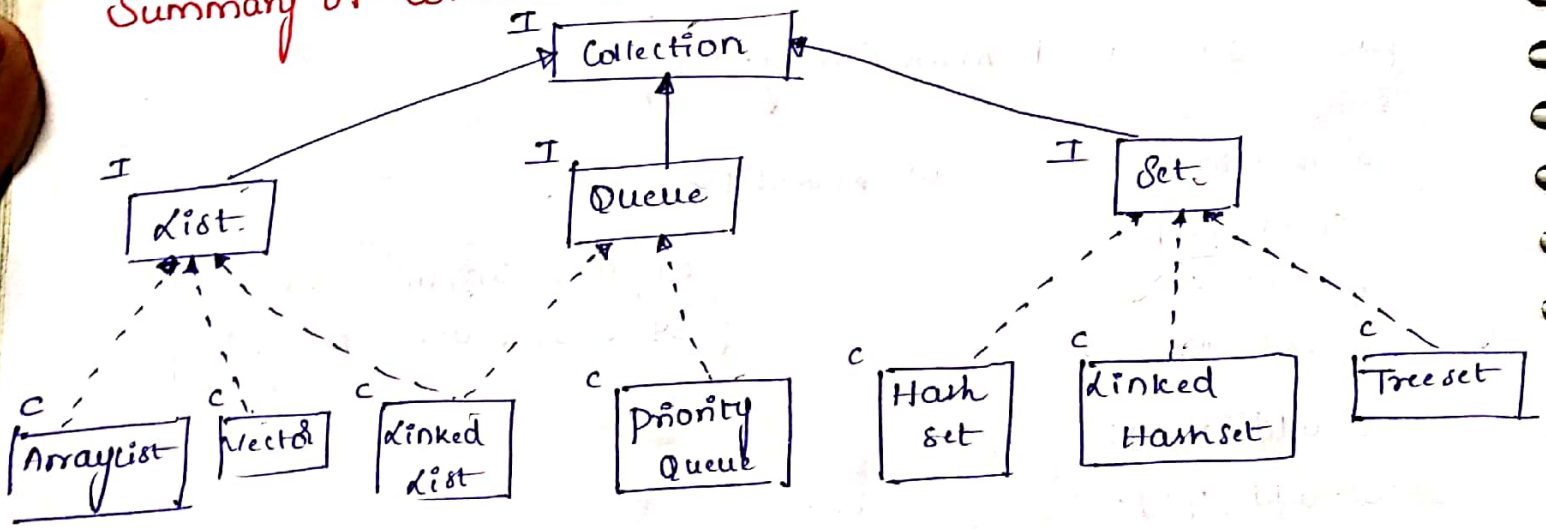
Prog:
```
public class Run
{
  public static void main(String[] args)
  {
    TreeSet s1 = new Treeset();
    s1.add(23);
    s1.add(45);
    s1.add(13);
    s1.add(97);
    Iterator i1 = new s1.iterator();
    while(i1.hasNext())
    {
      s.o.pln(i1.next());
    }
  }
}
```

output :- Sorting order.
[13, 23, 45, 97]

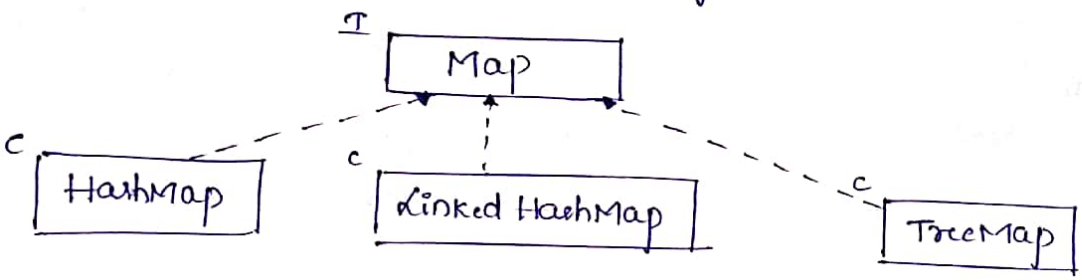
Summary of Collection :-



Date : 30/04/19

Maps :-

* Map is not a type of collection rather map itself is collection. map is not a type of collection means it doesnot inherit from an interface Collection rather it itself is a collection.
* There are three types of Map
  (i) HashMap → unordered
  (ii) LinkedHashMap → Insertion order
  (iii) TreeMap → sorting order.



* Incase of Map, we should always insert in the form of "key-value pair"
* Key acts like index for value.
* Value is our element
* We can retrive the value with the help of Key.

Prog:

```
Package demopack1
import java.util.HashMap;
public class Demo
{
  public static void main(String[] args)
  {
    HashMap m1 = new HashMap();
    m1.put(1, new student("abhi", 67.1));
    m1.put(2, new student("raja", 54.9));
    m1.put(3, new student("vijay", 75.1));
```

```
        S.O.pln(m1);
        SO.pln(m1.keySet());
        S.O.pln(m1.values());
        S.O.pln(m1.get(2));
        S.O.pln(m1.containsKey(7));
        S.O.pln(m1.containsValue(new Student("rani", 67.2)));
        S.O.pln(m1.remove(3));
        S.O.pln(m1);
    }
}
```

Output :-

{ 1 = Student

Prog:-

```
package demopack;
import java.util.TreeMap;
public class Demo1
{
    public static void main(String[] args)
    {
        TreeMap<String, Integer> tl = new TreeMap<String, Integer>();
        tl.put("mango", 56);
        tl.put("apple", 40);
        tl.put("orange", 23);
        tl.put("lemon", 45);
        SO.pln(tl);
    }
}
```

Output :-

{ apple = 40, lemon = 45,
mango = 56, orange = 23 }

Sorted order based on key.

* In case of TreeMap Sorting will happen always based on Key.

Exception Handling :-

* It is an unexpected event that is can occured during Runtime
* If exception is not handle, program gets terminated abruptly.

Note :- Exception occurs only during runtime but not during Coding time or Compile time.

* Exception is an object which gets created, when JVM encounter dangerous statement. [a statement which shows abnormal behaviour].
* For every exception object there is a corresponding Inbuilt class.
* Exceptions are handled by using try and catch block.
* catch block gets executed when there is an exception in try block.

Prog:

```
public class Sample
{
  public
  {
    S.O.pln ("main starts");
    int a = 10;
    int b = 0;
    try
    {
      S.O.pln (a/b);   // dangerous statement [exception].
    }
    catch (ArithmeticException ae)
    {
      S.O.pln ("Exception is caught");
    }
    S.O.pln ("main ends");
  }
}
```

* catch block gets executed when there is a exception in try block. If the exception not present in try block catch block cannot executed. Exception can also be called as dangerous statement.

* catch block contains the reference variable of exception class because exception is an object.

01/05/19

Prog:- ArrayIndexOutofBounds Exception program.

```
Package demopack3
// import java.util.*;
  public class Sample
  {
    public static void main (String[] args)
    {
      S.O.pln("main starts...");
      int[] a1 = new int[4];
      try
      {
        a1[5] = 30;
      }
      catch(ArrayIndexOutofBounds Exception rv)
      {
        S.O.pln("exception is caught");
      }
      S.O.pln("main ends...");
    }
  }
```

Above two example programs are unchecked exception

Prog

```
package demopack1
public class Samples
{
  public static void main (String[] args)
  {
    S.O.pln ("main starts ...");
    try
    {
      Keypad display();
    }
    catch ( InterruptedException xv )
    {
      S.O.pln (" Exception is caught ");
    }
    S.O.pln ( "main ends... ");
  }
  public static void display() throws InterruptedException
  {
    for ( int i = 1 ; i <= 5 ; i++ )
    {
      S.O.pln (i);
      Thread. sleep (2000);
    }
  }
}
```

output:—

```
main starts
1
2
3
4
5
main ends
```

* Throws is a keyword, throws indicates current method will not handle exception rather calling method will handle exception
* use 'throws' if current method doesnot have enough information to handle exception
* use try and catch if current method has enough information to handle exception.
* Above program is example for checked exception.

Prog:-

```
Package demopack
public class sample3
{
  public static void main (String[] args)
  {
    S.O.pln ("main starts ...");
    int i = 10;
(i) int j = 0;      (ii) int j = 2;
    int [] a1 = new int[4];
    try
    {
      S.O.pln (i/j);
      a1[6] = 50;
    }
    catch (ArrayIndexOutofBound exception rv) // Specialited catch block.
    {
      S.O.pln ("1st catch block...");
    }
    catch (Arithemetic exception rv)  // specialited catch block.
    {
      S.O.pln ("2nd catchblock....");
    }
    S.O.pln ("main ends...");
  }
}
```

(i) output:-

```
main starts
2nd catch block
main ends
```

(ii) output

```
main starts
5
1st catch block
main ends.
```

→ Single try block can have more than one catch blocks. (multiple catch blocks) (Specialized)

→ In the try block

Note:- If any statement causes Exception in "try block", further statements will not be executed, rather control ejects from try block and reaches corresponding catch block.

Prog:-

```
package demopack1;
public class sample1
{
 public static void main (String[] args)
 {
   S.O.println ("main starts ...");
   int i = 10;
(i) int j = 2;                    (ii) int j = 0;
   int[] a1 = new int[4];
   try
   {
    S.O.println (i/j);
    a1[6] = 50;
   }
   catch (Exception ni)     // Generalized catch block.
   {
    S.O.println ("Inside catch");
   }
   S.O.println ("main ends");
 }
}
```

Output:-

(i) main starts
5
Inside catch
main ends

(ii) main starts
Inside catch
main ends.

Single try block can have multiple specialized catch blocks
(or)
single try block can have single Generalized catch block.

Prog:-

```
package demopack1;
public class sample1
{
 public static void main (String[] args)
 {
    S.O.println ("main starts ...");
    int i = 10;
 (i) int j = 2;                    (ii) int j = 0;
    int[] a1 = new int[4];
    try
    {
      S.O.println (i/j);
      a1[6] = 50;
    }
    catch (Exception xv)     // Generalized catch block.
    {
      S.O.println ("Inside catch");
    }
    S.O.println ("main ends");
 }
}
```

Output:-

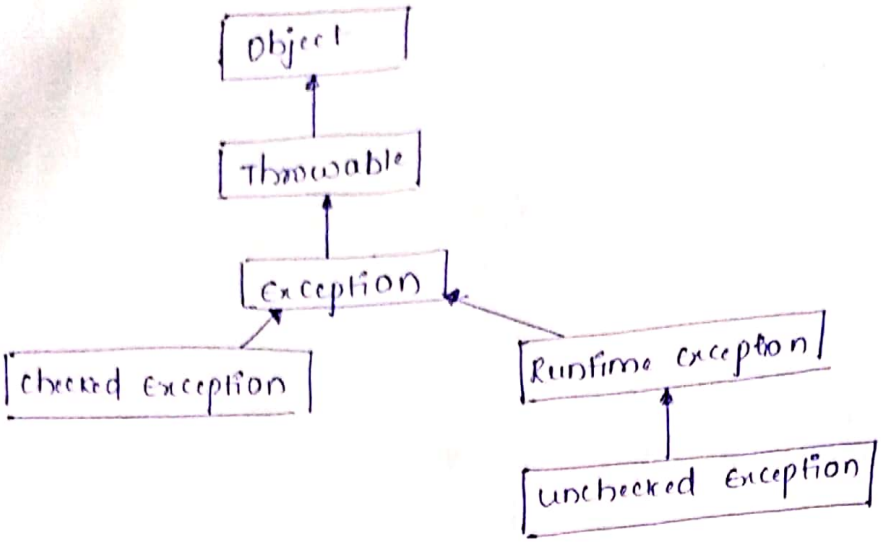
(i) main starts
5
Inside catch
main ends

(ii) main starts
Inside catch
main ends.

Single try block can have multiple specialized catch blocks

(or)

Single try block can have single Generalized catch block.



* All checked Exceptions are subclass to Exception class.
* All unchecked exceptions are subclass to Runtime exception.
* In checked exception compiler will check the identify the dangerous statement and gives warning.
* In unchecked exception Compiler will not identify the dangerous statement, programmer will identify the dangerous statements which can cause on unchecked exception.

finally() :-

* finally block should be used with "try and catch" (or) "try".
* finally block gets executed irrespective of exception.
* termination statement [database termination, Server termination, network termination etc...] should get inside finally block.

prog:-

```
package demopack
public class sampley
{
  public static void main (String[] args)
  {
```

```
S.o.pln ("main starts");
int i = 10;
(i) int j = 2;          (ii) int j = 0;
try
{
  s.o.pln ("Inside try...");
  s.o.pln (i/j);
}
catch (Arithemetic exception ref)
{
  s.o.pln ("Inside catch");
}
finally
{
  s.o.pln ("Inside finally");
}
s.o.pln ("main ends");
}
}
```

Output :-

(i)
```
main starts.
Inside try
5
Inside finally
main ends
```

(ii)
```
main starts
Inside try
Inside catch
Inside finally
main ends.
```

02/05/19

user-defined Exception :-

* When inbuilt exception doesnot fullfil application requirement then we have to create our own exception. This is called user-defined Exception.

Steps to create Userdefined Exception :-

1. Create a class
2. class should be a type of throwable.
   (a) if class inherits from Exception class its a checked Exception.
   (b) if class inherits from Runtime Exception class its a unchecked exception.
3. where ever necessary create an object and used throw keyword to throwon the exception.

Prog1: package demopack1

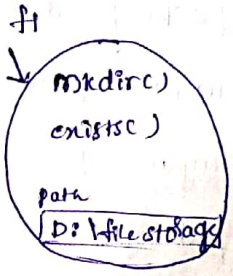
```
public class InvalidMonthNumberException extends RuntimeException
{
    public String tostring()
    {
        return "Entered monthnum is invalid";
    }
}
```

Prog2: this inbuilt exception is used in encapsulated calendar program.
throw keyword is used to

File Handling :-

* File Handling means,
  1. creation of folder
  2. creation of file
  3. checking the existance of folder & file.
  4. Deleting folder and file.
  5. Getting the filepath
  6. Reading data from file.
  7. Writing data to file . etc...

Prog:
```
Package demopack1
import java.io.File;
Public class Run
{
  Public static void main (String[] args)
  {
    File f1 = new File ("D://filestorage");
    boolean status1 = f1.mkdir();
    System.out.println (status1);
    boolean status2 = f1.exists();
    System.out.println (status2);
  }
}
```



Output:

1st time Run
True
True

2nd time Run
false, because already created.
True

Prog: Package demopack1;

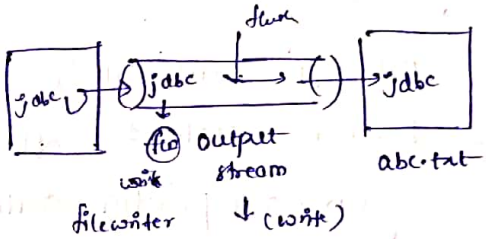
```
import java.io.File;
import java.io.IOException;
public class Run
{
  public static void main (String[] args) throws IOException
  {
(i) File f1 = new File ("D:/filestorage/xyz.txt");
    boolean status1 = f1.createNewFile();
    S.O.println(status1); // true o/p  // again new false o/p
(ii) f1.delete();
    boolean status2 = f1.exists();
    S.O.println(status2); // false o/p  // again false o/p
  }
}
```

(i) Run — True

(ii) Run — false

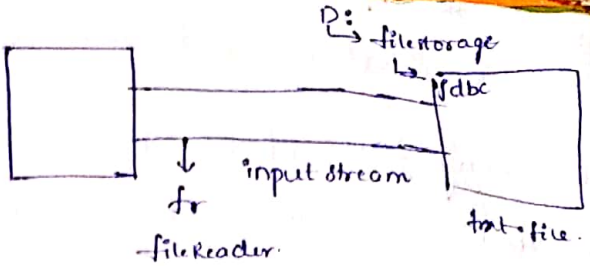
Prog:-

```
package demopack1;
import java.io.File;
import java.io.FileWriter;
public class Run2
{
public static void main(String[] args) throws Exception.
  {
    S.O.pln ("main starts");
    File f1 = new File ("D:/fileStorage/abc.txt");
    f1.createNewFile();
    FileWriter fw = new FileWriter(f1);
    fw.write("jdbc");
    fw.flush(); // it is used to flush the data from one file to another.
    fw.close(); // after copying the data close the file.
    S.O.pln ("main ends");
  }
}
```



output stream to txt file

03/05/19



Prog:
```
package demopack1
import java.io.File;
import java.io.FileReader;
public class Run3
{
  public static void main(String[] args) throws exception
  {
    File fl = new File("D:\filestorage\abc.txt");
    FileReader fr = new FileReader(fl);
    int size = (int) fl.length();
    char[] cArr = new char[size];
    fr.read(cArr);
    for(char ch : cArr) // foreach loop
    {
      s.o.pln(ch);
    }
    String s1 = new String(cArr);
    s.o.pln(s1);
    fr.close();
  }
}
```

output :—
```
j
d
b
c
jdbc
```

→ int size = (int) fl.length();:— length() can stores the value in long type but array can store value in integer type thats why we are converting length into int.

→ by using foreach loop data can be stored in letter by letter. (character by character)

→ by using String object we can store the data in string type.

File Reader is used to read the data in the txt file.

Serialization and De Serialization :-

* Saving state of an object within the file is called Serialization.
* Inorder to save state of an object, first of all object should be a type of Serializable. i.e., class should implement Serializable interface.
* Serializable interface is an empty interface

Note :- We are implementing to make difference of normal class and Serializable type of class

* Empty interfaces are called Marker Interfaces.
* If any datamember can be declared as transient, that datamembers state will not be same rather it will be reset to default value.
* When object is saved inside a file, it is automatically upcasted to object class type.
* Retrieving the state of an object from the file is called deserialization
* When object is retrieve from the file it has to be downcasted and use the members of object.

Prog :-

```
package demopackey
import java.io.Serializable;
public class Emp implements Serializable
{
  int empId;
  String empName;
  double empSal;     transient double empSal;
  public Emp(int empId, String empName, double empSal)
  {
    super();
    this.empId = empId;
    this.empName = empName;
    this.empSal = empSal;
  }
```

```
    public void empInfo()
    {
        S.O.pln("empId : "+this.empId);
        S.O.pln("emp Name : "+this.empName);
        S.O.pln("Emp Sal : "+this.empSal);
    }
}
```

Serialization program :—

```
package demopacky
import java.io.File;
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
public class sample1
{
    public static void main(String[] args) throws Exception
    {
        S.O.pln("main starts ...");
        Emp e1 = new Emp(23, "abhi", 6790.1);
        File f1 = new File("D:/filestorage/employee.ser");
        f1.createNewFile();
        FileOutputStream fos = new FileOutputStream(f1);
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(e1);
        oos.flush();
        oos.close();
        fos.close();
        S.O.pln("main ends ...");
    }
}
```

Deserialization program:-

```
package demopack4
import java.io.File;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
public class sample2
{
  public static void main(String[] args) throws Exception
  {
    S.O.pln("main starts ...");
    File f1 = new File("D:/filestorage/employee.ser");
    FileInputStream fis = new FileInputStream(f1);
    ObjectInputStream ois = new ObjectInputStream(fis);
    Emp e1 = (Emp) ois.readObject();
    e1.empInfo();
    ois.close();
    fis.close();
    S.O.pln("main ends ...");
  }
}
```

Output:-

```
main starts
Emp id : 23
Emp name : abhi
Emp sal : 0.0
main ends.
```

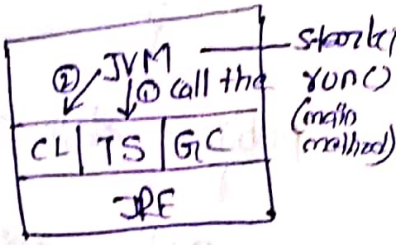
* File output stream is used to make the connection from file to employee object.
* Object output stream is used to send the object information to file with FOS.

* By default collections allows heterogenous data. Therefore we cannot apply sorting. If sorting has to be applied then collections must contain homogenous data.

* If I dont want to save the state of any datamember declare with the keyword transient (Reset to default value).

Multi Threadings:-

* A Thread is nothing but path to execute. (Stack area to execute)
* An inbuilt thread is created for main method.
* Creating a thread for user methods is called "User defined Thread"
* Stack area is given by Thread schedular.
* There are two ways to create User define Thread.
  1. By extending Thread class.
  2. By implementing Runnable Interface.



By Extending Thread class

Step1: extends Thread class.

Step2: Override run() method.

Step3: Call all the functionality from run() method.

Step4: Start the thread by calling start() method

By Implementing Runnable Interface

Step1: implements Runnable interface.

Step2: Override run() method.

Step3: Call all the functionality from run() method.

Step4: create an object of Thread class and pass runnable type of object to thread constructor. and then start the thread

Program By Using Thread class:

```
package demo.pack4;
public class Sample extends Thread
{
  public void run();
  {
   this.display();
```

```
    }
    public void display()
    {
    Sopln ("inside display method...");
    }
    }

public class Test
  {
    public static void main()
    {
    Sample s1 = new Sample();
    s1.start();
    }
  }
```

Program By Using Runnable Interface:

```
public class Demo implements Runnable
  {
    public void run()
    {
    this.display();
    }
    public void display()
    {
    Sopln ("inside display");
    }
    }
public class Test1
```

```
{
public static void main(String[] args)
  {
  Demo d1 = new Demo();
 Thread t1 = new Thread(d1);
   t1.start();
  }
}
```