Exception Handling

Q1) What is Exception in Java?

An Exception is an abnormal condition which occurs during the execution of a program and disrupts normal flow of a program. This exception must be handled properly. If it is not handled, program will be terminated abruptly.

Q2)What is Exception Handling?

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc. so that the normal execution flow of the program can be maintained.

Q3) What is the difference between error and exception in Java?

Errors are mainly caused by the environment in which an application is running. For example, OutOfMemoryError happens when JVM runs out of memory. Where as exceptions are mainly caused by the application itself. For example, NullPointerException occurs when an application tries to access null object.

Q4) How many types of exception can occur in a Java program?

There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:

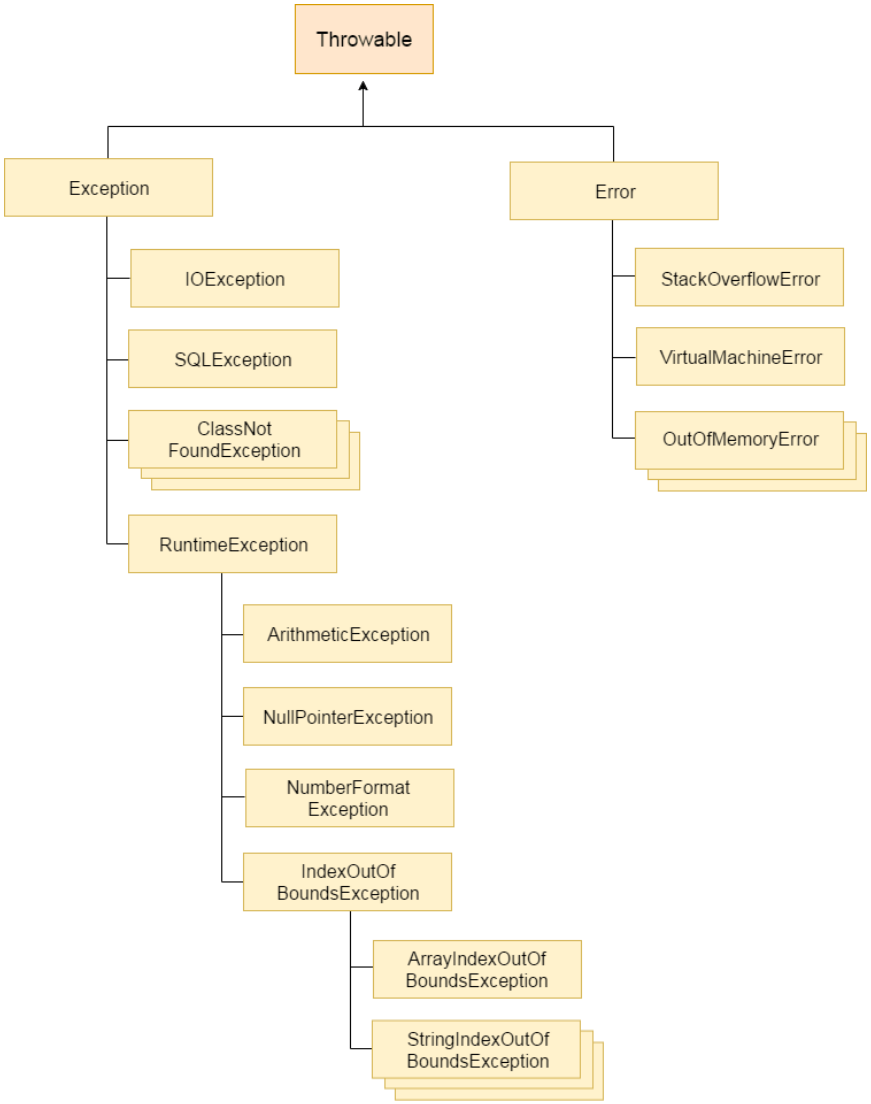
- **Checked Exception**: Checked exceptions are the one which are checked at compile-time. For example, SQLException, ClassNotFoundException, etc.

- **Unchecked Exception**: Unchecked exceptions are the one which are handled at runtime because they can not be checked at compile-time. For example, ArithmaticException, NullPointerException, ArrayIndexOutOfBoundsException, etc.

- **Error**: Error cause the program to exit since they are not recoverable. For Example, OutOfMemoryError, AssertionError, etc.

Q5) Explain the hierarchy of Java Exception classes?

The java.lang.Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses: Exception and Error. A hierarchy of Java Exception classes are given below:



Q6) What is the difference between Checked Exception and Unchecked Exception?

Ans:-

1) Checked Exception

The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions, e.g., IOException, SQLException, etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes that extend RuntimeException are known as unchecked exceptions, e.g., ArithmeticException, NullPointerException, etc. Unchecked exceptions are not checked at compile-time.

Q7) What is the base class for Error and Exception?

The Throwable class is the base class for Error and Exception.

Q8) Is it necessary that each try block must be followed by a catch block?

It is not necessary that each try block must be followed by a catch block. It should be followed by either a catch block OR a finally block. So whatever exceptions are likely to be thrown should be declared in the throws clause of the method. Consider the following example.

```
1.  public class Main{
2.      public static void main(String []args){
3.        try{
4.           int a = 1;
5.           System.out.println(a/0);
6.        }
7.        finally
8.        {
9.           System.out.println("rest of the code...");
10.       }
11.    }
12.}
13.
```

Output:

```
Exception in thread main java.lang.ArithmeticException:/ by zero
rest of the code...
```

Q9) What is the output of the following Java program?

```
1. public class ExceptionHandlingExample {
2. public static void main(String args[])
3. {
4.     try
5.     {
6.         int a = 1/0;
7.         System.out.println("a = "+a);
8.     }
9.     catch(Exception e){System.out.println(e);}
10.    catch(ArithmeticException ex){System.out.println(ex);}
11.}
12.}
```

Output

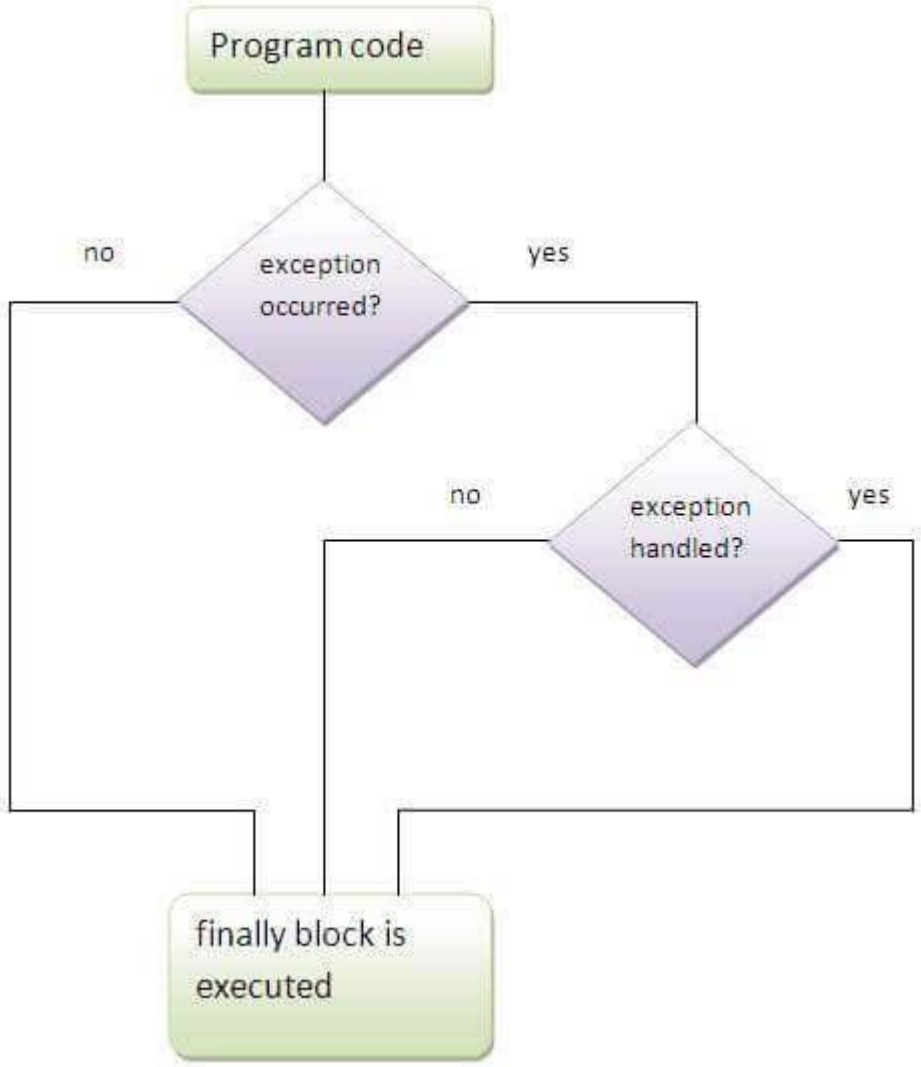
```
ExceptionHandlingExample.java:10: error: exception ArithmeticException has
already been caught
        catch(ArithmeticException ex){System.out.println(ex);}
        ^
1 error
```

Explanation

ArithmaticException is the subclass of Exception. Therefore, it can not be used after Exception. Since Exception is the base class for all the exceptions, therefore, it must be used at last to handle the exception. No class can be used after this.

Q10) What is finally block?

The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not. In other words, we can say that finally block is the block which is always executed. Finally block follows try or catch block. If you don't handle the exception, before terminating the program, JVM runs finally block, (if any). The finally block is mainly used to place the cleanup code such as closing a file or closing a connection. Here, we must know that for each try block there can be zero or more catch blocks, but only one finally block. The finally block will not be executed if program exits(either by calling System.exit() or by causing a fatal error that causes the process to abort).



More details.

Q11) Can finally block be used without a catch?
Yes, According to the definition of finally block, it must be followed by a try or catch block, therefore, we can use try block instead of catch. More details.

Q12) Is there any case when finally will not be executed?
Finally block will not be executed if program exits(either by calling System.exit() or by causing a fatal error that causes the process to abort)

Q13) What is the difference between throw and throws?

throw keyword	throws keyword
1) The **throw** keyword is used to throw an exception explicitly.	The **throws** keyword is used to declare an exception.
2) The checked exceptions cannot be propagated with throw only.	The checked exception can be propagated with throws
3) The **throw** keyword is followed by an instance.	The **throws** keyword is followed by class.
4) The **throw** keyword is used within the method.	The **throws** keyword is used with the method signature.
5) You cannot throw multiple exceptions.	You can declare multiple exceptions, e.g., public void method()throws IOException, SQLException.

More details.

Q14) What is the output of the following Java program?

```
1.  public class Main{
2.    public static void main(String []args){
3.      try
4.      {
5.         throw 90;
6.      }
7.      catch(int e){
```

```
8.              System.out.println("Caught the exception "+e);
9.          }
10.
11.     }
12.}
```

Output

```
Main.java:6:  error:  incompatible  types:  int  cannot  be  converted  to
Throwable
            throw 90;
            ^
Main.java:8: error: unexpected type
        catch(int e){
              ^
  required: class
  found:    int
2 errors
```

Explanation

In Java, the throwable objects can only be thrown. If we try to throw an integer object, The compiler will show an error since we can not throw basic data type from a block of code.

Q15) What is the output of the following Java program?

```
1.  class Calculation extends Exception
2.  {
3.      public Calculation()
4.      {
5.          System.out.println("Calculation class is instantiated");
6.      }
7.      public void add(int a, int b)
8.      {
9.          System.out.println("The sum is "+(a+b));
10.     }
11.}
12.public class Main{
13.     public static void main(String []args){
14.         try
```

```
15.         {
16.             throw new Calculation();
17.         }
18.         catch(Calculation c){
19.             c.add(10,20);
20.         }
21.     }
22.}
```

Output

Calculation class is instantiated

The sum is 30

Explanation

The object of Calculation is thrown from the try block which is caught in the catch block. The add() of Calculation class is called with the integer values 10 and 20 by using the object of this class. Therefore there sum 30 is printed. The object of the Main class can only be thrown in the case when the type of the object is throwable. To do so, we need to extend the throwable class.

Q16) Can an exception be rethrown?

Yes.

Q17) Can subclass overriding method declare an exception if parent class method doesn't throw an exception?

Yes but only unchecked exception not checked.

More details.

Q18) What is exception propagation?

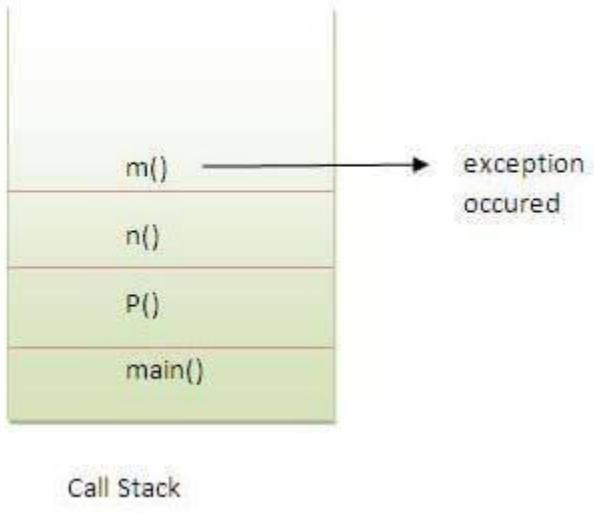
An exception is first thrown from the top of the stack and if it is not caught, it drops down the call stack to the previous method, If not caught there, the exception again drops down to the previous method, and so on until they are caught or until they reach the very bottom of the call stack. This procedure is called exception propagation. By default, checked exceptions are not propagated.

```
1.  class TestExceptionPropagation1{
2.    void m(){
3.      int data=50/0;
4.    }
5.    void n(){
6.      m();
7.    }
8.    void p(){
9.     try{
10.     n();
11.    }catch(Exception e){System.out.println("exception handled");}
12.   }
13.   public static void main(String args[]){
14.    TestExceptionPropagation1 obj=new TestExceptionPropagation1();
15.    obj.p();
16.    System.out.println("normal flow...");
17.   }
18. }
```

Test it Now

Output:

```
exception handled
normal flow...
```



More details.

Q19) What is the output of the following Java program?

```
1.  public class Main
2.  {
3.      void a()
4.      {
5.          try{
6.          System.out.println("a(): Main called");
7.          b();
8.          }catch(Exception e)
9.          {
10.             System.out.println("Exception is caught");
11.         }
12.     }
13.     void b() throws Exception
14.     {
15.      try{
16.         System.out.println("b(): Main called");
17.         c();
18.      }catch(Exception e){
19.         throw new Exception();
20.     }
21.     finally
```

```
22.    {
23.        System.out.println("finally block is called");
24.    }
25.   }
26.   void c() throws Exception
27.   {
28.      throw new Exception();
29.   }
30.
31.   public static void main (String args[])
32.   {
33.      Main m = new Main();
34.      m.a();
35.   }
36.}
```

Output

```
a(): Main called
b(): Main called
finally block is called
Exception is caught
```

Explanation

In the main method, a() of Main is called which prints a message and call b(). The method b() prints some message and then call c(). The method c() throws an exception which is handled by the catch block of method b. However, It propagates this exception by using **throw Exception()** to be handled by the method a(). As we know, finally block is always executed therefore the finally block in the method b() is executed first and prints a message. At last, the exception is handled by the catch block of the method a().

Q20) What is the output of the following Java program?

```
1.  public class Calculation
2.  {
```

```
3.  int a;
4.  public Calculation(int a)
5.  {
6.      this.a = a;
7.  }
8.  public int add()
9.  {
10.     a = a+10;
11.     try
12.     {
13.         a = a+10;
14.         try
15.         {
16.             a = a*10;
17.             throw new Exception();
18.         }catch(Exception e){
19.             a = a - 10;
20.         }
21.     }catch(Exception e)
22.     {
23.         a = a - 10;
24.     }
25.     return a;
26. }
27.
28. public static void main (String args[])
29. {
30.     Calculation c = new Calculation(10);
31.     int result = c.add();
32.     System.out.println("result = "+result);
33. }
34.}
```

Output

```
result = 290
```

Explanation

The instance variable a of class Calculation is initialized to 10 using the class constructor which is called while instantiating the class. The add method is called which returns an integer value result. In add() method, a is incremented by 10 to be 20. Then, in the first try block, 10 is again incremented by 10 to be 30. In the second try block, a is multiplied by 10 to be 300. The second try block throws the exception which is caught by the catch block associated with this try block. The catch block again alters the value of a by decrementing it by 10 to make it 290. Thus the add() method returns 290 which is assigned to result. However, the catch block associated with the outermost try block will never be executed since there is no exception which can be handled by this catch block.