangle>());
```

Problem #2 with ? generic type
Wild card generic type parameterized method allows all types of generic class objects. We need new syntax to pass specific category subclasses generic types.

For example

ArrayList<Rectangle>, ArrayList<Square>, ArrayList<Circle>

Solution: We must define boundaries for "?"

We can define two boundaries

1. subclasses boundary (? extends Shape)
2. superclasses boundary (? super Shape)

- ArrayList<? extends Shape> means the argument collection object should contain only Shape subclasses objects.
- ArrayList<? super Shape> means the argument collection object should contain only Shape superclasses objects.

For example:

```
void m1(List<? extends Shape>){

}

m1(new ArrayList<String>());
m1(new ArrayList<Integer>());
m1(new ArrayList<Rectangle>());
m1(new ArrayList<Square>());
m1(new ArrayList<Circle>());
m1(new ArrayList<Shape>());
m1(new ArrayList<Object>());

void m1(List<? super Shape>){

}

m1(new ArrayList<String>());
m1(new ArrayList<Integer>());
m1(new ArrayList<Rectangle>());
m1(new ArrayList<Square>());
m1(new ArrayList<Circle>());
m1(new ArrayList<Shape>());
m1(new ArrayList<Object>());
```

In the above two m1() methods also we are not allowed to call "add" method because of unknown type (?) but we can call "get" with destination variable type "Object".

I would try to update our site JavaEra.com everyday with various interesting facts, scenarios and interview questions. Keep visiting regularly.....

Thanks and I wish all the readers all the best in the interviews.

www.JavaEra.com

A Perfect Place for All Java Resources