Exception:

An exception is an unwanted or unexpected event that disturb the normal flow of program that is at run time it is called Exception.

Exception Handling:

Handle the unwanted and unexpected event to find alternative way that not to disturb the normal flow of program this is called Exception handling.

Example:

public class Exceptionex {  
 public static void main(String[] args) {  
 System.*out*.println(1);  
 System.*out*.println(2);  
 System.*out*.println(100/0);  
 System.*out*.println(3);  
 System.*out*.println(4);  
 }  
}

Output:

1

2

Exception in thread "main" java.lang.ArithmeticException: / by zero

at Exceptionex.main(Exceptionex.java:5)

1. Can checked exceptions be thrown from the static block?

No, checked exceptions cannot be thrown from a static block in Java. They

Object is parent class of Throwable.

Throwable is parent class of Exception class.

There are two type.

1.Exception

2.Error

|  |  |
| --- | --- |
| Exception | Error |
| Exception are occurred by Our program. | Error occur because of lack of System resourses.  (i).Memory is low  (ii).Ram is low  (iii). Slow proccessor |
| Exception are the problem which you can handle. | Error are the problem which cannot be handle. |
| There are two types: | Only one type:  unchecked |
| (i).Compile time Exception(checked Exception) |  |
| **Examples**: Checked Exceptions: SQLException, IOException Unchecked Exceptions: ArrayIndexOutOfBoundException, NullPointerException, ArithmeticException. | **Examples**: java.lang.StackOverflowError, java.lang.OutOfMemoryError |

Hierarchy of Exception:



1.Checked Exception(Compile time Exception):

Exception is the parent class of all exception in java.

Compiler check the exception is called checked exception.

public class Check {  
 public static void main(String[] args) {  
 FileInputStream s=new FileInputStream("d:/abc.txt");  
 }  
}

}

Output: java: unreported exception java.io.FileNotFoundException; must be caught or declared to be thrown.

2.Unchecked Exception(Run time Exception):

Compiler does not check the exception is called Unchecked Exception.

Example:

public class Check {  
 public static void main(String[] args) {  
 int a=100, b =0, c;  
 c = a/b;  
 System.*out*.println(c);  
 }  
}

Output:

Exception in thread "main" java.lang.ArithmeticException: / by zero

at Check.main(Check.java:6)

Difference between Checked and Unchecked :

|  |  |
| --- | --- |
| Checked Exception | Unchecked Exception |
| Checked Exception are exceptions that are checked and handled on compile time | Unchecked Exception are exceptions that are not checked at compile time |
| The program gives a compilation error if a method throws a checked exception | The program compiles fine because  the compiler is not able to check the exception. |
| Checked exception can be handled on throws keyword | It has not handled |
|  |  |

There are exception within line,then will create one exception object.

i.exception name

ii.Description

iii.Stack trace.

Example:

public class Check {  
 public static void main(String[] args) {  
 try {  
 FileInputStream s=new FileInputStream("d:/abc.txt");  
 }  
 catch(ArithmeticException e)  
 {  
 e.printStackTrace();  
 }  
 }  
}

Output:

D:\Exception Handling\Check.java:6:30

java: unreported exception java.io.FileNotFoundException; must be caught or declared to be thrown

We can handle the exception using 5 keyword:

* Try
* Catch

Syntax:

try

{

//risky code

}

Catch(ExceptionClassname ref.var.name) //Exception

{

//handling code

}

Example:

public class Check {  
 public static void main(String[] args) {  
 try {  
 int a = 100, b = 0, c;  
 c = a / b;  
 System.*out*.println(c);  
 }  
 catch(Exception/\*(Exception name)\*/ e)  
 {  
 System.*out*.println(e);  
 System.*out*.println("Number cannot divide by zero");  
 }  
 }  
}

Output:

java.lang.ArithmeticException: / by zero

Number cannot divide by zero

Example:

public class Check {  
 public static void main(String[] args) {  
 try {  
 FileInputStream s=new FileInputStream("d:/abc.txt");  
 }  
 catch(Exception/\*(Exception name)\*/ e)  
 {  
 System.*out*.println(e);  
 System.*out*.println("Number cannot divide by zero");  
 }  
 }  
}

Output:

java.lang.ArithmeticException: / by zero

Number cannot divide by zero

If we find the Exception in program then go to the catch block Otherwise no exception then not go to the catch block only try block executed.

3.Finally:

The finally block in Java is used to execute important code such as closing resources

Finally is the block that is always executed whether is exception is handled or not.

(i). If exception occurs in try block , then he go to first Catch block, after that he go to finally block and execute.

(ii). If exception does not occurs ,then he goes directly finally block and execute.

Syntax:

Try

{

//risky code

}

Catch(Exception e)

{

//handling code

}

Finally

{

//cleanup code

}

2.

Try

{

//risky Code

}

Finally

{

//Cleanup Code

}

1.Exception not occurs:

Example:

public class Check {  
 public static void main(String[] args) {  
 try {  
 int a=100, b=2, c;  
 c=a/b;  
 System.*out*.println(c);  
 }  
 catch(ArithmeticException e)  
 {  
 System.*out*.println(e);  
 }  
 finally  
 {  
 System.*out*.println("i m finally block");  
 }  
 }  
}

Output:

50

i m finally block

2.Exception Occur:

public class Check {  
 public static void main(String[] args) {  
 try {  
 int a=100, b=0, c;  
 c=a/b;  
 System.*out*.println(c);  
 }  
 catch(ArithmeticException e)  
 {  
 System.*out*.println(e);  
 }  
 finally  
 {  
 System.*out*.println("i m finally block");  
 }  
 }  
}

Output:

Exception Handling" Check

java.lang.ArithmeticException: / by zero

i m finally block

3.Catch block not use.

public class Check {  
 public static void main(String[] args) {  
 try {  
 int a=100, b=0, c;  
 c=a/b;  
 System.*out*.println(c);  
 }  
 finally  
 {  
 System.*out*.println("i m finally block");  
 }  
 System.*out*.println("hello");  
 }  
}

Output:

i m finally block

Exception in thread "main" java.lang.ArithmeticException: / by zero

at Check.main(Check.java:5)

1.we can use multiple catch block in one try block but we can only use single finally block with one try block not multiple.

2.If inside try block we use jump statement(break,continue,go to) and by using return,then always execute finally block, then control transfer.

3.Rule of finally block not execute.

(i). Using the System.exit method.

(ii). If occur error in try block,then not execute finally block.

(iii).Error occur in finally block,if we not handle,then finally block not execute.

(iv).The death of the thread.

Difference between final,finally and finalize .

|  |  |  |
| --- | --- | --- |
| Final | Finally | Finalize |
| Final is keyword | Finally is Block | Finalize is Method |
| Use with:  Variable(value has fixed cannot change)  Method(does not override)  Class(final class does not inherit) | Use with:  Try or try-catch block  i.Finally block provide the cleanup code.  ii.Finally block will execute to close their resources.  1.Try  {  //risky code  }  Catch(Exception e)  {  //handling code  }  Finally  {  //cleanup code  }  2.  Try  {  //risky Code  }  Finally  {  //Cleanup Code  } | Method is override for and object.  i.Finalize method will execute just prior to garbage collector.  Protected void finalize() throw throwable  {  Cleanup code  } |

Finalize Method:

The Java finalize() method of Object class is a method that the Garbage Collector always calls just before the deletion/destroying the object which is eligible for Garbage Collection to perform clean-up activity. Clean-up activity means closing the resources associated with that object like Database Connection, Network Connection, or we can say resource de-allocation. Remember, it is not a reserved keyword. Once the finalize() method completes immediately, Garbage Collector destroys that object.

Why finalize() method is used?

finalize() method releases system resources before the garbage collector runs for a specific object.

Why Not Use finalize():

Unpredictable Timing: You can't control when the Garbage Collector runs.

Performance Issues: Garbage Collection with finalize() can be slower, as objects with finalize() methods require additional processing.

Better Alternatives:

Use try-with-resources for managing resources like files and streams.

Explicitly close resources in a finally block.

Deprecated:

Starting with Java 9, the finalize() method is deprecated, and it is discouraged from being used because of its unpredictability and better alternatives being available.

Uses of try catch:

1.

public class Check {  
 public static void main(String[] args) {  
 try {  
 int a=100, b=2, c;  
 c=a/b;  
 System.*out*.println(c);  
 }  
 catch (ArithmeticException e)  
 {  
  
 }  
 catch(Exception e)  
 {  
  
 }  
 finally {  
 System.*out*.println("Finally block");  
 }  
 }  
}

Output:

50

Finally block

2.

public class Check {  
 public static void main(String[] args) {  
 try {  
 int a=100, b=2, c;  
 c=a/b;  
 System.*out*.println(c);  
 }  
 finally {  
 System.*out*.println("Finally block");  
 }  
 try{  
  
 }catch (Exception e)  
 {  
  
 }  
 }  
}

Output:

50

Finally block

3. Inside try nested try catch possible.

public class Check {  
 public static void main(String[] args) {  
 try {  
  
 try {  
  
 } catch (ArithmeticException e) {  
 }  
  
 }  
 catch (Exception e)  
 {  
  
 }  
 }  
}

4.Inside catch block nested try catch possible.

public class Check {  
 public static void main(String[] args) {  
 try {  
  
 }  
 catch (ArithmeticException e) {  
  
 try {  
  
 } catch (Exception b) {  
  
 }  
 }  
 }  
}

5.Inside finally try catch are possible.

public class Check {  
 public static void main(String[] args) {  
 try {  
  
 }  
 catch (ArithmeticException e) {  
  
 }  
 finally{  
 try {  
  
 } catch (Exception b) {  
  
 }  
 }  
 }  
}

6.

public class Check {  
 public static void main(String[] args) {  
 try {  
  
 }  
 System.*out*.println("1");  
 catch (ArithmeticException e) {  
  
 }  
 finally{  
 try {  
  
 } catch (Exception b) {  
  
 }  
 }  
 }  
}

Output:

java: 'try' without 'catch', 'finally' or resource declarations

Throw Exception:

1.

public class Check {  
 public static void main(String[] args) {  
 int a =100 , b=0, c=0;  
 c=a/b;  
 System.*out*.println(c);  
  
  
 }  
}

Output:

Exception in thread "main" java.lang.ArithmeticException: / by zero

at Check.main(Check.java:4)

if we handle this exception.

public class Check {  
 public static void main(String[] args) {  
 try  
 {  
 int a =100 , b=0, c=0;  
 c=a/b;  
 System.*out*.println(c);  
  
 }  
 catch (Exception e)  
 {  
 System.*out*.println(e);  
 }  
 System.*out*.println("kaushik");  
  
 }  
}

Output:

java.lang.ArithmeticException: / by zero

kaushik

2.

public class Check {  
 public static void main(String[] args) {  
 Check d = new Check();  
 d.divie();  
  
  
 }  
  
  
 void divie() {  
 int a = 100, b = 0, c = 0;  
 c = a / b;  
 System.*out*.println(c);  
 }  
}

Output:

Exception in thread "main" java.lang.ArithmeticException: / by zero

at Check.divie(Check.java:12)

at Check.main(Check.java:4)

If we handle this exception.

public class Check {  
 public static void main(String[] args) {  
 Check d = new Check();  
 d.divie();  
 }  
 void divie() {  
 try{  
 int a = 100, b = 0, c = 0;  
 c = a / b;  
 System.*out*.println(c);  
 }  
 catch(Exception e)  
 {  
  
 }  
 System.*out*.println("Handle this");  
 }  
}

Output:

Handle this

Another way:

public class Check {  
 public static void main(String[] args) {  
 Check d = new Check();  
 try {  
 d.divie();  
 } catch (Exception e) {  
  
 }  
 System.*out*.println("Handle this");  
 }  
   
 void divie() {  
  
 int a = 100, b = 0, c = 0;  
 c = a / b;  
 System.*out*.println(c);  
  
 }  
}

Output:

Handle this

4.Throw:

Throw keyword is used only to create exception object manually through programmer ,it cannot cannot handle the exception.

Throw exception handle to the jvm.

Throw keyword is used for checked and unchecked exception but throw is best for custom exception.

We can only throw class that comes in throwable child class.

It is user defined Exception.

After use throw keyword any statment provide then throw error.

Syntax:

Throw new ExceptionClassName(“ ”);

Example:

import java.util.Scanner;  
  
class YoungerAgeException extends RuntimeException  
{  
 YoungerAgeException(String msg)  
 {  
 super(msg);  
 }  
} //Customized exception class  
  
public class Customized {  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
 System.*out*.println("Enter your age");  
 int age= sc.nextInt();  
 if(age<18)  
 {  
 throw new YoungerAgeException("You are not eligble for votting");  
  
 }  
 else {  
 System.*out*.println("You can vote successfully:");  
 }  
 }  
}

5.Throws:

Throws keyword is uesd to declare an exception.it gives an information to the caller method that there may occur an exception to handle this exception so that normal flow can be maintained.

Throws keyword only indicate the exception,not handle the exception.if we not handle this exception then they provide unreported exception

Thows keyword only used for checked exception.

Example:

class ReadAndWrite  
{  
 void raedFile()  
 {  
 FileInputStream f = new FileInputStream("d:/abc.txt");  
 }  
}

Output:

java: unreported exception java.io.FileNotFoundException; must be caught or declared to be thrown

class ReadAndWrite  
{  
 void raedFile() throws FileNotFoundException  
 {  
 FileInputStream f = new FileInputStream("d:/abc.txt");  
   
 }  
}

import java.io.FileInputStream;  
import java.io.FileNotFoundException;  
  
class ReadAndWrite  
{  
 void raedFile() throws FileNotFoundException  
 {  
 FileInputStream f = new FileInputStream("d:/abc.txt");  
  
 }  
 void saveFile()  
 {  
 String str="txt";  
 FileInputStream f1 = new FileInputStream(":d/xyz.txt");  
 }  
}

Output:

java: unreported exception java.io.FileNotFoundException; must be caught or declared to be thrown.

import java.io.FileInputStream;  
import java.io.FileNotFoundException;  
  
class ReadAndWrite  
{  
 void raedFile() throws FileNotFoundException  
 {  
 FileInputStream f = new FileInputStream("d:/abc.txt");  
  
 }  
 void saveFile() throws FileNotFoundException  
 {  
 String str="txt";  
 FileInputStream f1 = new FileInputStream(":d/xyz.txt");  
 }  
}

Successfully compiled.

Handle this exceptiion:

import java.io.FileInputStream;  
import java.io.FileNotFoundException;  
  
class ReadAndWrite  
{  
 void raedFile() throws FileNotFoundException  
 {  
 FileInputStream f = new FileInputStream("d:/abc.txt");  
  
 }  
 void saveFile() throws FileNotFoundException  
 {  
 String str="txt";  
 FileInputStream f1 = new FileInputStream(":d/xyz.txt");  
 }  
}  
  
public class Throwws {  
 public static void main(String[] args) {  
 ReadAndWrite r = new ReadAndWrite();  
 try  
 {  
 r.raedFile();  
 }  
 catch (FileNotFoundException e)  
 {  
 e.printStackTrace();  
 }  
 System.*out*.println( "hello");  
 }  
}

Output:

java.io.FileNotFoundException: d:\abc.txt (The system cannot find the file specified)

at java.base/java.io.FileInputStream.open0(Native Method)

at java.base/java.io.FileInputStream.open(FileInputStream.java:213)

at java.base/java.io.FileInputStream.<init>(FileInputStream.java:152)

at java.base/java.io.FileInputStream.<init>(FileInputStream.java:106)

at ReadAndWrite.raedFile(Throwws.java:8)

at Throwws.main(Throwws.java:23)

hello

import java.io.FileInputStream;  
import java.io.FileNotFoundException;  
  
class ReadAndWrite  
{  
 void raedFile() throws FileNotFoundException  
 {  
 FileInputStream f = new FileInputStream("d:/abc.txt");  
  
 }  
 void saveFile() throws FileNotFoundException  
 {  
 String str="txt";  
 FileInputStream f1 = new FileInputStream(":d/xyz.txt");  
 }  
}  
  
public class Throwws {  
 public static void main(String[] args) {  
 ReadAndWrite r = new ReadAndWrite();  
 try  
 {  
 r.raedFile();  
 }  
 catch (FileNotFoundException e)  
 {  
 e.printStackTrace();  
 }  
 try{  
 r.saveFile();  
 }  
 catch (FileNotFoundException e)  
 {  
 e.printStackTrace();  
 }  
 System.*out*.println( "hello");  
 }  
}

Output:

java.io.FileNotFoundException: d:\abc.txt (The system cannot find the file specified)

at java.base/java.io.FileInputStream.open0(Native Method)

at java.base/java.io.FileInputStream.open(FileInputStream.java:213)

at java.base/java.io.FileInputStream.<init>(FileInputStream.java:152)

at java.base/java.io.FileInputStream.<init>(FileInputStream.java:106)

at ReadAndWrite.raedFile(Throwws.java:8)

at Throwws.main(Throwws.java:23)

java.io.FileNotFoundException: :d\xyz.txt (The filename, directory name, or volume label syntax is incorrect)

at java.base/java.io.FileInputStream.open0(Native Method)

at java.base/java.io.FileInputStream.open(FileInputStream.java:213)

at java.base/java.io.FileInputStream.<init>(FileInputStream.java:152)

at java.base/java.io.FileInputStream.<init>(FileInputStream.java:106)

at ReadAndWrite.saveFile(Throwws.java:14)

at Throwws.main(Throwws.java:30)

hello

Difference between throw and throws keyword:

|  |  |
| --- | --- |
| Throw | Throws |
| Throw keyword is used to create an exception object manually. | Throws keyword is used for declare the exception. |
| Throw keyword is used for checked  And unchecked exception buit is better for custom exception. | Throws keywod is used for checked exception |
| It is used for single exception | Multiple |
| It is used for inside method | It is used together method name |
| It is used for new keyword | It is used for class. |
| After throw we cannot provide any satement | It is not allow |

Throw keyword is the best option of create customized exception.

Throws keyword is used for checked exception.

Custom Checked Exception:

class UnderAgeException extends Exception{  
 UnderAgeException()  
 {  
 super("You are under age:");  
 }  
 UnderAgeException(String s)  
 {  
 super(s);  
 }  
}  
public class Custom {  
 public static void main(String[] args) {  
 int age = 17;  
 try {  
 if (age < 18) {  
 throw new UnderAgeException( );  
 }  
 else {  
 System.*out*.println("you can vote");  
 }  
 }  
 catch (UnderAgeException e)  
 {  
 e.printStackTrace();  
 }  
 }  
}

Output:

UnderAgeException: You are under age:

at Custom.main(Custom.java:16)

class UnderAgeException extends Exception{  
 UnderAgeException()  
 {  
 super("You are under age:");  
 }  
 UnderAgeException(String s)  
 {  
 super(s);  
 }  
}  
public class Custom {  
 public static void main(String[] args) {  
 int age = 17;  
 try {  
 if (age < 18) {  
 throw new UnderAgeException("You cannot vote" );  
 }  
 else {  
 System.*out*.println("you can vote");  
 }  
 }  
 catch (UnderAgeException e)  
 {  
 e.printStackTrace();  
 }}

Output:

UnderAgeException: You cannot vote

at Custom.main(Custom.java:16)

Custom unchecked Exception:

class UnderAgeException extends RuntimeException{  
 UnderAgeException()  
 {  
 super("You are under age:");  
 }  
 UnderAgeException(String s)  
 {  
 super(s);  
 }  
}  
public class Custom {  
 public static void main(String[] args) {  
 int age = 17;  
 try {  
 if (age < 18) {  
 throw new UnderAgeException("You cannot vote" );  
 }  
 else {  
 System.*out*.println("you can vote");  
 }  
 }  
 catch (UnderAgeException e)  
 {  
 e.printStackTrace();  
 }  
 System.*out*.println("hello");  
 }  
}

Output:

UnderAgeException: You cannot vote

at Custom.main(Custom.java:16)

hello