Exception Handling

* an exception is an event that disrupts the normal flow of the program
* The core advantage of exception handling is **to maintain the normal flow of the application**.
* 

Difference between Exception and error

**Error**” is a critical condition that cannot be handled by the code of the program. “**Exception**” is the exceptional situation that can be handled by the code of the program

* Checked Exception
* Unchecked Exception
* Error

1) Checked Exception

* The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.
* checked exceptions denote error scenarios which are outside the immediate control of the program. They occur usually interacting with outside resources/network resources e.g. database problems, network connection errors, missing files etc.
* **Example of checked exceptions** are : **ClassNotFoundException**, **IOException**, **SQLException** FileNotFoundException  and so on.

2) Unchecked Exception

* The classes which ss RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.
* 3) Error
* Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

|  |  |
| --- | --- |
| **Keyword** | **Description** |
| try | The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone. |
| catch | The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later. |
| finally | The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not. |
| throw | The "throw" keyword is used to throw an exception. |
| throws | The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature. |

public class Test

{

    public static void main(String[] args)

    {

        try

        {

            System.out.printf("1");

            int sum = 9 / 0;

            System.out.printf("2");

        }

        catch(ArithmeticException e)

        {

            System.out.printf("3");

        }

        catch(Exception e)

        {

            System.out.printf("4");

        }

        finally

        {

            System.out.printf("5");

        }

    }

}

Answer: d) 135

**Ans.**(d)  
**Explanation:**Once an exception occurs in try block, the execution passes to **corresponding** catch statement and doesn’t return back to try block. Only one of the catch blocks are executed at a time. finally block is always executed whether or not the exception occurred.

public class Test

{ private void m1()

{ m2(); System.out.printf("1"); }

private void m2()

{ m3(); System.out.printf("2"); }

private void m3()

{ System.out.printf("3");

try

{ int sum = 4/0;

System.out.printf("4"); }

catch(ArithmeticException e)

{ System.out.printf("5");

} System.out.printf("7");

}

public static void main(String[] args)

{ Test obj = new Test(); obj.m1();

}

}

Answer: 35721

public class Test

{ public static void main(String[] args)

{ try{ System.out.printf("1");

int data = 5 / 0; }

catch(ArithmeticException e)

{ System.out.printf("2"); System.exit(0);

}

finally

{System.out.printf("3");

}

System.out.printf("4");

}

}

Answer: 12

public class Test

{

public static void main(String[] args)

{

try

{

System.out.printf("1");

int data = 5 / 0;

}

catch(ArithmeticException e)

{

Throwable obj = new Throwable("Sample");

try

{

throw obj;

}

catch (Throwable e1)

{

System.out.printf("8");

}

}

finally

{

System.out.printf("3");

}

System.out.printf("4");

}

}

Answer: 1834 **Explanation:**Exceptions can be thrown in catch clause. This is done in order to change the exception type at run time. Exceptions in catch clause are thrown by creating instances of class Throwable as shown in the program.

import java.io.EOFException;

import java.io.IOException;

public class Test

{

public static void main(String[] args)

{

try

{

System.out.printf("1");

int value = 10 / 0;

throw new IOException();

}

catch(EOFException e)

{

System.out.printf("2");

}

catch(ArithmeticException e)

{

System.out.printf("3");

}

catch(NullPointerException e)

{

System.out.printf("4");

}

catch(IOException e)

{

System.out.printf("5");

}

catch(Exception e)

{

System.out.printf("6");

}

}

}

Answer:- 13 **Explanation:**In multi-catch statements, the exceptions must be listed from more specific to more general. Only one catch statement which is most specific to the occurred exception is executed.

class Base extends Exception {}

class Derived extends Base {}

public class Main {

public static void main(String args[]) {

// some other stuff

try {

// Some monitored code

throw new Derived();

}

catch(Base b) {

System.out.println("Caught base class exception");

}

catch(Derived d) {

System.out.println("Caught derived class exception");

}

}

}

Answer::-Compiler Error because base class exception is caught before derived class

class Main {

public static void main(String args[]) {

try {

throw 10;

}

catch(int e) {

System.out.println("Got the Exception " + e);

}

}

}

Answer::- Got the Exception 0 **Explanation:** In Java only throwable objects (Throwable objects are instances of any subclass of the Throwable class) can be thrown as exception. So basic data type can no be thrown at all.

class Test extends Exception { }

class Main {

public static void main(String args[]) {

try {

throw new Test();

}

catch(Test t) {

System.out.println("Got the Test Exception");

}

finally {

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

}

}

}

Answer::- Got the Test Exception

Inside finally block

class Test

{

public static void main(String[] args)

{

try

{

int a[]= {1, 2, 3, 4};

for (int i = 1; i <= 4; i++)

{

System.out.println ("a[" + i + "]=" + a[i] + "\n");

}

}

catch (Exception e)

{

System.out.println ("error = " + e);

}

catch (ArrayIndexOutOfBoundsException e)

{

System.out.println ("ArrayIndexOutOfBoundsException");

}

}

}

Compiler error **Explanation:** ArrayIndexOutOfBoundsException has been already caught by base class Exception. When a subclass exception is mentioned after base class exception, then error occurs.

class Test

{

String str = "a";

void A()

{

try

{

str +="b";

B();

}

catch (Exception e)

{

str += "c";

}

}

void B() throws Exception

{

try

{

str += "d";

C();

}

catch(Exception e)

{

throw new Exception();

}

finally

{

str += "e";

}

str += "f";

}

void C() throws Exception

{

throw new Exception();

}

void display()

{

System.out.println(str);

}

public static void main(String[] args)

{

Test object = new Test();

object.A();

object.display();

}

}

Abdec **Explanation:** ‘throw’ keyword is used to explicitly throw an exception.  
finally block is always executed even when an exception occurs.  
Call to method C() throws an exception. Thus, control goes in catch block of method B() which again throws an exception. So, control goes in catch block of method A().

class Test

{ int count = 0;

void A() throws Exception

{

try

{

count++;

try

{

count++;

try

{

count++;

throw new Exception();

}

catch(Exception ex)

{

count++;

throw new Exception();

}

}

catch(Exception ex)

{

count++;

}

}

catch(Exception ex)

{

count++;

}

}

void display()

{

System.out.println(count);

}

public static void main(String[] args) throws Exception

{

Test obj = new Test();

obj.A();

obj.display();

}

}

Ans::- 5 **Explanation:** ‘throw’ keyword is used to explicitly throw an exception.  
In third try block, exception is thrown. So, control goes in catch block.  
Again, in catch block exception is thrown. So, control goes in inner catch block.

class NestedTry {

// main method

public static void main(String args[])

{

// Main try block

try {

// initializing array

int a[] = { 1, 2, 3, 4, 5 };

// trying to print element at index 5

System.out.println(a[5]);

// try-block2 inside another try block

try {

// performing division by zero

int x = a[2] / 0;

}

catch (ArithmeticException e2) {

System.out.println("division by zero is not possible");

}

}

catch (ArrayIndexOutOfBoundsException e1) {

System.out.println("ArrayIndexOutOfBoundsException");

System.out.println("Element at such index does not exists");

}

}

// end of main method

}

ArrayIndexOutOfBoundsException

Element at such index does not exists

**Exception Handling with Method Overriding**

When Exception handling is involved with Method overriding, ambiguity occurs. The compiler gets confused as which definition is to be followed. Such problems were of two types:

* **Problem 1: If The SuperClass doesn’t declare an exception:**

In this problem, two cases arise:

* + **Case 1: If SuperClass doesn’t declare any exception and subclass declare checked exception**

**Example:**

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| import java.io.\*;    class SuperClass {        // SuperClass doesn't declare any exception      void method()      {          System.out.println("SuperClass");      }  }    // SuperClass inherited by the SubClass  class SubClass extends SuperClass {        // method() declaring Checked Exception IOException      void method() throws IOException      {            // IOException is of type Checked Exception          // so the compiler will give Error            System.out.println("SubClass");      }        // Driver code      public static void main(String args[])      {          SuperClass s = new SubClass();          s.method();      }  } |

**Compile Errors:**

prog.java:16: error:

method() in SubClass cannot override method() in SuperClass

void method() throws IOException

^

overridden method does not throw IOException

1 error

* + **Case 2: If SuperClass doesn’t declare any exception and SubClass declare Unchecked exception**

**Example:**

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| import java.io.\*;    class SuperClass {        // SuperClass doesn't declare any exception      void method()      {          System.out.println("SuperClass");      }  }    // SuperClass inherited by the SubClass  class SubClass extends SuperClass {        // method() declaring Unchecked Exception ArithmeticException      void method() throws ArithmeticException      {            // ArithmeticException is of type Unchecked Exception          // so the compiler won't give any error            System.out.println("SubClass");      }        // Driver code      public static void main(String args[])      {          SuperClass s = new SubClass();          s.method();      }  } |

**Output:**

SubClass

* **Problem 2: If The SuperClass declares an exception:**

In this problem also, three cases arise:

* + **Case 1: If SuperClass declares an exception and SubClass declares exceptions other than the child exception of the SuperClass declared Exception**

**Example:**

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| import java.io.\*;    class SuperClass {        // SuperClass declares an exception      void method() throws RuntimeException      {          System.out.println("SuperClass");      }  }    // SuperClass inherited by the SubClass  class SubClass extends SuperClass {        // SubClass declaring an exception      // which are not a child exception of RuntimeException      void method() throws Exception      {            // Exception is not a child exception          // of the RuntimeException          // So the compiler will give an error            System.out.println("SubClass");      }        // Driver code      public static void main(String args[])      {          SuperClass s = new SubClass();          s.method();      }  } |

**Compile Errors:**

prog.java:16: error:

method() in SubClass cannot override method() in SuperClass

void method() throws Exception

^

overridden method does not throw Exception

1 error

* + **Case 2: If SuperClass declares an exception and SubClass declares an child exception of the SuperClass declared Exception.**

**Example:**

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| import java.io.\*;    class SuperClass {        // SuperClass declares an exception      void method() throws RuntimeException      {          System.out.println("SuperClass");      }  }    // SuperClass inherited by the SubClass  class SubClass extends SuperClass {        // SubClass declaring a child exception      // of RuntimeException      void method() throws ArithmeticException      {            // ArithmeticException is a child exception          // of the RuntimeException          // So the compiler won't give an error          System.out.println("SubClass");      }        // Driver code      public static void main(String args[])      {          SuperClass s = new SubClass();          s.method();      }  } |

**Output:**

SubClass

* + **Case 3: If SuperClass declares an exception and SubClass declares without exception.**

**Example:**

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| import java.io.\*;    class SuperClass {        // SuperClass declares an exception      void method() throws IOException      {          System.out.println("SuperClass");      }  }    // SuperClass inherited by the SubClass  class SubClass extends SuperClass {        // SubClass declaring without exception      void method()      {          System.out.println("SubClass");      }        // Driver code      public static void main(String args[])      {          SuperClass s = new SubClass();      try {          s.method();      } catch (IOException e) {          e.printStackTrace();      }      }  } |

**Output:**

SubClass

**Conclusion for Handling such Exceptions:** Hence, following conclusions can be derived from the above examples:

1. If SuperClass does not declare an exception, then the SubClass can only declare unchecked exceptions, but not the checked exceptions.
2. If SuperClass declares an exception, then the SubClass can only declare the child exceptions of the exception declared by the SuperClass, but not any other exception.
3. If SuperClass declares an exception, then the SubClass can declare without exception.