IO in Java:

Input and Output in Java is stream based. Stream is a flow of database. It is called sequence of Data also.

Stream are following 2 types:

1. Byte Oriented Stream: It is based on ASCII code.
2. Character Oriented Stream: It is based on UNI code.

java.io package contains classes and interfaces those used to implements IO in java.

Byte Oriented Stream classes:

1. InputStream: It is an abstract class those responsible to read bytes from source by the help of read() method.
2. OutputStream: It is an abstract class those responsible to write bytes on destination by the help of write() method.
3. FileInputStream: It is used to read bytes from file.
4. FileOutputStream: It is used to write bytes on file.
5. PrintStream: It is a special class those contains println() method.

Etc…

Example: to read data from file

import java.io.\*;

public class ReadData

{

public static void main(String args[])

{

try

{

FileInputStream fin = new FileInputStream(args[0]);

int b=0;

while(true)

{

b=fin.read();

if(b==-1)

{

break;

}

System.out.print((char)b);

}

}

catch(Exception ex)

{

System.out.println(ex);

}

}

}

Example: Write bytes on file.

import java.io.\*;

import java.util.Scanner;

public class WriteData

{

public static void main(String args[])

{

try

{

Scanner sc = new Scanner(System.in);

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

String str = sc.nextLine();

FileOutputStream fo = new FileOutputStream(args[0]);

byte b[] = str.getBytes();

fo.write(b);

System.out.println("File has successfull saved.");

}

catch(Exception ex)

{

System.out.println(ex);

}

}

}

Character Oriented Stream classes:

1. Reader: It is used to read characters.
2. Writer: It is used to write characters.
3. FileReader: It is used to read characters from file.
4. FileWriter: It is used to write characters on file.
5. BufferedReader: It is used to contains characters in buffer.
6. PrintWriter: It is a child class of PrintStream class.
7. InputStreamReader: It is used to convert bytes into Characters.
8. OutputStreamWriter: It is used to convert characters into bytes.

Etc…

Example:

import java.io.\*;

public class Test

{

public static void main(String args[])

{

try

{

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

System.out.print("Enter your name : ");

String name = br.readLine();

System.out.print("Enter your age : ");

int age = Integer.parseInt(br.readLine());

System.out.println("Hi "+name+", you are only "+age+" years old.");

}

catch(Exception ex)

{

System.out.println(ex);

}

}

}

Networking:

Networking is a medium of communication between more than one node. Networking are following 2 types:

1. Connection Oriented Networking: It is based on TCP/IP Protocol
2. Connection Less Networking: It is based on UDP Protocol.

java.net package contains classes and interfaces those used to implements Networking in java.

1. ServerSocket: It is responsible for provide numbered port and its have following method those used to obtain reference of Socket class.

public Socket accept();

1. Socket: It is a special class those responsible to set or get data from network by the help of following methods:
2. getInputStream(): It is used to get data on Socket.
3. getOutputStream(): It is used to set data on Socket.

Example:

1. Server.java

import java.net.\*;

import java.io.\*;

import java.util.\*;

public class Server

{

public static void main(String args[])

{

try

{

System.out.println("Server is ready, waiting for request...");

ServerSocket server = new ServerSocket(6565);

Socket socket = server.accept();

BufferedReader br = new BufferedReader(new InputStreamReader(socket.getInputStream()));

PrintStream out =new PrintStream(socket.getOutputStream());

Scanner sc = new Scanner(System.in);

while(true)

{

String msg = br.readLine();

if(msg.equals("bye"))

{

break;

}

System.out.println("Client message : "+msg);

System.out.print("Enter server response : ");

String str = sc.nextLine();

out.println(str);

System.out.println("Server has send response for client.");

}

System.out.println("Server is closing...");

Thread.sleep(2000);

server.close();

socket.close();

System.out.println("Server has closed.");

}

catch(Exception ex)

{

System.out.println(ex);

}

}

}

1. Client.java

import java.io.\*;

import java.util.\*;

import java.net.\*;

public class Client

{

public static void main(String args[])

{

try

{

System.out.println("Client is ready for sending request...");

Socket socket = new Socket("localhost",6565);

BufferedReader br = new BufferedReader(new InputStreamReader(socket.getInputStream

()));

PrintStream out = new PrintStream(socket.getOutputStream());

Scanner sc = new Scanner(System.in);

while(true)

{

System.out.print("Enter message for server : ");

String msg = sc.nextLine();

out.println(msg);

if(msg.equals("bye"))

{

break;

}

System.out.println("Client has send request, wait for response.");

String str = br.readLine();

System.out.println("Server response is : "+str);

}

System.out.println("Client is closing...");

Thread.sleep(2500);

socket.close();

System.out.println("Client has closed.");

}

catch(Exception ex)

{

System.out.println(ex);

}

}

}

Serialization: It is a process to convert objects into bytes. When bytes convert into object its called Deserialization.

Serialization is implemented by Serializable interface.

Serializable interface is a marker interface those does not contains any method or variables but those class objects want to serialized then must be implements Serializable interface.

Following classes are used to implements Serialization:

1. ObjectOutputStream: It is used to write objects into bytes.
2. ObjectInputStream: It is used to read objects from bytes.

Example:

1. Student.java

import java.io.\*;

public class Student implements Serializable

{

String name,course;

int age;

public Student(String name, int age,String course)

{

this.name=name;

this.course=course;

this.age=age;

}

public void show()

{

System.out.println("Hi "+name+", you are only "+age+" years old. Your course is : "+course);

}

}

1. SaveObject.java

import java.io.\*;

public class SaveObject

{

public static void main(String args[])

{

try

{

Student s1 = new Student("Rahul",22,"Core Java");

Student s2 = new Student("Noor",21,"Advance Java");

Student s3 = new Student("Sachin",20,"PHP");

FileOutputStream fo =new FileOutputStream("friday.data");

ObjectOutputStream oos =new ObjectOutputStream(fo);

oos.writeObject(s1);

oos.writeObject(s2);

oos.writeObject(s3);

System.out.println("Object has successfully saved.");

}

catch(Exception ex)

{

System.out.println(ex);

}

}

}

1. ReadObject.java

import java.io.\*;

public class ReadObject

{

public static void main(String args[])

{

try

{

FileInputStream fin = new FileInputStream("friday.data");

ObjectInputStream oin = new ObjectInputStream(fin);

Student s1 =(Student)oin.readObject();

Student s2 = (Student)oin.readObject();

Student s3 = (Student)oin.readObject();

s1.show();

s2.show();

s3.show();

}

catch(Exception ex)

{

System.out.println(ex);

}

}

}

Collection Framework:

Framework is a set of predefine classes and interface. It is always domain specific.

API is also set of predefine classes and interfaces but API is mandatory whereas Framework are optional.

Collection Framework is used to implements Data Structure in java.

Collection can be divided following 3 parts:

1. List: It contains data in index bases.
2. Set: It contains only unique element, it does not support duplicate data.
3. Map: It contains data in key and value pair.

Basically Collection is an interface those contains following methods:

1. add(): It is used to add object in Collection.

public boolean add(Object o);

1. addAll(): It is used to add all objects of another Collection.

public boolean addAll(Collection c);

1. size(): It is used to find out number of objects in Collection.

public int size();

1. contains(): It is used to search object from Collection.

public boolean contains(Object o);

1. containsAll(): It is used to search all objects of another Collection in current Collection.

public boolean containsAll(Collection c);

1. remove(): It is used to remove object from Collection.

public boolean remove(Object o);

1. removeAll(): It is used to remove all objects for another Collection from current Collection.

public boolean removeAll(Collection c);

1. iterator(): It is used to obtain reference of Iterator interface.

public Iterator iterator();

1. clear(): It is used to remove all objects from Collection.

public void clear();

etc…

Following diagram represent Collection Framework:

List: List is an interface those extends Collection interface. Implementation of List interface provided by ArrayList, Vector and LinkedList classes.

ArrayList and Vector both class contains common methods but ArrayList and LinkedList classes and non-synchornized methods those can’t be used in multithread environment whereas Vector class have synchronized method those can be used in multithread environment.

List interface extends ListIterator those used to traverse data in bi-direction.

Example:

import java.util.\*;

public class MyList

{

public static void main(String args[])

{

List asia = new ArrayList();

asia.add("India");

asia.add("China");

asia.add("Japan");

asia.add("Nepal");

List europ = new Vector();

europ.add("England");

europ.add("France");

List who = new LinkedList();

who.add("USA");

who.add("Rusia");

who.addAll(asia);

who.addAll(europ);

who.remove("Japan");

System.out.println("All countries in who : "+who.size());

Iterator i = who.iterator();

while(i.hasNext())

{

System.out.println("Country name is : "+i.next());

}

}

}

Using ListIterator:

import java.util.\*;

public class MyList

{

public static void main(String args[])

{

List asia = new ArrayList();

asia.add("India");

asia.add("China");

asia.add("Japan");

asia.add("Nepal");

List europ = new Vector();

europ.add("England");

europ.add("France");

List who = new LinkedList();

who.add("USA");

who.add("Rusia");

who.addAll(asia);

who.addAll(europ);

who.remove("Japan");

System.out.println("All countries in who : "+who.size());

ListIterator i = who.listIterator(who.size());

while(i.hasPrevious())

{

System.out.println("Country name is : "+i.previous());

}

}

}

Set interface: Set interface is child interface of Collection framework. It contains only unique elements. Implementation of Set interface provided by HashSet and TreeSet classes.

Example:

import java.util.\*;

public class MySet

{

public static void main(String args[])

{

Set set = new TreeSet();

Scanner sc = new Scanner(System.in);

while(true)

{

System.out.print("Enter your city name : ");

String city = sc.nextLine();

city =city.trim();

if(city.equals("end"))

{

break;

}

set.add(city);

}

System.out.println("Total cities : "+set.size());

System.out.println(set);

}

}