Exception hierarchy
==========
refer Exception.png

Exception:: Most of the cases exceptions are caused by our program and these are recoverable
ex:: If FileNotFoundException occurs then we can use local file and we can continue rest of
the program execution normally.

Error:: Most of the cases errors are not caused by our program these are due to lack of system
resources and these are non-recoverable.
ex:: If OutOfMemoryError occurs being a programmer we can't do anything the program will be terminated abnormally.

Checked vs UnCheckedExceptions
==========
=> The exceptions which are checked by the compiler whether programmer handling or not, for smooth execution of the program at the
runtime are called CheckedException.
eg::FileNotFoundException,IOException,SQLException...

=> The exceptions which are not checked by the compiler whether programmer is handling or not such type of exceptions are called
as "UnCheckedExceptions".
eg::NullPointerException,ArithmeticException

Note:: RunTimeException and its child classes,Error and its child classes are called as "UncheckedException",remaining all exceptions
are considered as "CheckedExceptions".
Note:: Whether the exception is checked or unchecked compulsorily it should occurs at runtime only and there is no chance of
Occuring any exception at compile time.

A checked exception is said to be fully checked exception if and only if all its child classes are also checked.
1. IOException
2. InterruptedException

A checked exception is said to be partially checked if and only if some of its child classes are unchecked.
eg:: Throwable, Exception

Describe the behaviour of following exceptions?
A. RunTimeException =====> Unchecked
B. Error =====> Unchecked
C. IOException =====> FullyChecked
D. Exception =====> PartiallyChecked
E. InterruptedException===> FullyChecked
F. Throwable ===> PartiallyChecked
G. ArithemeticException===> Unchecked
H. NullPointerException===> Unchecked
I. FileNotFoundException==> FullyChecked

Customized Exception handling
==========
1. It is highly recomended to handle exceptions
2. In our program the code which may rise exception is called "risky code"

3. We have to place our risky code inside try block and corresponding handling code inside catch block.

```
Example::
try{
     ...
     ... risky code
     ...
 }catch(XXXX e){
    ...
    ... handling code
    ...
}
```

```
Code without using try catch
============================
 class Test{
      public static void main(String... args){
            System.out.println("statement1");
            System.out.println(10/0);
            System.out.println("statement2");
      }
 }

output:
 statement1
 RE: AE:/by zero
    at Test.main()
Abnormal termination

with using try catch
====================
public class Test{
      public static void main(String... args){
            System.out.println("statement1");
            try{
                System.out.println(10/0);
            }catch(ArithemticException e){
                System.out.println(10/2);
            }
            System.out.println("statement2");
      }
 }
output:
  Statement1
    5
  Statement2

========================
Control flow in try catch
========================
try{
      Statement-1;
      Statement-2;
      Statement-3;
}catch( X e){
      Statement-4;
}
      Statement5;
```

Case 1:: If there is not exception.
Output: 1,2,3,5 normal termination.

Case 2:: if an exception raised at statement2 and corresponding catch block matched.
Output: 1,4,5 normal termination.

Case 3:: if any exception raised at statement2 but the corresponding catch block not matched.
Output: 1 Abnormal termination.

Case 4:: if an exception raised at statement 4 or statement 5.
Output: Abnormal termination.

Note::
1. Within the try block if anywhere an exception raised then rest of the try block wont
be executed even though we handled that exception.Hence we have to place/take only risk
code inside try block and length of the try block should be as less as possible.

2. If any statement which raises an exception and it is not part of any try block then it is
always abnormal termination of the program.

3. There may be a chance of raising an exception inside catch and finally blocks also in
addition to try block.

Various methods to print exception information
==========
Throwable class defines the following methods to print exception information to the console

printStackTrace()
This method prints exception information in the following format.
Name of the exception:description of exception stack trace

toString()=> This method prints exception information in the following format
Name of the exception : description of exception

getMessage() => This method returns only description of the exception Description.

eg::
```
public class Test{
	public static void main(String[] args){
		try{
			System.out.println(10/0);
		}catch(ArithmeticException e){
			e.printStackTrace(); => java.lang.ArithmeticException :/by Zero
                                                at Test.main(Test.java:6)
			System.out.println(e); => java.lang.ArithmeticException:/by Zero
			System.out.println(e.getMessage());=> /by Zero
		}
	}
```

```
}
```
Default exception handler internally uses printStatckTrace() method to print exception information to the console.

Try with mulitple catch Blocks
==========
The way of handling the exception is varied from exception to exception,hence for every exception type it is recomended
to take a separate catch block. That is try with multiple catch blocks is possible and recomended to use.

Example#1
=========
```
try{
      ...
      ...
      ...
}
catch(Exception e){
      default handler
}
```
This approach is not recomended because for any type of Exception we are using same catch block.
==========

Example#2
==========
```
try{
      ....
      ....
      ....
}catch(FileNotFoundException fe){

}catch(ArithmeticException ae){

}catch(SQLException se){

}catch(Exception e){

}
```
This approach is highly recomended because for any exception raise we are defining a seperate catch block.

++++++++++
If try with multiple catch blocks present then order of catch blocks is very important, it should be from child to parent by mistake
if we are taking from parent to child then we will get "CompileTimeError" saying
"exception XXXX has already been caught".

Example#1:
```
class Test{
      public static void main(String[] args){
            try{
                  System.out.println(10/0);
            }catch(Exception e){
                  e.printStackTrace();
            }catch(ArithmeticException ae){
```

```
                ae.printStackTrace();
            }
      }
}
CE: exception java.lang.ArithmeticException has already been caught
=====================================================================
Example#2:
class Test{
      public static void main(String[] args){
            try{
                  System.out.println(10/0);
            }catch(ArithmeticException ae){
                  ae.printStackTrace();
            }catch(Exception e){
                  e.printStackTrace();
            }
      }
}
Output:
Compile succesfully
=====================================================================
```

finally
  . It is not recomended to take clean up code inside try block becoz there is no gurantee for the execution of every statement
    inside a try block.
  . It is not recomended to place clean up code inside catch block becoz if there is no exception then catch block wont be executed.
  . we require some place to maintain clean up code which should be executed always irrespective of whether exception raised or not raised and
    whether or not handled.
    such type of best place is nothing but finally block.
  . Hence the main objective of finally block is to maintain cleanup code.

Example#1.
==========

```
try{
      risky code
}catch( X e){
      handling code
}finally{
      cleanup code
}
```

The speciality of finally block is it will be executed always irrespecitve of whether the exception is raised or not raised and
whether handled or not handled.

Case-1: If there is no Exception

```
class Test{
      public static void main(String... args){
            try{
                  System.out.println("try block gets executed");
            }catch(ArithmeticException e){
                  System.out.println("catch block gets executed");
            }finally{
                  System.out.println("finally block gets executed");
            }
```

```
	}
}
Output
try block gets executed
finally block gets executed

Case-2: If an Exception is raised,but the corresponding catch block matched
class Test{
	public static void main(String... args){
		try{
			System.out.println("try block gets executed");
			System.out.println(10/0);
		}catch(ArithmeticException e){
			System.out.println("catch block gets executed");
		}finally{
			System.out.println("finally block gets executed");
		}
	}
}
Output
try block gets executed
catch block gets executed
finally block gets executed

Case-3: If an Exception is raised,but the corresponding catch block not matched
class Test{
	public static void main(String... args){
		try{
			System.out.println("try block gets executed");
			System.out.println(10/0);
		}catch(NullPointerException e){
			System.out.println("catch block gets executed");
		}finally{
			System.out.println("finally block gets executed");
		}
	}
}
Output
Try block gets executed
finally block gets executed
Exception in thread "main" java.lang.ArithemeticException :/by Zero
 atTest.main(Test.java:8)

return vs finally
=================
Even though return statement present in try or catch blocks first finally will be
executed and
after that only return statement will be considered.ie finally block dominates
return statement.

Example:
class Test{

	public static void main(String... args){
		try{
			System.out.println("try block executed");
			return;
		}catch(ArithmeticException e){
			System.out.println("catch block executed");
```

```
            }finally{
                  System.out.println("finally block executed");
            }
      }
}
Output
 try block executed
 finally block executed

Example::
If return statement present try,catch and finally blocks then finally block return
statement will be considered.
class Test{
      public static void main(String... args){
            System.out.println(m1());
      }
      public static int m1(){
            try{
                  System.out.println(10/0);
                  return 777;
            }catch(ArithmeticException e){
                  return 888;
            }finally{
                  return 999;
            }
        }
}

finally vs System.exit(0)
=========================
   There is only one situation where the finally block wont be executed is whenever
we are using System.exit(0) method.
   When ever we are using System.exit(0) then JVM itself will be shutdown, in this
case finally block wont be executed.
      ie,.. System.exit(0) dominates finally block

public class Test {
      public static void main(String[] args) {
                  try{
                        System.out.println("Inside try");
                        System.exit(0);//shutting down jvm
                  }catch (Exception e){
                        System.out.println("catch block executed");
                  }finally{
                        System.out.println("finally block executed");
                  }
      }
}
Output::
Inside try

Note:: System.exit(0);
1. This argument acts as status code, Instead of Zero, we cant take any integer
value
2. Zero means normal termination, non zero means abnormal termination
3. This status code internally used by JVM,whether it is zero or non-zero there is
no change
   in the result and effect is same w.r.t program.
```

Difference b/w final,finally and finalize
==========
final
  => final is the modifier applicable for classes, methods and variables
  => If a class declared as the final then child class creation is not possible.
  => If a method declared as the final then overriding of that method is not possible.
  => If a variable declared as the final then reassignement is not possible.

finally
  => It is a final block associated with try-catch to maintain clean up code, which should
     be executed always irrespective of whethere exception raised or not raised and whether
     handled or not handled.

finalize
  => It is a method, always invoked by Garbage Collector just before destroying an object to
     perform cleanup activites.

Note::
  1. finally block meant for cleanup activites related to try block where as finalize() method
     for cleanup activites related to object.
  2. To maintain cleanup code finally block is recomended over finalize() method because we can't
     expect exact behaviour of GC.