24.04.23

```
package collectionDemo;

import java.util.HashSet;
import java.util.LinkedHashSet;
import java.util.Set;

public class MainClass
{
  public static void main(String[] args)
  {
    Set movies = new HashSet();
    // Set movies = new LinkedHashSet();
    movies.add(new String("Leo"));
    movies.add(new String("Leo"));
    movies.add("vadachennai");
    movies.add("Hangover");
    movies.add("physco");
    movies.add(null);
    movies.add(new StringBuffer("Legend"));
    movies.add(new StringBuffer("Legend"));
    for(Object o1 : movies)
    {
      System.out.println(o1);
    }
  }
}
```

```
Leo
vadachennai
Hangover
physco
null
Legend
Legend
```

```
package collectionDemo;

import java.util.LinkedHashSet;
import java.util.Set;

public class Mainclass10
{
  Set<Food> foods = new LinkedHashSet<Food>();
  foods.add(new Food(1,"Dosa", 40.0, "A2B", 1, "veg"));
  foods.add(new Food(2,"Pongal", 50.0, "saravana",
                                    2,"veg");
  foods.add(new Food(3,"karuvadu", 20.0, "KEB",
                                    "Non-veg"));
  for(Food f1 : foods)
  {
    S.O.println(f1);
  }
}
```

O/P:

Food[id=1, name=Dosa, price=40.0, hotel=A2B, qty=1, type=veg]

Food[id=2, name=Pongal, price=50.0, hotel=saravana, qty=2, type=non-veg]

Food[id=3, name=karuvadu, price=20.0, hotel=KEB, qty=1, type=non-veg]

```
package collectionDemo;

import java.util.Set;
import java.util.TreeSet;

public class Mainclass10
{
  public static void main(String[] args)
  {
    Set set = new TreeSet();
    set.add("raina");
    set.add("dhoni");
    set.add("kholi");
    set.add("dhoni");
    // set.add(null); // NullPointerException
    // set.add(new StringBuffer()); // ClassCastException
    S.O.println(set);
  }
}
```

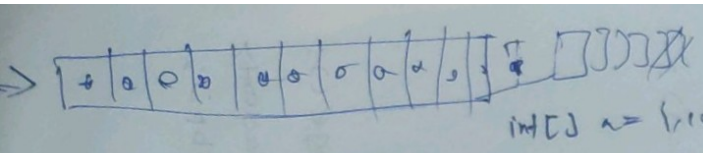
O/P:

[dhoni, kholi, raina]

```
package collectionDemo;

public class student implements comparable
{
   int id;
  String name;
  double perc;
  public student (int id, String name, double perc)
  {
    this.add = id;
    this.name = name;
    this.perc = perc;
  }
  @override
  public String toString()
  {
   return "student [id =" + id + ", name = " + name + ",
  }                    perc =" + perc + "]";
  public int hashcode()
  {
    return id;
  }
  public boolean equal (object o1)
  {
    return this.hashcode() == o1.hashcode();
  }
  @override
  public int compareTo (object o)
  {
    // sorting based on ID:
       return this.hashcode () - o.hashcode (); // ascending
    // return o.hashcode () - this.hashcode (); // descending
    // student std1 = (student) o;
    // sorting based on name:
    // return this.name.compareTo (std1.name);
    // sorting based on percentage:
    /* if (this.perc > std1.perc)
         return 5;
       else if (this.perc < std1.perc)
         return -5;
       else
         return 0; */
  }
}
```

```
package collectionDemo;

import java.util.Comparator;

public class SortByPercentage implements Comparator
{
@Override
public int compare (object o1, object o2)
{
  Student std1 = (Student)o1;
  Student std2 = (Student)o2;

  if (std1.perc > std2.perc)
      return 3;
  else if (std1.perc < std2.perc)
      return -4;
  else
      return 0;
  }
}
```

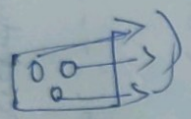
```
package collectionDemo;

import java.util.Set;
import java.util.TreeSet;

public class mainclass12
{
  public static void main (String[] args)
  {
    Set<Student> students = new TreeSet<Student>();

    students.add(new student (10, "kumar", 90.0));
    students.add (new student (7, "Akash", 95.0));
    students.add (new student (11, "Mohan", 96.0));
    students.add (new student (15, "lokesh", 98.0));
    students.add (new student (6, "Alex", 97.0));

    for (Student : students)
    {
      S.O.Pln (std);
    }

    S.O.Pln("----------");

    SortByPercentage sbp = new SortByPercentage();
    Set<Student> students1 = new TreeSet<Student>
    students1.addAll(students);                    (sbp);
    for (Student std : students1)
    {
      S.O.Pln (std);
    }
```



int[] a = {1,10,}

List l = new ArrayList() int t = 6 = new int[a.length+1

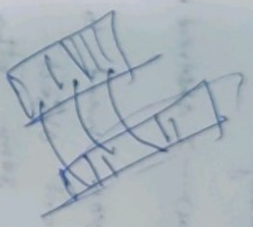
```
l.add("deva");
l.add("java");
l.add("venky");
for(String s1:l)
{
s.o.println(s1)
}
```

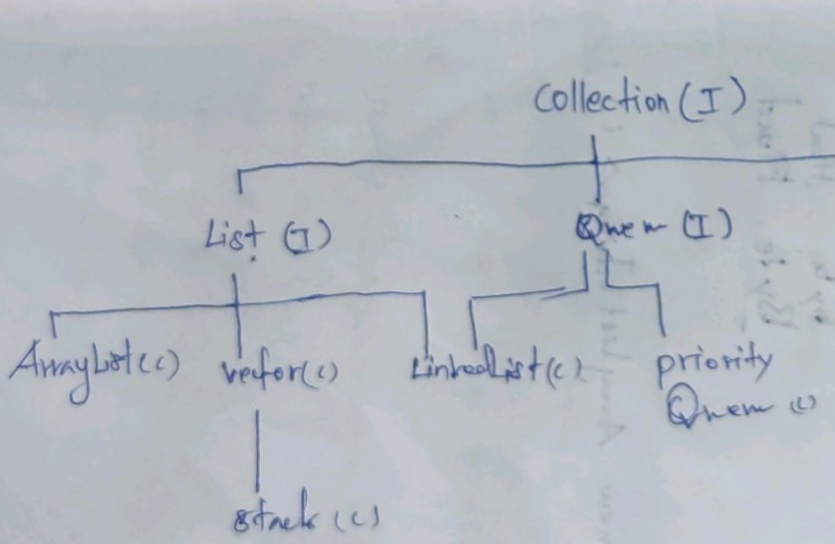


Map m = new HashTable<>

```
m1.put(1,"deva");
m1.add(2,"java");
```

Student [id = 6, name = Alex, perc = 97.0]
Student [id = 7, name = Dhoni, perc = 95.0]
Student [id = 10, name = Kumar, perc = 90.0]
Student [id = 11, name = Mohan, perc = 96.0]
Student [id = 18, name = Lokesh, perc = 98.0]
Student [ id = 10, name = Kumar, perc = 90.0]
Student [ id = 7, "
Student [ id =
Student [ id =
Student [ id =






- collection
- List (I)
- ArrayList (c)
- vector (c), stack (c)
- Linked List (c)
- priority Queue (c)
- Set (I)
- Hashset (c)

10, tree set (c)

comparable

comparable is a interface present in java.lang package

* compareto() → abstract method

* collection stores objects in sorted order based on comparable implementation

ClassCastException
NullPointerException

```
class Main
{
  p s v m
{
  List l1 = n
  List<Integer> l1 = new ArrayList<Integer>();
  l1.add(15);
  l1.add(10);
  l1.
```

comparator is a interface present in java.util package

compare() → abstract m...

* if you want to store sort object based on some other factor we must use comparator.

byte float
Byte Float

25.04.

package collectionDemo;

import java.util.Comparator;
import java.util.TreeSet;

```
Class DescendingOrder implements Comparator<Integer>
{
  public int compare(Integer i1, Integer i2)
  {
    if(i1 > i2)
      return -5;
    else if(i1 < i2)
      return 5;
    else return 0;
  }
}

public class Maincls1
{
  public static void main(String[] args)
  {
    TreeSet<Integer> nums = new TreeSet<>(new DescendingOrder());
    nums.add(10);
    nums.add(5);
    nums.add(3);
    nums.add(4);
    S.O.println(nums);
  }
}
```

[10, 5, 4, 3]

PRIORITY

```
package collectionDemo ;

import java.util.PriorityQueue ;

public class mainclass1
{
  public static void main (String[] args )
  {
    PriorityQueue pq = new PriorityQueue();
    pq.add ("lion");
    pq.add ("tiger");
    pq.add("dog");
    pq.add ("cat");
    pq.add ("lion");

  // pq.add (null) ;// NullPointerException
  // pq.add (new StringBuffer()); // ClassCastException

    S.O.pln (pq);
    S.O.pln ("Head element = "+pq.peek());

    Object o1 = pq.poll();
    While (o1 != null)
    {
      S.O.pln (o1);
      o1 = pq.poll();
    }
    S.O.pln (pq.size());
}
```

```
[cat, dog, lion, tiger, lion]
Head element = cat
cat
dog
lion
lion
tiger
0
```

MAP :

→ It is an interface present in java.util package.

→ It stores object in the form of key and value pair.

→ key and values both are objects.

→ key can't be duplicate but value can be duplicate.

* put (object key, object value)
* putAll (map m1)
* remove (object key)
* containsKey (object key)
* containsValue (object value)
* get (object key)
* keySet () [set keys]
* values ()
* entrySet () [set of key and value]
* size ()
* isEmpty ()
* clear ()

* implementation class HashMap, HashTable, TreeMap

```
package collectionDemo;

import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Set;

public class Mainclass15
{
    public static void main(String[] args)
    {
        Map<Integer, String> map = new HashMap
                                   <Integer, String>();
// Map<Integer, String> map = new Hashtable<Integer,
                              String>();
        map.put(1, "one");
        map.put(2, "two");
        map.put(3, "three");
        map.put(10, "one");
        map.put(null, null);

        // map.clear();
        // map.remove(2);
        S.o.println(map.isEmpty());
        S.o.println(map.size());
        S.o.println(map.get(3));
        S.o.println(map.getOrDefault(30, "Thirty"));
        map.replace(2, "Twooo");
        S.o.println(map.keySet());
        S.o.println(map.values());
        S.o.println(map.entrySet());
        S.o.println(map.containsKey(2));
        S.o.println(map.containsValue("Three"));
        S.o.println("----------");
        for(Entry<Integer, String> entry : map.entrySet())
        {
            S.o.println(
        Set<Integer> keys = map.keySet();
        for(Integer key : keys)
        {
            S.o.println(key + "-->" + map.get(key));
        }
        S.o.println("----------");
        for(Entry<Integer, String> entry : map.entrySet())
        {
            S.o.println(entry.getKey()+"-->"+entry.getValue());
        }
    }
}
```

o/p false
5
Three
Thirty
[null, 1, 2, 3, 10]
[null, one, twoo, three, one]
[null=null, 1=one, 2=twoo, 3=Three, 10=one]
true
true

---

null -> null
1 -> one
2 -> twoo
3 -> Three
10 -> one

---

null -> null
1 -> one
2 -> Twoo
3 -> Three
10 -> one

```
package CollectionDemo;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class Mainclass
{
  private static Map<String, String> users
      = new HashMap<String, String>();

  public static void main(String[] args)
  {
    users.put("root","root");
    users.put("scott","tiger");
    users.put("dinga","dingi");

    Scanna sc1 = new Scanner(System.in);
    S.O.plm("Enter User name:");
    String username = sc1.next();
    if (users.containsKey(userName))
    {
      S.O.plm("Enter the password..!!");
      String password = sc1.next();
      String actualPassword = users.get(userName);
```

```
if (actualPassword.equals(password)
{
  s.o.pln("Welcome" + userName);
}
else
  {
   s.o.pln("Invalid password!");
  }
}
else
  {
   s.o.ph("Invalid userName!");
  }
}
```

o/p

Enter user Name :
root
Enter the passcard..!!
root
Welcome root

---

```
package CollectionDemo;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import java.util.TreeSet;
```

```
public class Mainclass
{
  public static void main(String[] args)
  {
    int[] arr = {5, 3, 8, 1, 7};
    Arrays.sort(arr);
    s.o.pln(Arrays.toString(arr));
    s.o.pln(Arrays.binarySearch(arr, 8));

    List<Integer> nums = Arrays.asList(10, 3, 7, 1, 8);
    s.o.pln(nums);

    // create to TreeSet
    // TreeSet<Integer> nums1 = new TreeSet<>(nums);
    // s.o.pln("nums1");

    Collections.sort(nums);
    s.o.pln(nums);
  }
}
```

```
package ExceptionHandling ;

public class Mainclass1
{
  public static void main (String[] args)
  {
    int n1 = 10;
    int n2 = 0 ;
    int n3 = 0 ;
    try
    {
      n3 = n1/n2 ;
    }
    catch ( Exception e)
    {
      S.o.pln ("Execution Solution...");
      n3 = n1/1 ;
    }

    S.o.pln ("n1 value = "+n1);
    S.o.pln (" n2 value = "+n2);
    S.o.pln ( "n3 value = " +n3);
}
```

O/P

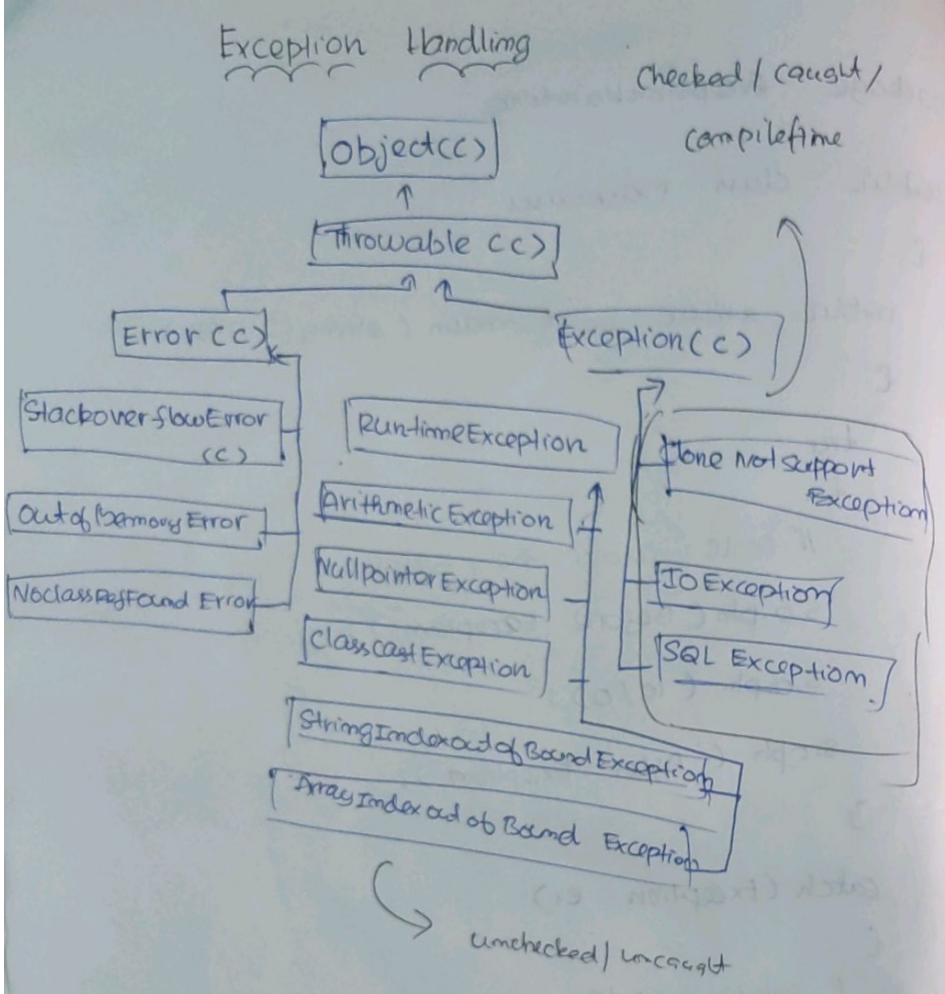
```
Executing solution....
n1 value = 10
n2 value = 0
n3 value = 10
```

```
package ExceptionHandling

public class Mainclass1
{
  public static void main (String[] args)
  {
    try
    {
      // code causes the Exception
      S.o.pln ("Before Exception");
      S.o.pln (10/0);
      S.o.pln ("After Exception");
    }
    catch (Exception e1)
    {
      // solution code
      S.o.pln (" I am in catch()....");
    }
    finally
    {
      // mandatory code
      S.o.pln (" I am in finally");
    }
}
```

O/P

```
before Exception
I am in catch()..
I am in finally()...
```

# Exception Handling




```
package exceptionHandling ;

public class mainclass 3
{
  static int test()
  {
    try {
        S.O.ptln (10 / 0) ;
        return 1 ;
      }

    catch ( Exception e1)
    {
      return 2 ;
    }
    finally
    {
      return 3 ;
    }
    // return 4 ; // error
}
  static void demo()
  {
    demo() ;
  }

  void show()
  {
    try {
        this . clone() ;
      }
    catch (Exception.. e1)
    {
```

```
public static void main(String[] args)
{
  S.o.pln (test());
}
  // demo();
}
}
```

O/P:

3

## TRY WITH MULTIPLE CATCH BLOCK

```
package exceptionHandling

public class Mainclass4
{
  public class static void main (String[] args)
  {
    try
    {
      S.o.pln (10/1);
      String s1 = "abc";
      S.o.pln (s1.charAt (10));
    }
    catch (ArithmeticException | NullPointerException e1)
    {
      S.o.pln ("I am in AE|NPE ...");
    }
    catch (RuntimeException e1)
    {
      S.o.pln ("I am in RuntimeException..");
    }
```

```
    catch (Exception e1)
    {
      S.o.pln ("I am in Exception");
    }
```

O/P

10

I am in RuntimeException

27-04-23

## Default Exception Handling

```
Package Exception Handling

public class Mainclass2
{
  public static void main (String[] args)
  {
    S.o.pln ("*********");
    m1();
    S.o.pln ("*********");
  }
  private static void m1()
  {
    m2();
  }
  private static void m2()
  {
    S.o.pln (10/0);
  }
}
```

O/P:

Exception in thread "main" java.lang.ArithmeticException: / by zero
at Exception Handling.Mainclass2.m2(Mainclass2.java:15)
at Exception Handling.Mainclass2.m1(Mainclass2.java:10)
at Exception Handling.Mainclass2.main(Mainclass2.java:5)

Exception propagation → once the exception is occured in the programm it will ↑ going towards the bottom of stack.

e. printStackTrace - to print Exception ;

## Customized Exception.

```
package ExceptionHandling

public class InvalidAmountException extends Exception
{
  InvalidAmountException()
  {
  s.o.pln ("InvalidAmountException object is created");
  }

  void check()
  {
   s.o.pln (" Please Enter valid amount ");
  }
}
```

---

```
package ExceptionHandling

public class MoneyTransferApp
{
  public static void main(String[] args)
  {
    try
     {
       transfer (-10000.0)
     }
    catch ( InvalidAmountException e)
    {
      e.check();
} }
```

// 2 ways we can handle Exception

1. by try & catch block
2. by using throws keyword

```
  private static void transfer (double amt)
*/ {
    if (amt > 0)
    {
      s.o.pln (" Transaction started");
    }
    else
    {
      try
       {
        throw new InvalidAmountException();
       }
      catch (InvalidAmountException e1)
       {
         e1.check();
       }
    }
} */
```

```
private static void transfer (double amt)
                    throws InvalidAmountException
{
   if(amt > 0)
   {
     s.o.pln (" transaction started ");
   }
```

```
        else
        {
            throw new InvalidAmountException();
        }
    }
}
```

Output:

InvalidAmountException objed is created
Please enter valid amount.

---

# THREADS

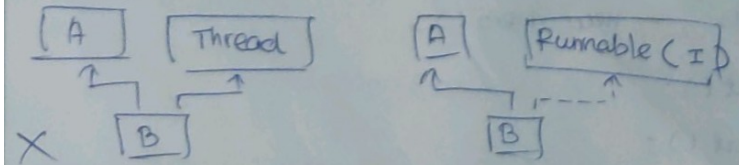
```
package threadDemo;

class Hero extends Thread
{
    public void run()
    {
        for (int i=1 ; i<=10 ; i++)
        {
            S.O.pln ("Hero" + i);
        try
            {
                thread.sleep(1000);
            }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
    }
}
```

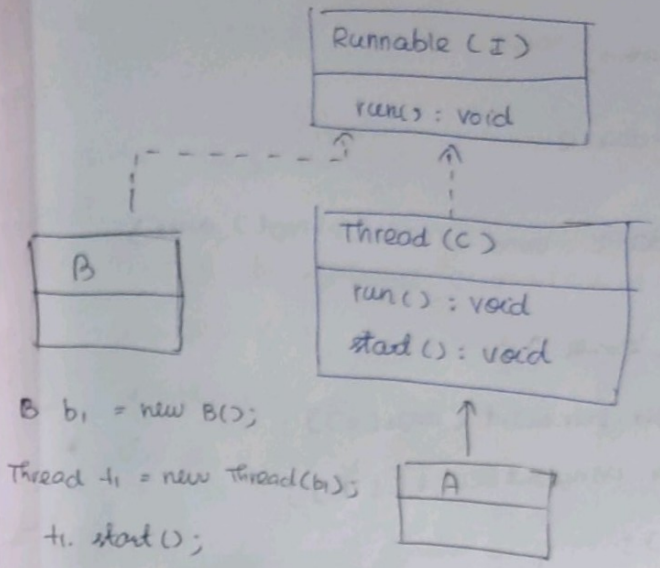
```
Class Demon implements Runnable
{
    @override
    public void run()
    {
        for (int j = 100; j<110 ; j++)
        {
            S.O.pln ("j value = " + j);
        try{
                thread.sleep(1000);
            }
        catch (Interrupted Exception e)
            {
                e.printStackTrace();
            }
        }
    }
}

public class Mainclass
{
    public static void main (String[] args)
    {
        Hero h1 = new Hero();
        Hero h2 = new Hero();

        h1.start();
        h2.start();

        Demon d1 = new Demon();
        Demon d2 = new Demon();
```

```
Thread t1 = new Thread(d1);
Thread t2 = new Thread(d2);

t1.start();
t2.start();

}
}
```

O/P

```
Hero 1
Hero 1
j value 101
j value 101
Hero 2
Hero 2
j value 102
j value 102
.
.
.
```

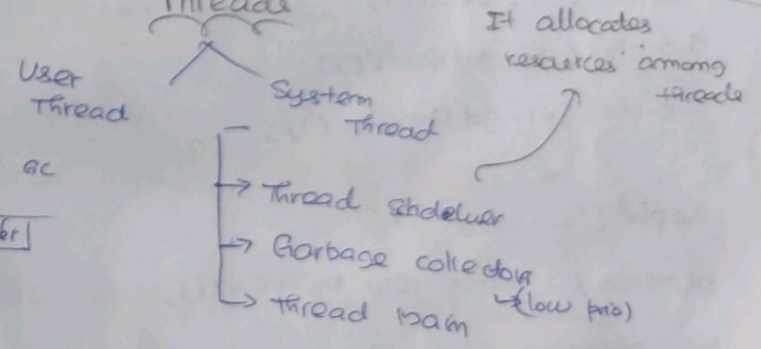



```
B b1 = new B();
Thread t1 = new Thread(b1);
t1.start();

A a1 = new A();
a1.start();
```

08.04.23

# Threads




t1 t2 t3 GC

threadScheduler

Resource

```
Class A
{
  public static void main(String[] args)
  {

  }
}
```

> java A 10 20 abc

Command line arguments: "10" "20" "abc"

```
package ThreadDemo;

public class mainclass2
{
  public class static void main(String[] args)
  {
    s.o.pln(args.length);
    int x = Integer.parseInt(args[0]);
    int y = Integer.parseInt(args[1]);
    s.o.pln(x+y);

    s.o.pln("----------");
    Thread t1 = Thread.currentThread();
    s.o.pln(t1.getId());
    s.o.pln(t1.getName());
    s.o.pln(t1.getPriority());

    s.o.pln("------");
    MyThread mt1 = new MyThread();
    mt1.setName("child");
    mt1.setPriority(Thread.MIN_PRIORITY);
    mt1.start();
  }
}
```

```
Class MyThread extends Thread
{
  public void run()
  {
    Thread t1 = Thread.currentThread();
    s.o.pln(t1.getId());
    s.o.pln(t1.getName());
    s.o.pln(t1.getPriority());
  }
}
```

```
o/p
3
30
--- ---
1
main
5
------
20
Child
1
```

CHUMA INTERVIEW

```
package ThreadDemo;
public class Mainclass
{
  // public static void main(String[] args)
  // private static void main(String[] args) // won't work
  // protected static void main(String[] args) // wont
  // public void main(String[] args) Won't
  // static public void main(String[] args) work
  // public void static main(String[] args) // error.
  // public static int main(String[] args) // won't work
  public static void Main(String[] args) // won't work
```

// public static void main (int [] args) // won't work

// public static void main (String [] xyz) // Works.

```
public static void main (String [] args)
{
  for (int h=0; h<24 ; h++)
  {
    for (int m=0 ; m<60; m++)
    {
      for (int s=0 ; s<60 ; s++)
      {
        s.o.p.ln ("%02d : %02d : %02d (h", h, m, s);
        try
        {
          Thread . sleep (1000);
        }
        catch(
          InterruptedException e);
        {
          e. printStackTrace ();
        }
      }
    }
  }
```

o/p

00 : 00 : 01

:

:

Thread safety :

If multiple thread modifying objects it leads to data inconsistency.

Instead of allowing multiple threads to access object we will allow only one thread to access object this is known as thread safety

2 ways we can achieve thread safety

① develop methods with synchronized.

② develop class as immutable.

Multiple threads are waiting for same shared resource infinitely then it is known as deadlock

We can solve deadlock by Inter thread communication methods :

wait ()

on notify ()

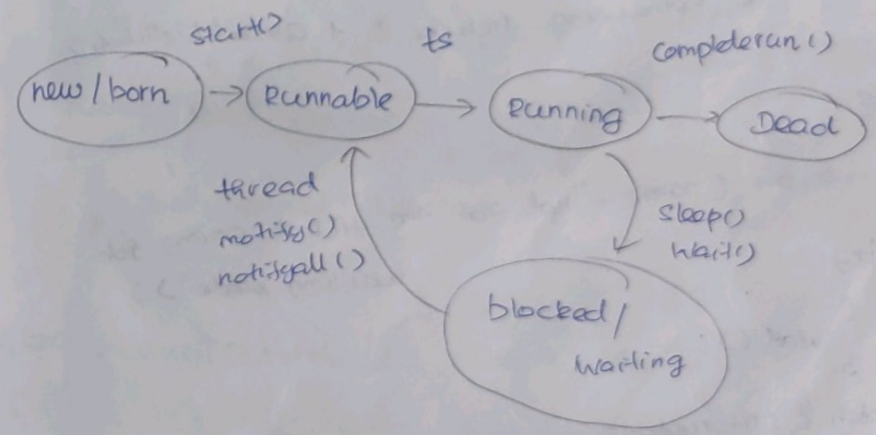
notifyall ()

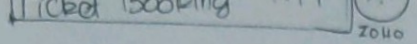
```
package threadDemo;

class Wish
{
  public synchronized static void great()
  {
    String S1 = "Good Afternoon...!!";
    for(int i=0; i< S1.length(); i++)
    {
      S.O.println(S1.charAt(i));
      try
      {
        Thread.sleep(500);
      }
      catch (InterruptedException e)
      {
        e.printStackTrace();
      }
    }
    S.O.println();
  }
}

class WishThread extends Thread
{
  public void run()
  {
    Wish.great();
  }
}
```

```
public class MainClass
{
  public static void main(String[] args)
  {
    WishThread wt1 = new WishThread();
    WishThread wt2 = new WishThread();
    WishThread wt3 = new WishThread();
    wt1.start();
    wt2.start();
    wt3.start();
  }
}
```

op

Good Afternoon...!
Good Afternoon...!
Good Afternoon...!






```
ckage ThreadDemo;

ss tickets

int availableTickets = 2;

public synchronized void bookTicket()
{
 String name = Thread.currentThread().getName();
 S.o.pln("name +" is trying to book the ticket");
 if(availableTickets > 0)
 {
  S.o.pln(name + " is got the ticket");
  availableTickets -= 1;
  S.o.pln("remaining tickets =" + availableTickets)
 }
 else
 {
   S.o.pln(name + "not got the ticket going for
                                waiting state");
   try
    {
      wait();
    }
   catch(InterruptedException e)
    {
      e.printStackTrace();
    }
   bookTicket();
 }
}
```

```
public synchronized void cancelTicket()
{
  String name = Thread.currentThread().getName();
  S.o.pln(name + "is cancelling 3 tickets");
  availableTickets += 3;
  notifyAll();
}

Class BookTicketThread extends Thread
 {
   Tickets tickets;
   BookTicketThread(Tickets tickets)
    {
      this.tickets = tickets;
    }
    public void run()
    {
      tickets.bookTicket();
    }
}

class CancelTicketThread extends Thread
{
    Tickets tickets;
    CancelTicketThread(Tickets tickets)
    {
      this.tickets = tickets;
    }
```

```
    public void run()
    {
      tickets.cancelTicket();
    }
}

public class TicketBooking
{
    public static void main(String[] args)
    {
      Tickets t1 = new Tickets();

      BoobTicketThread btt1 = new BoobTicketThread(t1);
      BoobTicketThread btt2 = new BoobTicketThread(t2);
      BoobTicketThread btt3 = new BoobTicketThread(t3);
      BoobTicketThread btt4 = new BoobTicketThread(t1);
      BoobTicketThread btt5 = new BoobTicketThread(t1);

      cancelTicketThread ctt1 = new cancelTicketThread(t1);

      btt1.setName("btt1");
      btt2.setName("btt2");
      btt3.setName("btt3");
      btt4.setName("btt4");
      btt5.setName("btt5");

      ctt1.setName("ctt1");

      btt1.start();
      btt2.start();
      btt3.start();
      btt4.start();
      btt5.start();

      try{
          Thread.sleep(1000*60*2);
      }
      catch(InterruptedException e)
      {
        e.printStackTrace();
      }
      ctt1.start();
    }
}
```

Output

```
btt1 is trying to boob the ticket....
btt1 is got the ticket....
Remaining tickets = 1

btt3 is trying to book the ticket.
btt3 is got the ticket.
Remaing tickets = 0

btt2 is trying to book the ticket.
btt2 is not got the ticket going for waiting state

btt5 is trying to boob the ticket
btt5 is not got the ticket going for waiting state

btt4 is trying to book the ticket
btt4 is not get the ticket going for waiting state

ctt1 is cancelling the tickets
btt2 is trying to book the ticket
```

bth4 is trying to book the ticket

bth4 is got the ticket

Remaining ticketts = 1

bth5 is trying to book the ticket

bth5 is got the ticket

Remaing ticketts = 0.

---

01.05.23

Sreadikap.

```
package fileHandling;

import java.io.File;

Public class mainclass1
{
  public static void main(String[] args)
  {
    File fi = new File("C:\\users\\@spiders\\Desktop\\abc");
    if (fl.exists()).
    {
      S.o.pln ("Deleteling old folder and creating new folder");
      fi.delete();
      fi.mkdir();
    }
    else
    {
      S.o.pln("creating new folder");
      fl.mkdir();
    }
  }
}
```

o/p

Deletimd old folder &
Creating new folder

---

```
package fileHandling

import java.io.file;
import java.io.IOException;

public class mainclass2
{
  Public static void main(String[] args)
  {
    File fi = new File("C:\\Users\\@spiders\\Desktop\\abc\\data.txt");
    try {
      fl.CreateNewfile();
      S.o.pln (fi.canRead());
      S.o.pln (fi.canWrite());
      S.o.pln (fi.canExecute());
      S.o.pln (fl.getAbsolutePath());
      S.o.pln (fi.getCanonicalPath());
      fi.setWritable (false);
      S.o.pln ("done");
    }
```

# Serialization:

→ process of writing object into file is known as serialization.

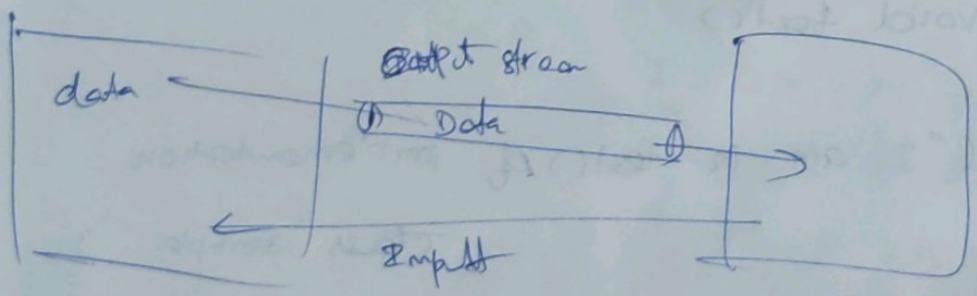
→ process of reading object from the file is known as deserialization.

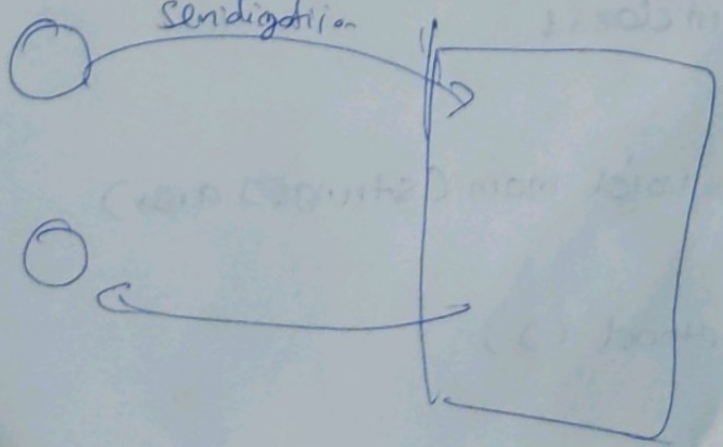
→ We can serialize object only if it is of type serializable

→ Serializable is a marker interface present in java.io package.

→ the following members can't be serialized

1. Static members
2. transient members



# JDK 8 Features

```
Package javafeatures;

interface Demo
{
    void test();
    default void defaultMethod()
    {
      S.o.ptln ("I am in defaultMethod");
    }
    static void staticMethod()
    {
      S.o.ptln ("I am in staticMethod");
    }
}

class sample implements Demo
 {
    public void test()
    {
      S.o.ptln("I am in test() of implementation
                                class sample...");
    }
}

public class mainclass1
{
    public static void main (String[] args)
    {
      Demo.staticMethod ();
      Demo d1 = new sample();
      d1. defaultMethod ();
      d1. test()
      Demo d2 = new Demo()        // Annonymous inner
      {                                  class
          public void test(){
              S.o.ptln ("I am in test() of Anno. Inner class");
          }
      };
      d2. test ();
      Demo d3 = () -> S.o.ptln ("I am in test() of
      d3. test();                        lamda Expression");
    }
}
```

O/P:

```
I am in static method
I am in defaultmethod
I am in test() of Implementation
                  class sample
I am in test() of Annonymous
                  Inner class
I am in test() of lamda
                  Expression
```

```
package javafeatures;

interface A
{
   int square(int n);
}
interface B
{
  boolean isEven(int n);
}
@FunctionalInterface
interface calci
{
  double add(double num1, double num2);
}
public class Mainclass2
{
  public static void main(String[] args)
  {
    A a1 = (e)->e*e
    S.o.println(a1.square(5));
    B b1 = (e)->e*2==0;
    S.o.println(b1.isEven(30));
    Calci c1 = (e1, e2)->e1+e2;
    S.o.println(c1.add(10.5, 22.2));
```

op

```
25
true
32.7
```

```
package javafeatures;

import java.util.Arrays;
import java.util.Collection;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

public class Mainclass3
{
  public static void main(String[] args)
  {
    Runnable r1 = ()->S.o.println(Thread.currentThread().
    new Thread(r1).start();                    getName());

    List<Integer> nums = Arrays.asList(10,20,3,40,5);
    // nums.forEach(e->S.o.println(e));
    // nums.forEach(e->S.o.println(e*2));
    // Collections.sort(nums);
    /*
      Collections.sort(nums, new Comparator<Integer>(){
      @Override public int compare(Integer o1, Integer o2)
      {
        return o1>o2 ? -5 : 5;}
      });
```

```
// Collection.sort(nums, (o1, e2) -> e1 > e2 ? -5 : 5);

// nums.forEach(s.o.println S.o::println); //method reference

S.o.println(nums);
}
}
```

o/p

Thread - 0

[10, 20, 3, 40, 5]

---

```
package javafeatures;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Stream;

public class Mainclass4
{
  public static void main(String[] args)
  {
    // List<String> names = Arrays.asList("Alpha", "Beta",
                                          "Gamaa");
    // names.forEach((e) -> S.o.println(e.length()));
    Stream<Integer> sn = Stream.of(10, 20, 10, 10, 30,
                                   40, 20);
    Stream<String> sn = Stream.of("Kholi", "D
                                  "Gambhir", "Ganguly");
```

```
// long count = sn.count();
// S.o.println(count);

// boolean res = sn.anyMatch((e) -> e == 200);
// S.o.println(res);

// long count = sn.distinct().count();
// S.o.println(count);

// S.o.println(sn.filter((e) -> e.contains("G").findFirst().
                                                get());

// S.o.println(sn.filter((e) -> !e.contains("G")).toList());
```

---

```
package java8.features
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Set;
import java.util.stream.Collection;

public class Mainclass5
{
  public static void main(String[] args)
  {
    List<Integer> nums = Arrays.asList(10, 20, 3, 40, 5,
                                       10, 20, 10);
```

```
/*
for (int x : nums) {i
{ if (x%2==0) { evens.add(x); }}
   S.o.println (evens);
}*/
```

```
/*
List<Integer> evens = nums.stream().
                      .filter(e -> e%2==0)
                      .distinct()
                      .toList(); */
Set<Integer> evens = nums.stream()
                      .filter(e -> e%2==0)
                      .distinct()
                      .collect(Collectors.toSet());
S.o.println (evens);
}
}
```

[20, 40, 10]

---

```
package javaFeatures;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;
```

```
public class Mainclass6
{
  public static void main(String[] args)
  {
    List<String> names = Arrays.asList("Dhoni", "Kholi",
                                       "Gambir", "Ganguly");
    List<String> updated = names.stream()
                           .map(e -> e.substring(0, 2))
                           .collect(Collectors.toList());
    S.o.println(updated);
  }
}
```

[Dh, Kh, Ga, Ga]

---

```
package &features;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

public class Mainclass7
{
  public static void main(String[] args)
  {
    List<Integer> nums = new ArrayList();
    Random ra = new Random();
```

```
long start = System.currentTimeMillis();

for (int i= 1; i< 1000000; i++)
{
   nums.add (ra.nextInt());
}

long end = System.currentTimeMillis();
// S.O.println (nums);

start = System.currentTimeMillis();
List<Integer> pnums = nums.stream().filter(e->e>0).
                                    toList();
end = System.currentTimeMillis();
S.O.println(end - start);

start = System.currentTimeMillis();
pnums = nums.parallelStream().filter(e->e>0)
                             .toList();
end = System.currentTimeMillis();
S.O.println(end - start);
}
}
```

Op
982
366

# PRIORITY QUEUE :

* It is implementation class of queue present in java.util
* It stores object in natural ascending order.
* It stores only comparable type object.
* It allows duplicates.
* It won't allow null values.
* It won't preserve insertion order
* Retrival - for each / iterator (retrival is random)
→ peek() and poll() we can use to retrive element in sorted arrays.
  - peek() Returns the head element
  - poll() returns the head element & remove the head element.

# MAP :

* It is an interface present in java.util package
* It stores object in the form of key and value pair.
* key and value both are objects.

* key can't be duplicate but value can be duplicate.

put (object key, object value)

putAll (map m1);

remove (object key)

containsKey (object key)

get (object key)

keySet()

values()

entrySet()

size()

isEmpty()

clear()

* Implementation class HashMap, HashTable, TreeMap.

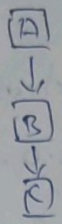
| HashMap | Hash table |
|---|---|
| In HashMap key and value can be null. | In hashtable key and value can't null. |
| HashMap is not legacy class. | Hashtable is a legacy class. |

| Hashmap is not thread safe | Hashtable is thread safe |
|---|---|
| Hashmap faster in performance | Hashtable slower in performance |

| COMPARABLE | COMPARATOR |
|---|---|
| comparable is an interface present is java.lang | Comparator is an interface present in java.util. |
| Comparable Abstract method is : Public int compareTO (object o1) | Comparator abstract method : public int compare (object o1, object o2) |
| Collection stores object in sorted order based on comparable implementation. | If we want to sort objects based on some other factors. we must use comparator. |

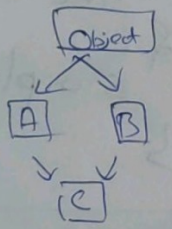
## Multi-level Inheritance:

```
class A {
}

class B extends A {
}

class C extends B {
}
```




## Multiple Inheritance:

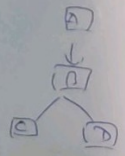
```
class A {
}

class B {
}

class C extends A, B {
}
```




## Hybrid Inheritance

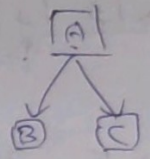
* combination of other.

```
class A {
}

Class B extends A {
}

class C extends B {
}

class D extends B {
}
```




## Heirarchical Inheritance

```
class A {
}

class B extends A {
}

class C extends A {
}
```




---

```
Interface I1 {
}

Interface I2 {
}

class A implements I1, I2 {
}
```