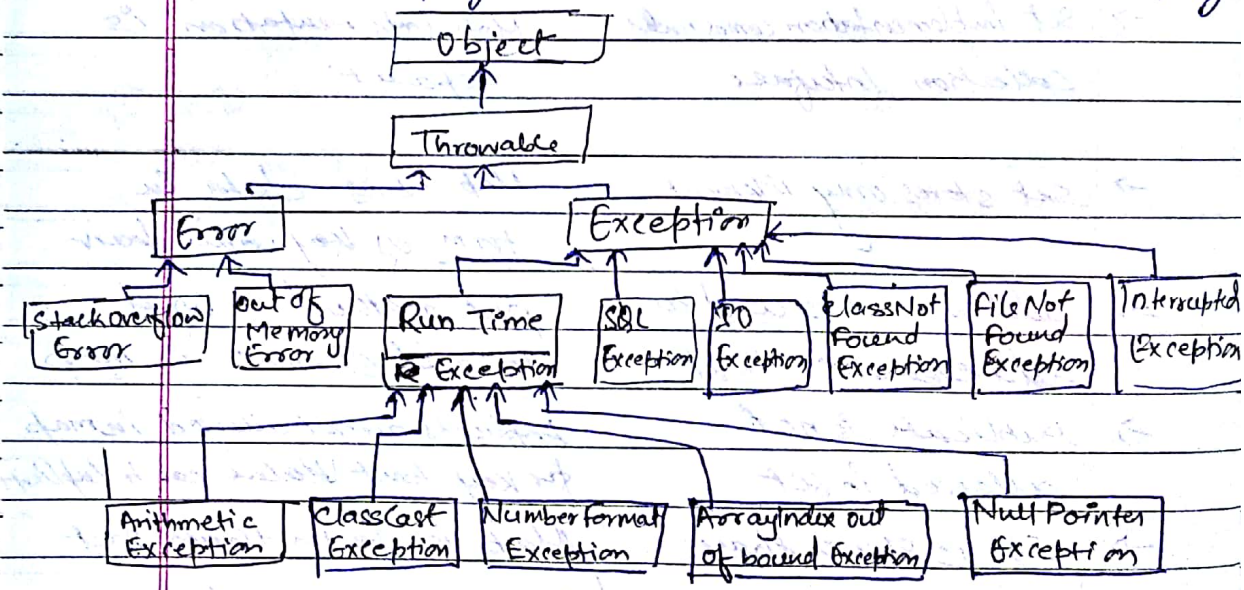
Exception Handling :-

→ Exceptions are nothing but runtime error
→ Some statements will be syntactically perfect but might behave abnormally at runtime, so the program will result in run-time error.
→ If a program result in run time error, the it will be terminated abruptly
→ so, addressing that run-time error through try catch block, so that program continues its execution, is called as exception handling.
→ we can handle RunTime Exception using try catch block
→ Risky statement which might behave abnormally at runtime should be developed inside try block and a catch block to address the exception
→ If you are handling the exception using try catch block then program will not be terminated abruptly



→ whenever any statement behaves abnormally at runtime, the runtime environment will create an appropriate object of Throwable class type, and it will throw that to method in which abnormal statement is encountred or found. This is called as an exception. And handling this exception using try catch block is called as exception handling.

→ If the exception is not handled through by catch block then the program will be terminated abruptly

** → If the exception occurs at run-time then only catch block will be executed

→ and if the exception is NOT triggered at runtime then catch block will NOT be executed

→ When we are handling dynamic data then we should use by Catch block

eg- int res = a/b ; where b value can be anything at runtime but if it is zero, then it will be a runtime error, so your program should terminate abruptly. It should not continue execution.

```
PSVM (String[] args)
{ sopln ("Main starts ....");
  int [] arr = { 10, 20}
  try { sopln(arr[20]); // arr[1]
      }
  Catch (ArithmeticException e)
  { sopln (" caught !!"); }
  finally { sopln ("finally block Running}
  Sopln ("Main Ends");
}
```

①	②
Main starts..	Main starts..
Caught	20
Running finally	Running finally
Main Ends	Main Ends

Finally

→ finally is a block which will be used in exception handling

→ finally block will be executed irrespective of exception

→ If you want to execute certain statements without fail at any scenario then we should develop those statements inside finally block

eg- sql connection
- file connection
- system input/output

→ All the connection established b/w database, file filesystem of computer can be closed using finally block so that resources will used efficiently

** In which scenario finally block will not be executed

→ In only one scenario, if there is a call to system.exit then finally will not be executed

→ Order of writing → try catch finally
or try finally (without catch)

→ One try block can have multiple catch block but appropriate catch block will execute

or One try block can be followed by multiple catch block but appropriate catch block will be executed

```
package
class A
{  void test1()
   { sopln ("Non-static method");}
}Public class Tester
{PSVM (String[] args)
{
 Sopln("Main starts");
 try{
     A  a1 = null;
        a1.test1();
 }
 Catch (ArithmeticException e)
 { Sopln ("1st catch");}
 Catch (NullPointerException e)
 { Sopln ("2nd Catch");}
 Sopln ("Main Ends");
}}
```

Main starts
Null Pointer Exceptio

If exception is handled properly it will execute all the statements properly, even all loops will be executed

```
Package exceptionhandlingtopic;
public class Tester
{ psvm (String[] args)
 { sopln ("Main starts---");
   int[] arr1 = {10,20,30,40,50,60,70,80};
   int[] arr2 = {10,0,30,0,50}
  for(int i=0 ; i<arr.length; i++)
  { try{
      sopln (arr1[i]/arr2[i])
      }
    catch (Arithmetic Exception e)
    {sopln("Divide by Zero");}
    Catch (ArrayIndex out of bound e)
    { sopln("No Such Index");}
  }
  Sopln("Main Ends");
}}
```

Main starts
1
Divide by zero
1
Divide by zero
1
No such Index
No such Index
No such Index
Main Ends

~~Main Ends~~

Auto upcasting Concept

→ Throwable catch block can handle any type of exception, because Throwable is the supermost class for all the exception related classes (Auto upcasting)

```
package
public class Tester
{
  psvm (String[] arg)
 { sopln ("Main starts");
   try
   {//int a = 90/0; // Arithmetic Exception
    // int.[] arr = new [-5] // ArrayIndex Out of bound
    /*A a1 = null
      a1.test() */  // null Pointer Exception
      int a = Integer.parseInt("hey") // Number format Exception
   }
   catch (Throwable e)
   { sopln ("caught"); }
  sopln ("Main Ends");
}
```

```
package
Public class Tester
{ PSVM (String[] args)
  { Soln ("Main starts ---");
    try { int a = 90/0; }
    Catch (Arithmetic Exception e)
    { Soln ("1st Caught"); }
    Catch (Runtime Exception e)
    { Soln ("2nd Caught");
    Catch (Exception e)
    { Soln ("3rd Caught"); }
    Catch (Throwable e)
    { Soln ("4th Catch"); }
    Soln ("Main Ends");
  }
}
```

```
Main starts
1st Caught
Main Ends
```

→ While Handling Exception through by catch block we should follow an order

→ we should develop the catch block from most specific exception class to most generic exception class and we should not duplicate exception

eg

Method of exception class!

```
PSVM (String[] args)
{ Soln ("The Main starts");
  try
  { int a = 10/0; }
  Catch (Arithmetic Exception ov)
  { // ov.printStackTrace();
    Soln (ov.getMessage());
  }
  Soln ("Main Ends");
}
```

```
The Main starts
/ by zero
Main Ends -
```

Checked and Unchecked exception

Date
Page

→
```
PSVM (String[] ag)
{ Sopln ("Main starts");
  try
  { Class. forName ("Salman"); }
  Catch (Class Not found Exception e)
  { e. printStackTrace ();
  }
  Sopln ("Main Ends");
}
```

e ——— StackTrace

Main starts

MainEnds

Class is a class in Java having static method forName()

```
PSVM (String[] args) throws ClassNotFound Exception
{ Sopln ("Main Starts");
  Class. forName ("Salman");
  Sopln ("Main Ends");
}
```

Main starts

RTE

program terminate Abruptly

Package

```
Public class Tester
{   static void test1() throws ClassNotFoundE —
    { Class.forName ("Salman"); }
    Static void test2() throws ClassNotFoundExc
    { test1(); }
    Static void test3() throws ClassNotfound Exception
    { test2(); }
    PSVM (String[] args)
    { Sopln ("Main starts");
      try
      { test1(); }
      Catch (class NotFound Exception e)
      { e. PrintStackTrace (); }
      Sopln("Main Ends");
    }
```

Main starts

Exception

Main Ends

→ There are two types of exception
① checked
② Unchecked

①→ Unchecked exceptions will not be detected at Compile time
Here compiler cannot identify that certain statement might generate an abnormal condition
These kind of exceptions are called as unchecked exception
eg. int res = a/b
res = a/0

→ All the exception classes related to RunTime Exception class which are subclass to Exception class will fall under this category.
→ Also the immediate subclass to Error Class also fall under unchecked exception category

② checked
checked exception will be detected at compile Time

→ Here compiler can identify that certain statements that might generate an abnormal condition at the time of execution
eg- Class.forName("packs.A");
Class.forName("Salman")
Thread.sleep( )
Scan.nextInt

There are two ways to handle checked exceptions.
① try Catch
② Throws

throws is a keyword which is used to re-throw the already thrown exception
or to delegate the exception to other method

→ throws should be used at the method declaration part

→ through throws keyword, the user of the method can understand what kind of error/exception method can generate for illogical values or invalid values.

Q Difference b/w error class and exception class

→ issue because of external resource vs issue because of java code

Difference b/w final, finally, and finalise()

→ final is a keyword which is used to avoid overridding
- final variable's value cannot be changed, final method's cannot be overridden and final class cannot be inherited
- i.e. final class method's cannot be overridden

→ finally is a block, which is used in exception handling

→ finally block will be executed irrespective of exception

If you want to execute certain statements without fail irrespective of exceptions then those statements should be executed in finally block. closing the database connection, input output connection, file connection

** - finalise() is a non-static method of Object class which will be inherited to every class of java which will be used for garbage collection by Garbage collector thread i.e thread

What is Garbage Collection

→ The process of removing the abandoned or unreachable object from the heap memory is called as garbage collection

abandoned object means those objects which will not have any references.

Garbage collector is a background thread or daemon thread (lowest priority thread) which will clear the abandoned object from heap memory by using finalise() method of the abandoned object

We can predict when exactly any object is eligible for garbage collection but we cannot predict when actually garbage collection happens

We can also place a request to garbage collector to perform garbage collection by using System.gc();

package

```
class B
{
@override
protected void finalise() throws Throwable
{
  Sopln ("Executing finalise method");
  super.finalise();
}
public class Tester19
{
  PSVM(String[] args)
  { Sopln ("Main start");
    B b1 = new B()
    b1 = new B()
    System.gc();
    Sopln ("Main End");
  }
}
```

Arrays	Collections
→ Fixed in Size	
→ Memory	
→ performance	
→ only homogeneous	
Inherit functionality x	
→ Programs	
→ Can store primitive/object	

collection :→ if we want to represent a group of individual object as a single entity then we should for collection

collection framework :- It contains several classes and interfaces which can be used to represent a group of individual object as a single entity

```
class X { int i, int i
          X(int i, int j) { this.i = i; this.j = j; }
over
toString
psvm(String[] args){ ArrayList list = new ArrayList()
list.add(new X(10,20));
                 (20,15));
                 (15,10));
                 (1,200));
                 (18,0));
Sopln(list)
Collection.sort(list,(o1,o2) -> (((X)o1 - ((X)o2));
```

Threads

Multi Tasking → performing multiple Task Simultaneously is called as Multitasking

Multitasking can be achieved in two ways

① Multiproccessing → executing multiple application or proces simultaneously

Here some amount of CPU time (3-4 miliseconds) will be shared with multiple applications

② Multi Threading :→ dividing the application in multiple parts and allocating separate CPU time and memory some amount of memory

→ Thread is nothing but an execution instance which will have its own memory and process time.

→ Thread is a concreate class which is available in java.lang package and is also subclass to Object class

→ Using Thread class and it feature we can achieve multithreading

→ Thread class is having lot of static and non-static method

→ ~~Non~~-static member

① sleep (long ms)

usage

```
class Util
{ static void sleep (long millis)
  { try {
      Thread.sleep(millis); {
    Cater (InterruptedException e)
    { e.printStackTrace();
  } } }
```

2. currentThread() → returns object of current thread
we can explore non-static proties of Thread

Non-static members

① Name → All threads will have a name and it can be changed

```
t1 = Thread.currentThread();
String s1 = t1.getName();
t1.setName();
```

② priority → All threads will have priority between 1 to 10
1 lowest 10 highest, 5 normal
child thread will have same priority as parent thread by default

```
int i = t1.getPriority()
t1.setPriority()
```

③ IsDaemon() → true/false → least priority thread
→ If any thread life depends on parent thread life that thread is daemon thread
→ Daemon thread is always looking for parent thread that if parent thread is still executing or not, if not then Daemon will exit/stop even if any allocated task is pending
If a thread if not a daemon thread then it is user thread or foreground thread.
we can convert any thread to daemon using setDaemon(true) before starting thread

④ ID → every thread is having a unique id and that unique id is long
We can identify any thread using id uniquely.
No setter method for id only getter

→ Whenever we run any java program, automatically 3 threads will be created

① Main Thread ② Thread Scheduler ③ Garbage Collection

① Main thread is user thread or foreground thread
Its main responsibility is to run main method

② Thread Scheduler is a daemon thread. Its main responsibility is to perform thread registration
Thread registration means allocating CPU time and memory

③ Garbage Collector is a daemon thread or background thread. Its main responsibility is to perform Garbage Collection using finalize method of that object.

Any unreachable object / abdandoned object is eligible for Garbage Collection
Garbage Collection is automatic in java, we can still place a request System.gc()

Thread Development

In thread class two important non static method
① start and ② run
task which has to be completed simultaneously should be developed inside run method while overriding

start() method performs 2 operation
① Thread Registration through Thread Registration
② It will call the run method of the thread object

Process 1

→ Create a subclass to thread class
Override run() method
Call the start() method through subclass reference

we cannot call start method twice on same thread
it will throw IllegalThreadException

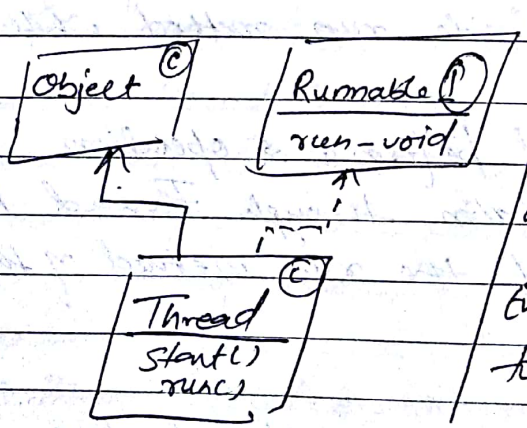
2nd Procedure to Thread Development.

① Create a subclass to Runnable interface (function interface @ runc) method
② override run method
③ create a subclass object
④ Creat a Thread class object by supplying the subclass reference to thread constructor
⑤ Call start() method through Thread reference

Here we can call any number of thread using on thread reference

* ~~late~~ If exception is occuring in any thread and it is not handled then which ever thread recieves unhandled exception only that thread terminates

* If any exception occurs in child thread then that thread is only terminating not the main thread. main thread completes its execution and viceversa



run() method is an abstract method of Runnable Interface.

Even thread is subclass to runable Interface.

```
PSVM(String[] args)
{   new Thread(new Runnable(){
                @Override
                public void run(){
                    for(int i=0; i<1000; i++){
    }).start();     SOPln("from runnable");
}
```