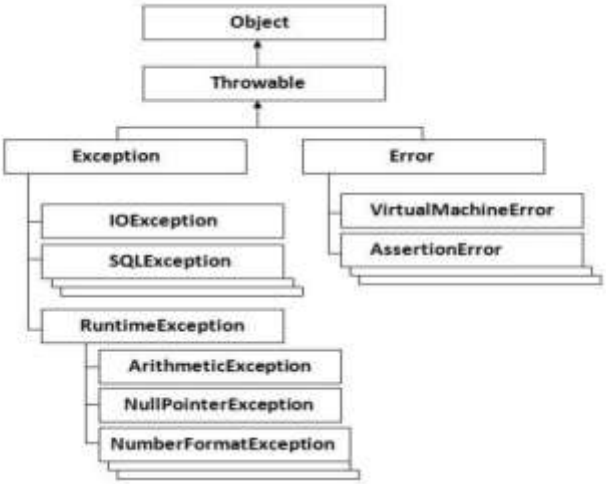
Exception Handling in Java

1. **Description**
    a. powerful mechanism to handle the **runtime errors**

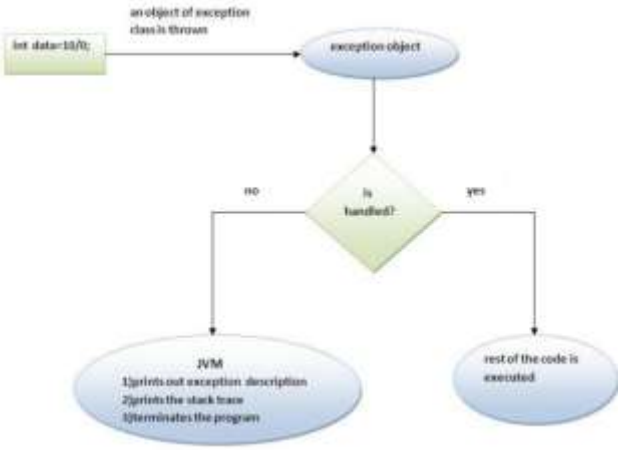


2. **Types of Exception**
    a. Checked Exception
        i. The classes that extend **Throwable** class except RuntimeException and Error. E.g. *IOException*, *SQLException*
    b. Unchecked Exception
        i. classes that extend **RuntimeException.** e.g *ArithmeticException*, *NullPointerException*, *ArrayIndexOutOfBoundsException*
    c. Error
        i. Error is irrecoverable. e.g. *OutOfMemoryError*, *VirtualMachineError*, *AssertionError*

3. **Java Exception Handling Keywords**
    a. try
    b. catch
    c. finally
    d. throw
    e. throws

    f. try-catch
        i. try block
            1. used to enclose the code that might throw an exception. must be followed by either catch or finally block.
        ii. catch block
            1. used to handle the Exception.

    g. Internal working of java try-catch block

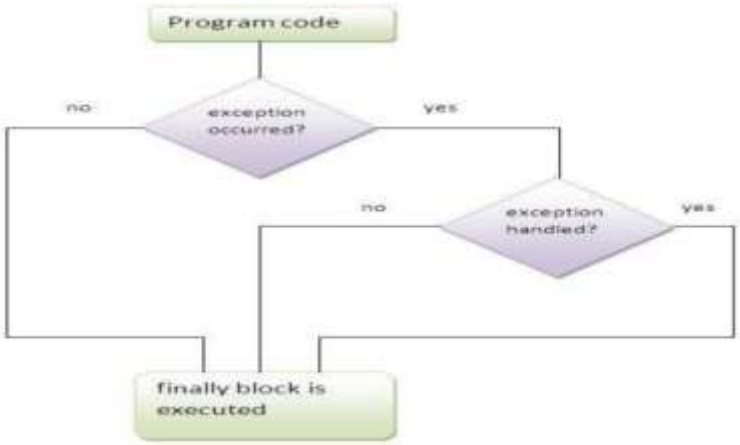


4. **Rules**
    a. Rule 1
        i. At a time only one Exception is occured and at a time only one catch block is executed.
    b. Rule 2
        i. All catch blocks must be ordered from most specific to most general i.e. catch for ArithmeticException **must** come before catch for Exception. (Else Compile-time error)

5. **Nested try block**

```
1.  ....
2.  try
3.  {
4.      statement 1;
5.      statement 2;
6.      try
7.      {
8.          statement 1;
9.          statement 2;
10.     }
11.     catch(Exception e)
12.     {
13.     }
14. }
15. catch(Exception e)
16. {
17. }
18. ....
```

6. **finally block**
    a. always executed whether exception is handled or not



    b. Cases

Case 1

Let's see the java finally example where **exception doesn't occur**.

```
1.  class TestFinallyBlock{
2.    public static void main(String args[]){
3.    try{
4.     int data=25/5;
5.     System.out.println(data);
6.    }
7.    catch(NullPointerException e){System.out.println(e);}
8.    finally{System.out.println("finally block is always executed");}
9.    System.out.println("rest of the code...");
10.   }
11. }
```

```
Output:5
       finally block is always executed
       rest of the code...
```

Case 2

Let's see the java finally example where **exception occurs and not handled**.

```
1.  class TestFinallyBlock1{
2.    public static void main(String args[]){
3.    try{
4.     int data=25/0;
5.     System.out.println(data);
6.    }
7.    catch(NullPointerException e){System.out.println(e);}
8.    finally{System.out.println("finally block is always executed");}
9.    System.out.println("rest of the code...");
10.   }
11. }
```

```
Output:finally block is always executed
       Exception in thread main java.lang.ArithmeticException:/ by zero
```

Let's see the java finally example where **exception occurs and handled**.

```
1.  public class TestFinallyBlock2{
2.    public static void main(String args[]){
3.    try{
4.     int data=25/0;
5.     System.out.println(data);
6.    }
7.    catch(ArithmeticException e){System.out.println(e);}
8.    finally{System.out.println("finally block is always executed");}
9.    System.out.println("rest of the code...");
10.   }
11. }
```

```
Output:Exception in thread main java.lang.ArithmeticException:/ by zero
       finally block is always executed
       rest of the code...
```

c. Rule 3

i. For each try block there can be zero or more catch blocks, but only one finally block.

d. Note: 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).

7. throw exception

a. used to explicitly throw an exception

b. can throw either **checked or uncheked exception** in java by throw keyword

c. mainly used to throw custom exception

d. syntax of java throw keyword

i. throw exception;

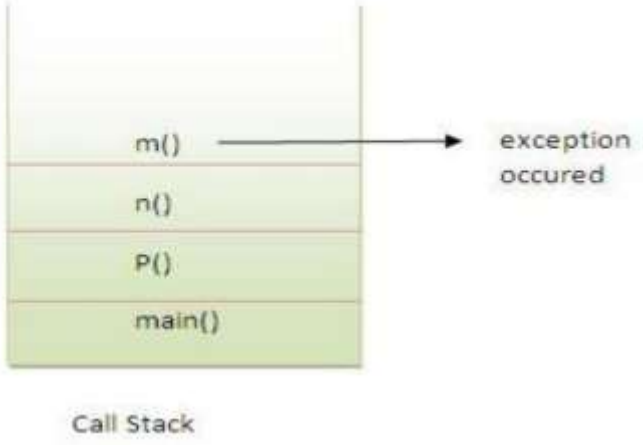
ii. throw new IOException("sorry device error);

e. Rule 4

i. By default **Unchecked Exceptions** are forwarded in calling chain (propagated)

8. Exception propagation

```
Output:exception handled
       normal flow...
```



i. In the above example exception occurs in m() method where it is not handled, so it is propagated
ii. to previous n() method where it is not handled, again it is propagated to p() method where exception is handled.

iii. Exception can be handled in any method in call stack either in **main() method**, **p()** method,**n()** method or **m()** method.

b. Rule 5
   i. By default, **Checked Exceptions** are not forwarded in calling chain (propagated).
   Output:Compile Time Error

9. throws keyword

a. used to declare an exception(Checked Exceptions only)
b. Exception Handling is mainly used to handle the checked exceptions
c. any unchecked exception such as NullPointerException, it is programmers fault

d. Syntax of java throws
   i. return_type method_name() throws exception_class_name{
   ii. //method code
   iii. }

e. Note: Now Checked Exception can be propagated (forwarded in call stack).

Java throws example

Let's see the example of java throws clause which describes that checked exceptions can be propagated by throws keyword.

```
1.  import java.io.IOException;
2.  class Testthrows1 {
3.    void m()throws IOException{
4.      throw new IOException("device error");//checked exception
5.    }
6.    void n()throws IOException{
7.      m();
8.    }
9.    void p(){
10.   try{
11.     n();
12.   }catch(Exception e){System.out.println("exception handled");}
13.   }
14.   public static void main(String args[]){
15.   Testthrows1 obj=new Testthrows1();
16.   obj.p();
17.   System.out.println("normal flow...");
18.   }
19. }
```

Output:

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

f. Rule6

i. If you are calling a method that **declares an exception**(throws), you must either caught or declare the exception.
ii. There are two cases:
    1. Case1: You caught the exception i.e. handle the exception using **try/catch.**
    2. Case2: You declare the exception i.e. specifying **throws with the method.**
        a. Subcase1 : Exception occur
            i. Output:Runtime Exception
        b. Subcase 2: Does not occur
            i. Code is fine

10. Difference between throw and throws in Java

No.	throw	throws
1)	Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
2)	Checked exception cannot be propagated using throw only.	Checked exception can be propagated with throws.
3)	Throw is followed by an instance.	Throws is followed by class.
4)	Throw is used within the method.	Throws 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.

Java throw example

```
1. void m(){
2. throw new ArithmeticException("sorry");
3. }
```

Java throws example

```
1. void m()throws ArithmeticException{
2. //method code
3. }
```

Java throw and throws example

```
1. void m()throws ArithmeticException{
2. throw new ArithmeticException("sorry");
3. }
```

11. Difference between final, finally and finalize

No.	final	finally	finalize
1)	Final is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden and final variable value can't be changed.	Finally is used to place important code, it will be executed whether exception is handled or not.	Finalize is used to perform clean up processing just before object is garbage collected.
2)	Final is a keyword.	Finally is a block.	Finalize is a method.

12. ExceptionHandling with MethodOverriding

a. Rules

i. If the superclass method does not declare an exception subclass overridden method cannot declare the **checked exception** but **it can declare unchecked exception**(Arithmetic).
Output:Compile Time Error

ii. If the superclass method declares an exception subclass overridden method can declare same, subclass exception or no exception

13. Java Custom Exception

```
1. class InvalidAgeException extends Exception{
2.  InvalidAgeException(String s){
3.   super(s);
4.  }
5. }
```

```
1. class TestCustomException1{
2.
3.   static void validate(int age)throws InvalidAgeException{
4.     if(age<18)
5.       throw new InvalidAgeException("not valid");
6.     else
7.       System.out.println("welcome to vote");
8.    }
9.
10.  public static void main(String args[]){
11.    try{
12.    validate(13);
13.    }catch(Exception m){System.out.println("Exception occured: "+m);}
14.
15.    System.out.println("rest of the code...");
16.  }
17. }
```

```
Output:Exception occured: InvalidAgeException:not valid
       rest of the code...
```