# Try catch control flow

1. 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***.

1. There may be a chance of raising an exception inside catch and finally blocks

also in addition to try block.

# Nested try catch control flow

If exception raised in inner try, then JVM search for corresponding inner catch block. If not found, then that exception get propagate to outer try and search for corresponding outer catch block.

# Try catch finally control flow

**Case 1:** exception not raised : **finally block with normal termination**

**Case 2 :** exception raised in try block and caught in catch block : **finally block with normal termination**

**Case 3 :** exception raised in try block but not caught in catch block : **finally block with abnormal termination**

**Case 4 :** exception raised in **catch** block : **finally block with abnormal termination**

**Case 5 :** exception raised in **finally** block : **abnormal termination**

**\*\***Some examples from Hari Krishna sir is given at last page

# Finally with return statement

1. **Return statement in try or catch block**

Though return statement is here in try/catch block, first **finally block will execute** then return statement.

**public class** Rough {  
 **public static void** main(String[] args) {  
 System.***out***.println(*test*());  
 }  
  
 **private static** String test() {  
 **try** {  
 **return "try block return"**;  
 } **catch** (Exception e) {  
 **return "catch block return"**;  
 } **finally** {  
 System.***out***.println(**"finally block"**);  
 }  
 }  
}

output:

**finally block**

**try block return**

**public class** Rough {  
 **public static void** main(String[] args) {  
 System.***out***.println(*test*());  
 }  
 **private static** String test() {  
 **try** {  
 **int** a=10/0;  
 **return "try block return"**;  
 } **catch** (Exception e) {  
 **return "catch block return"**;  
 } **finally** {  
 System.***out***.println(**"finally block"**);  
 }  
 }  
}

**output:**

**finally block**

**catch block return**

1. **Return statement in try, catch, finally block**

Here **finally block return statement** is considered.

**case 3 : try with exception and final block has return statement**

**public class** Rough {  
 **public static void** main(String[] args) {  
 System.***out***.println(*test*());  
 }  
 **private static** String test() {  
 **try** {  
 **return "try block return"**;  
 } **catch** (Exception e) {  
 **return "catch block return"**;  
 } **finally** {  
 **return "finaly block return"**;  
 }  
 }  
}

**output:**

**finally block return**

**if finally block has return statement and exception raised in try block, then that exception is not propagate to calling method, because return values of finally block overrides exception object in JDH.**

**public class** Case6 {  
 **public static void** main(String[] args) {  
  
 System.***out***.println(*test*());  
 }  
  
 **private static int** test() {  
 **try** {  
 System.***out***.println(**"try block"**);  
 System.***out***.println(10 / 0);  
 **return** 20;  
 } **catch** (NullPointerException pointerException) {  
 System.***out***.println(**"catch block"**);  
 **return** 25;  
 } **finally** {  
 System.***out***.println(**"finally block"**);  
 **return** 30;  
 }  
  
 }  
}

**output:**

**try block**

**finally block**

**30**

* **You cannot place any statement after the finally block having return statement. Otherwise we get CE: unreachable code**

# Various possible combinations of try catch finally

# Throw statement

* We use throw statement for **customized exception** but not for predefined exception
* After throw statement, we cannot write any statement other wise we will get CE: un reachable code
* We can use throw keyword only for Throwable type

# Throws statement

* main objective of "**throws**" keyword is to ***delegate the responsibility of exception handling to the caller method***
* Use throws keyword only on **constructor** or **method** level
* "throws" keyword required only ***checked exceptions***. Usage of throws for unchecked exception there is no use.
* "throws" keyword required only to convince complier. Usage of throws keyword doesn't prevent abnormal termination of the program.

**If try block is not raising any exception then we cannot write catch block for that exception otherwise it will give us compile time error saying *exception XXX is never thrown in body of corresponding try statement.*. But this rule is applicable for fully checked exception**

# Exception handling keywords summary:

1. **try**: To maintain risky code.
2. **Catch**: To maintain handling code.
3. **Finally**: To maintain cleanup code.
4. **throw**: To handover our created exception object to the JVM manually.
5. **throws**: To delegate responsibility of exception handling to the caller method.

# 1.7 enhancement

## 1. try with resources

Before 1.7, programmer needs to close all the resources compulsorily which are open in finally block which increase code length and readability.

Try with resources will automatically close all the resource at end of try block, programmer no need to close it.

Syntax:

**Try(resource1,resource2){**

**}**

**Catch(){**

**}**

**Ex:**

**more resource**

**try(**

**FileOutputStream fileOutputStream =new FileOutputStream("/java7-new-features/src/abc.txt");**

**InputStream input = new FileInputStream("/java7-new-features/src/abc.txt")){**

        //logic

**}catch(Exception exception){**

**System.out.println(exception);**

**}**

In parenthesis of try , we can have one or more resources. For more resources, they should be separated by semicolon.

* Resources must be **AutoCloseable** (class should implement AutoCloseable interface)
* Resources in try with resource are **final** by default so we cannot reassign those.
* We can declare any numbers of resources but those should be separated by semicolon(;)

## 2. multi catch block

While handling the exception we need to write same handling code for many exceptions. So this will increase code length and reviews readability. To overcome this, we use multi catch block where we can write same code for many exception.

**try{**

**}**

**catch(exception1 e1 | exception1 e1){**

**}**

In multi catch block, there should not be any relation between Exception types(either child to parent Or parent to child Or same type , otherwise we will get Compile time error )

*Map*<Integer, String> map1 = users.stream().collect(Collectors.*toMap*(User::getId, User::getName));  
System.out.println(map1);  
*Map*<String, String> map2 = users.stream().collect(Collectors.*toMap*(User::getName,User::getCity ,(oldValue,newValue)-> oldValue, LinkedHashMap::new));  
System.out.println(map2);

## Exception handling in method overriding

* If SuperClass does not declare an exception, then the SubClass can only declare unchecked exceptions, but not the checked exceptions.
* If SuperClass declares an exception, then the SubClass can only declare the same or child exceptions of the exception declared by the SuperClass and any new Runtime Exceptions, just not any new checked exceptions at the same level or higher.
* If SuperClass declares an exception, then the SubClass can declare without exception.

**\*\*** **HARI KRISHNA SIR’as examples**

// try/catch/finally without exception

public class Case1 {

public static void main(String[] args) {

try {

System.out.println("try block");

} catch (Exception e) {

System.out.println("catch block");

} finally {

System.out.println("finally block");

}

System.out.println("after try/catch/finally block");

}

}

===============

// try/catch/finally with exception caught in catch block

public class Case2 {

public static void main(String[] args) {

try {

System.out.println("try block");

int a =10/0;

} catch (Exception e) {

System.out.println("catch block");

} finally {

System.out.println("finally block");

}

System.out.println("after try/catch/finally block");

}

}

=============

// try/catch/finally with exception but not caught in catch block

public class Case3 {

public static void main(String[] args) {

try {

System.out.println("try block");

int a =10/0;

} catch (NullPointerException e) {

System.out.println("catch block");

} finally {

System.out.println("finally block");

}

System.out.println("after try/catch/finally block");

}

}

=======================

//try/finally without exception

public class Case4 {

public static void main(String[] args) {

try {

System.out.println("try block");

} finally {

System.out.println("finally block");

}

System.out.println("after try/catch/finally block");

}

}

===============

//try/finally without exception

public class Case5 {

public static void main(String[] args) {

try {

System.out.println("try block");

System.out.println(10/0);

} finally {

System.out.println("finally block");

}

System.out.println("after try/catch/finally block");

}

===========