### Exception Handling

**Introduction:-**

* Dictionary meaning of the exception is abnormal termination.
* **An exception is an event that occurs during execution of the program that disturbs normal flow of the program instructions.**
* An unexpected even that disturbs the normal termination of the application is called exception.
* In application whenever the exception occurred,
  + 1. Program terminated abnormally
    2. Rest of the application is not executed.

To overcome above limitation in order to execute the rest of the application & to get normal termination of the application must handle the exception.

There are two ways to handle the exceptions in java.

* 1. By using try-catch block.
  2. By using throws keyword.

#### Exception Handling:-

* The main objective of exception handling is,
  1. To get normal termination of the application
  2. To execute the rest of the application code.
* Exception handling means just we are providing alternate code to continue the execution of remaining code &to get normal termination of the application.
* Every Exception is a predefined class present in different packages. java.lang.ArithmeticException **java.lang** java.io.IOException  **java.io**

java.sql.SQLException **java.sql**

**Types of Exceptions:-**

As per the sun micro systems standards The Exceptions are divided into three types

* + 1. Checked Exception
    2. Unchecked Exception
    3. Error

#### Unchecked Exception:-

* The exceptions which are not checked by the compiler are called unchecked Exception.

ArithmeticException,ArrayIndexOutOfBoundsException,NumberFormatException….etc  The classes that extends RuntimeException class are called unchecked exceptions.

class Test

{ public static void main(String[] args)

{ System.out.println(10/0); **java.lang.ArithmeticException: / by zero**

int[] a={10,20,30};

System.out.println(a[5]); **java.lang.ArrayIndexOutOfBoundsException: 5**

System.out.println("balu".charAt(12)); **java.lang.StringIndexOutOfBoundsException**

}

}

* If the application contains un-checked Exception code is compiled but at runtime JVM display exception message & program terminated abnormally.
* To overcome runtime problem must handle the exception either using try-catch blocks or by using throws keyword.

#### Checked Exception:-

* The Exceptions which are checked by the compiler are called Checked Exceptions. IOException,SQLException,InterruptedException……..etc  The classes that extends Exception class are called checked exceptions.

import java.io.\*; class Test

{ public static void main(String[] args)

{ FileInputStream fis = new FileInputStream("abc.txt"); **FileNotFoundException**

Thread.sleep(1000); **InterruptedException**

}

}

* If you are trying to compile the above compilation the compiler will show the compilation error. error: unreported exception FileNotFoundException; must be caught or declared to be thrown
* If the application contains checked Exception code is not compiled, the compiler will give the exception information in the form of compilation error but exception occurred at runtime.
* To overcome above problem to compile the application must declare the try-catch blocks or throws keyword then only code is compiled .

**Note**: **Whether it is a checked Exception or unchecked exception exceptions are raised at runtime but not compile time.**

**Note**: **whether it is a checked Exception or unchecked Exception must handle the Exception by using try-catch blocks or throws keyword to get normal termination of application & to execute rest of the application.**

##### Checked Exception scenarios:-

1. **java.lang.InterruptedException**

When we used **Thread.sleep(2000);**your thread is entered into sleeping mode then other threads are able to interrupt then the program is terminated abnormally & rest of the application is not executed.

To overcome above problem compile time compiler is checking that exception & displaying exception information in the form of compilation error.

Based on compiler generated error message write the try-catch blocks or throws , if runtime any exception raised the try-catch or throws keyword executed program is terminated normally.

1. **Java.io.FileNotFoundException**

If we are trying to read the file from local disk but at runtime if the file is not available program is terminated abnormally rest of the application is not executed.

To overcome above problem compile time compiler is checking that exception & displaying exception information in the form of compilation error.

Based on compiler generated error message write the try-catch blocks or throws , if runtime any exception raised the try-catch or throws keyword executed program is terminated normally.

1. **Java.sql.SQLException**

If we are trying to connect to data base but at runtime data base is not available program is terminated abnormally rest of the application is not executed.

To overcome above problem compile time compiler is checking that exception & displaying exception information in the form of compilation error.

Based on compiler generated error message write the try-catch blocks or throws , if runtime any exception raised the try-catch or throws keyword executed program is terminated normally.

#### Exception vs.Error:-

* The exception are occurred due to several reasons
  1. Developer mistakes
  2. End-user input mistakes.
  3. Resource is not available
  4. Networking problems.

But errors are caused due to lack of system resources.

StackOverFlowError, OutOfMemoryError, AssertionError…………etc

* It is possible to handle the exceptions by using try-catch blocks or throws keyword but it is not possible to handle the errors.  Error is an un-checked type exception. class Test

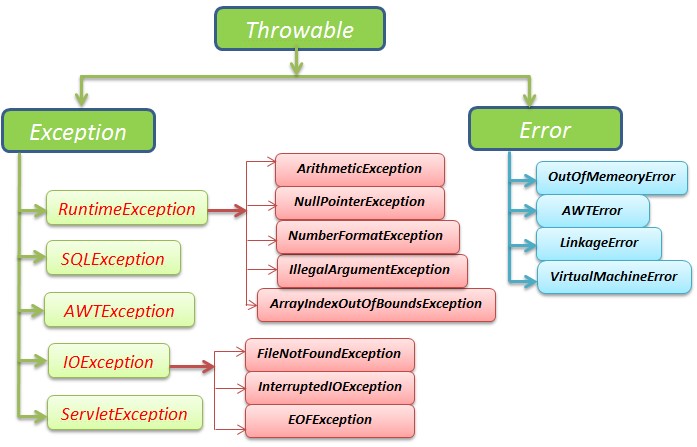
{ public static void main(String[] args)

{ Test[] t = new Test[100000000]; **" java.lang.OutOfMemoryError”**

}

};

#### Exception Handling Tree Structure:-



* The root class of exception handling is **Throwable** class.
* In above tree Structure RuntimeException its child classes& Error its child classes are Unchecked remaining all exceptions are checked Exceptions.

Exception Handling keywords:-1. Try

* 1. Catch
  2. Finally
  3. Throws
  4. Throw

**Default exception handler :-**

Whenever the exception raised default exception handler is responsible to create exception object & print the excpetion message.

##### Fully Checked vs partially checked:-

* The root class& all its child class are checked then that root class is called **fully checked exception.**  IOException,SQLException….etc
* The root class contains some child classes are checked exceptions & some child classes are unchecked exception then that root class is called partially checked exception.

Exception , Throwable..etc

**There are two ways to handle the exceptions in java.**

* 1. By using try-catch block.
  2. By using throws keyword.

##### Exception handling by using Try –catch blocks:-

**Syntax:-**try

{exceptional code;

}

catch (Exception\_Name reference\_variable) { Alternate code : Code to run if an exception is raised

}

**Example-1 :-**Whenever exception raised in the try block JVM will check the corresponding catch block,

1. If the catch block is matched then that block will be executed & rest of the application executed & program is terminated normally.
2. If the catch block is not matched program is terminated abnormally.

|  |  |
| --- | --- |
| **Application without try-catch blocks** class Test  { public static void main(String[] args)  { System.out.println("balu");  System.out.println(10/0);  System.out.println("rest of the application");  }  }  E:\>java Test balu  Exception : ArithmeticException: / by zero    **Disadvanatges**   1. program terminated abnormally. 2. rest of the application notexecuted. | **Application with try-catch blocks:-** class Test  { public static void main(String[] args)  { System.out.println("balu");  try  {System.out.println(10/0);  }  catch (ArithmeticException ae)  { System.out.println(10/2);  }  System.out.println("rest of the application");  }  }  E:\>java Test balu 5 rest of the application **Advanages :**   1. program terminated normally 2. rest of the application executed/ |

**Example-2 :-** In below example catch block is not matched hence program is terminated abnormally.

try

{ System.out.println("sravya");

System.out.println(10/0);

}

catch(NullPointerException e) { System.out.println(10/2);

}

**Example 3:-**If there is no exception in try block thecorresponding catch blocks are not checked.

class Test

{ public static void main(String[] args)

{ try

{ System.out.println("sravya");

}

catch(NullPointerException e) { System.out.println(10/2);

}

System.out.println("rest of the app");

}

}

**E:\sravya>java Test sravya**

**rest of the app**

**Example 4:-**

* In Exception handling independent try blocks declaration are not allowed must declare **try-catch** or**try- finally** or **try with resources.** class Test

{ public static void main(String[] args)

{ try

{ System.out.println("sravya");

}

System.out.println("rest of the app");

}

}

**E:\sravya>javac Test.java**

**Test.java:4: 'try' without 'catch' or 'finally' or resources**

**Example 5:-**

* In between try-catch blocks it is not possible to declare any statements, if we are declaring statements compiler will generate error message.
* In between any two blocks statements delaration nor possible.

try

{ System.out.println(10/0);

}

System.out.println("anu"); catch(ArithmeticException e) { System.out.println(10/2);

}

**Example 6:-**

* If the exception raised in other than try block it is always abnormal termination.
* In below example exception raised in catch block hence program is terminated abnormally.

catch(ArithmeticException e) { System.out.println(10/0);

}

**Example 7:-**

* If the exception raised in try block the remaining code of try block is not executed.
* Once the control is out of the try block the control never entered into try block once again.
* Don’t take normal code inside try block because no guarantee all statements in try-block will be executed or not.

|  |  |  |
| --- | --- | --- |
| class Test  { public static void main(String[] args)  { try{  System.out.println("durga");  System.out.println("balu");  **System.out.println(10/0);**  }  catch(ArithmeticException e)  { System.out.println(10/2);  }  System.out.println("rest of the app");  }  }  E:\sravya>java Test  Durga balu rest of the app |  | class Test  { public static void main(String[] args)  { try{  **System.out.println(10/0);**  System.out.println("durga");  System.out.println("balu");  }  catch(ArithmeticException e)  { System.out.println(10/2);  }  System.out.println("rest of the app");  }  }  E:\sravya>java Test  5  rest of the app |

**Example 8: -** is it possible to pass exceptions as method parameters.import java.io.\*; class Test

{ void m1(ArithmeticException e)

{ System.out.println("m1 method code="+e);

}

void m1(Exception ee)

{ System.out.println("m2 method code="+ee);

}

public static void main(String[] args)

{ Test t = new Test();

t.m1(new ArithmeticException());

t.m1(new IOException());

}

}

**Example 9 :-**There are three methods to print Exception information

1. toString()
2. getMessage()
3. printStackTrace()

class Test

{ void m3()

{ try{ System.out.println(10/0);

} catch(ArithmeticException ae) { System.out.println(ae.toString()); System.out.println(ae.getMessage()); ae.printStackTrace();

}

}

void m2()

{ m3();

}

void m1()

{ m2();

}

public static void main(String[] args)

{ new Test().m1();

}

}

D:\DP>java Test

java.lang.ArithmeticException: / by zero **//toString() method output**

/ by zero **//getMessage() method output**

java.lang.ArithmeticException: / by zero **//printStackTrace() method**  at Test1.m3(Test1.java:8) at Test1.m2(Test1.java:5) at Test1.m1(Test1.java:3) at Test1.main(Test1.java:17)

**Internally JVM uses printStackTrace() method to print exception information.**

**Example 10 : Exception propagation**

If the exception raised in top of the stack method but if you are not handled it drops down to the stack previous method, if you are not catch it drop down until end of the stack(up to main method) this is called exception propagation.

**Note: only the unchecked Exceptions are propagated automatically but not checked.**

class Test

{ void m3()

{ System.out.println(10/0);

}

void m2()

{ m3();

}

void m1()

{ try{ m2(); }

catch(ArithmeticException ae)

{ System.out.println("Arithmetic Exception propagation.....");

}

}

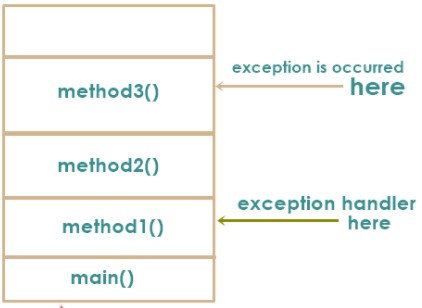
public static void main(String[] args)

{ new Test().m1();

}

}

* In above example the exception raised in m3() method but it is not handled so it is propagated to m2() method.
* Here the m2() method is not handled exception so it is propagated to m1().
* In above example m1() is handled exception.



##### Category-1

**Case 1 :** The way of handling the exceptions is varied from exception to the exception hence it is recommended to write try with multiple catch blocks.

import java.util.\*; class Test

{ public static void main(String[] args)

{ Scanner s=new Scanner(System.in); **//Scanner object used to take dynamic input**

System.out.println("provide the division value"); int n=s.nextInt();

try

{ System.out.println(10/n);

System.out.println("balu".charAt(10));

}

catch (ArithmeticException ae) { System.out.println("Balu soft");

}

catch (StringIndexOutOfBoundsException se)

{ System.out.println("durga soft");

}

System.out.println("rest of the code");

}

}

|  |  |
| --- | --- |
| **Output:- provide the division value: 5**  **Write the output** | **Output:- provide the division value: 0**  **Write the output** |

**Case 2 :**By using **Exception** class catch block it is possible to hold any type of exceptions. import java.util.\*; class Test

{ public static void main(String[] args)

{ Scanner s=new Scanner(System.in); //Scanner object used to take dynamic input

System.out.println("provide the division value"); int n=s.nextInt();

try{ System.out.println(10/n); System.out.println("balu".charAt(10));

}

catch (Exception ae)

{ System.out.println("Balusoft");

}

System.out.println("rest of the code");

}

}

|  |  |
| --- | --- |
| **Output:- provide the division value: 5**  **Write the output** | **Output:- provide the division value: 0**  **Write the output** |

**Case 3:**When we declare multiple catch blocks then the catch block order must be **child-parent**.

import java.util.\*;

class Test

{ public static void main(String[] args)

{ Scanner s=new Scanner(System.in);

System.out.println("provide the division val"); int n=s.nextInt();

try

{ System.out.println(10/n);

System.out.println(“balu”.charAt(20));

}

**//catch block order is child to parent**  catch (ArithmeticException ae) { System.out.println("Exception"+ae);

}

catch (Exception ne)

{ System.out.println("Exception"+ne);

}

System.out.println("rest of the code");

}

}

**Case 4 : Invalid : catch block order is parent to child compiler generate error message.**

try

{ System.out.println(10/n);

System.out.println(“balu”.charAt(20));

}

catch (Exception ne)

{ System.out.println("Exception"+ne);

}

catch (ArithmeticException ae) { System.out.println("Exception"+ae);

}

G:\>javac Test.java error: exception ArithmeticException has already been caught

**Category -2 (pipe symbol)**

**Case 1:-**

It is possible to handle more than one exception in single catch by using pipe(|) symbol.**(java 7 version)** catch(ArithmeticException | StringIndexOutOfBoundsException a) .

catch(NumberFormatException | NullPointerException | StringIndexOutOfBoundsException a)

import java.util.Scanner; import java.io.\*; public class Test

{ public static void main(String[] args)

{ Scanner s = new Scanner(System.in); System.out.println("enter a number"); int n = s.nextInt();

try { System.out.println(10/n); System.out.println("balu".charAt(13));

}

catch(ArithmeticException | ClassCastException a)

{ System.out.println("exception info="+a);

}

catch(NumberFormatException | NullPointerException | StringIndexOutOfBoundsException a)

{ System.out.println("exception info="+a);

}

System.out.println("Rest of the application");

}

}

* When we declared unchecked exception in catch block by using pipe symbol,those exceptions are not mandatory to present in try block.
* In above example, o In catch we declared ArithmeticException but this exception raised in try block.(chance) o In catch we declared ClassCastException but this exception not raised in try block.(no chance)

**Case 2** :- import java.io.\*;

class Test

{ public static void main(String[] args)

{ try

{ FileInputStream f = new FileInputStream("abc.txt");

}

catch(FileNotFoundException|InterruptedException a)

{ System.out.println("exception info="+a);

}

}

}

**error: exception InterruptedException is never thrown in body of corresponding try statement**

* When we declared checked exception in catch block by using pipe symbol,those exceptions must present in try block otherwise compiler generates error message.
* In catch block two exceptions are declared but try block contains one exception hence compiler generates error message. **Case 3:- valid**

import java.io.\*;

class Test

{ public static void main(String[] args)

{ try

{ FileInputStream f = new FileInputStream("abc.txt");**// FileNotFoundException**

Thread.sleep(1000); **// InterruptedException**

}

catch(FileNotFoundException|InterruptedException a)

{ System.out.println("exception info="+a);

}

}

}

* When we declared checked exception in catch block by using pipe symbol,those exceptions must present in try block**.**
* In catch block two exceptions are declared & try block contains those two exceptions it is valid..

**Case-4 :**By using pipe symbol possibleit is possible to declare the both checked exceptions & un-checked exception but checked exceptions must be present in try block.

public static void main(String[] args)

{ try

{ FileInputStream f = new FileInputStream("abc.txt");

Thread.sleep(1000);

}

catch(FileNotFoundException|InterruptedException|ArithmeticException a)

{ System.out.println("exception info="+a);

}

}

**Case 5:-**

* It is not possible to declare the both parent & child classes by using pipe symbol.
* In above case declare only parent class exception.

Here the FileNotFoundException is the child class of IOException

**Invalid : - valid : -**

catch(FileNotFoundException | IOException a) catch( IOException a)

{ System.out.println("exception info="+a); { System.out.println("exception info="+a);

} }

##### Category-3 try with resources (java 7 version)

**Case 1:**

* When we declare the resource by using try block once the try block is completed resource is released.
* When we declare the try with resource, if the resource is throws unchecked exception in this case catch block is optional.
* In below example we declared scanner class as a resource it may raise InputMissmatchException & it is unchecked exception hence catch block is optional.

import java.util.\*; class Test

{ public static void main(String[] args)

{ try(Scanner s = new Scanner(System.in)) { System.out.println("enter id"); int a = s.nextInt();

System.out.println("input value="+a);

}

}

}

**Case 2:**

* When we declare the try with resource, if the resource is throws checked exception in this case catch block is mandatory.
* In above example we declared File resource it throws FileNotFoundException it is a checked exception hence catch block is mandatory.

import java.io.\*;

class Test

{ public static void main(String[] args)

{ try(FileInputStream fis = new FileInputStream("abc.txt"))

{ System.out.println("reading data from text file");

}

catch (FileNotFoundException e)

{ System.out.println("file in not available");

}

}

}

**Case 3:**

* By using try block it is possible to declare more than one resource but every resource is separated with semicolon.
* If the try block contains more than one resource in those resources at least one resource throws checked exception in such case catch block is mandatory.

try(Scanner s = new Scanner(System.in);FileInputStream fis = new FileInputStream("abc.txt"))

{ **//some code here**

}

catch (FileNotFoundException e)

{ //some code here

}

**Possibilities of try-catch:- Case-1**

###### try Case-3 try

{ try { } { }

catch () } catch () { catch () { try

} { { } }

catch () catch ()

{ {

} }

**Case-2**  } try

###### { Case-6

} try

catch () { try { **Case-4**  { } try }

try { try catch ()

{ { { } } }

catch () catch () }

{ { catch ()

} } { try

} {

catch () } { catch ()

} {

}

**Case-5** }

#### Finally block:-

 Finally block code is always executed irrespective of try and catch block code. It is used to write the resource releasing code like,

|  |  |  |
| --- | --- | --- |
| **a.** connection closing. |  | **Connection.close();** |
| **b.** streams closing. |  | **inputstreamclose();** |
| **c.** channel closing |  | **scanner.close();** |
| **d.** Object destruction . |  | **Test t = new Test();t=null;** |

**Finally block Syntax:-** try

{ risky code;

}

catch (Exception obj)

{ code to be run if the exception raised (**handling code);**

} finally

{ **Clean-up code**;(database connection closing , streams closing……etc)

}

**final vs return statement :** if the try,catch,finally contains return statement the return value will be finally block return value.

class Test

{ int m1()

{ try

{ return 10;

}

catch(Exception e)

{ return 20;

}

finally { return 30;

}

}

public static void main(String[] args)

|  |  |  |
| --- | --- | --- |
| } | { int a = new Test().m1();  System.out.println("return value="+a);  } | **Output G:\>java Test return value=30** |

##### All possibilities of finally block execution :-

|  |  |  |
| --- | --- | --- |
| **Case 1:-**  try  { System.out.println("try");  }  catch (ArithmeticException ae)  { System.out.println("catch");  }  finally  { System.out.println("finally");  }  **Output:- Try finally** | **case 2:-** | |
|  | try  { System.out.println(10/0);  }  catch (ArithmeticException ae)  { System.out.println("catch");  } finally  { System.out.println("finally");  }  **Output:- catch finally** |

##### case 3:-

try

{ System.out.println(10/0);

}

catch (NullPointerException ae)

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

}

finally

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

}

Output: finally

Exception in thread "main" java.lang.ArithmeticException: / by zero at Test.main(Test.java:4)

##### case 4:-

try

{ System.out.println(10/0);

}

catch (ArithmeticException ae)

{ System.out.println(10/0);

}

finally

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

}

Output: finally

Exception in thread "main" java.lang.ArithmeticException: / by zero at Test.main(Test.java:7)

##### case 5:-

try

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

}

catch(ArithmeticException ae) { System.out.println("catch");

}

finally { System.out.println(10/0);

}

System.out.println("rest of the code");

Output:- try

Exception in thread "main"

java.lang.ArithmeticException: / by zero

**In two cases finally block won’t be executed case 6:-it is possible to provide try-finally.**

try

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

}

finally

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

}

System.out.println("rest of the code");

Output:- try finally

rest of the code

**Case 1:** whenever the control is entered into try block then only finally block will be executed otherwise it is not executed.

class Test

{ public static void main(String[] args)

|  |  |  |
| --- | --- | --- |
|  | { | System.out.println(10/0); |
|  |  | try  { System.out.println("balu");  } |
|  |  | finally |
|  |  | { System.out.println("finally block");  } |
|  |  | System.out.println("rest of the code"); |
| } | } |  |
| **Case 2:** | | In your program when we used System. Exit (0) the JVM will be shutdown hence the rest of the code won’t be executed . |

class Test

{ public static void main(String[] args)

{ try{ System.out.println("balu");

System.exit(0);

}

finally

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

}

System.out.println("rest of the code");

}

};

**D:\>java Test**

**Balu**

**Example :-**if the try,catch finally blocks contains exception the default exception handler is able to display only one exception at a time that most recently raised. try

{ System.out.println(10/0);

}

catch(Exception e)

{ System.out.println("balu".charAt(20));

}

finally

{ int[] a={10,20,30};

System.out.println(a[9]);

}

G:\>java Test

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Array index out of range: 9

**Example :-** statement 1

|  |  |
| --- | --- |
| statement 2  try  { statement 3  statement 4  try  { statement 5 statement 6  }  catch ()  { statement 7 statement 8  }  }  catch ()  { statement 9 statement 10  try  { statement 11 statement 12  }  catch ()  { statement 13 statement 14  }  }  Finally{ statement 15 statement 16  }  Statement -17  Statement -18 | Case1: No Exception in the above example.  **1, 2, 3, 4, 5,6,15,16,17,18 Normal Termination**    Case 2:- if the exception is raised in statement 2.  **1 , Abnormal Termination**    Case 3:-Exception is raised in the statement 3 the corresponding catch block Is not matched.  **1,2,15,16Abnormal termination**    Case 4:- Exception is raise in the statement-4 corresponding catch block is matched.  **1,2,3,9,10,11,12,15,16,17,18NT**    Case 5:-Exception is raised in the statement 5 and corresponding catch block is matched.  **1,2,3,4,7,8,15,16,17,18 NT**    Case 6:-If the exception is raised in the statement 6 and corresponding catch block is notmatched but outer catch block is matched.  **1,2,3,4,5,9,10,11,12,15,16,17,18 NT** |
| Case 7:-If the exception is raised in the statement 5 and the corresponding catch block is matched but while executing catch block exception raised in statement-7, the outer catch block is matched while executing outer catch exception raised in st-11, the inner catch block is matched but while executing inner catch the exception raised in st-13.  **1,2,3,4,9,10,15,16Abnormal termination.**    Case 8:-If the exception is raised in the statement 6 and the corresponding catch block is matched but while executing catch block exception raised in statement-8, the outer catch block is matched while executing outer catch exception raised in st-12, the inner catch block is matched but while executing inner catch the exception raised in st-14.  **1,2,3,4,5,7,9,10,11,13,15,16Abnormal termination.**    Case 9:- If the exception raised in statement 15. **1,2,3,4,5 Abnormal termination.**  Case 10:- if the Exception raised in statement 18. **1,2,3,4,5,6,15,16,17 Abnormal termination.** | |

#### Throws keyword:-

There are two approaches two handle the exceptions in java

1. By using try-catch blocks.
2. By using throws keyword.

##### Handling exception by using Try-catch Handling Exception by using throws keyword

1. Try-catch blocks are used to write the Throws keyword is used to delegate the exception handling code. responsibilities of exception handling to caller method.
2. By using try-catch blocks it is possible to By using throws it is possible to handle multiple handle multiple exceptions by using multiple exceptions because one method is able to catch blocks. throws multiple exceptions at time.
3. We can write the try-catch blocks at method We can write the throws keyword at method implementation level. declaration level.
4. We can provide the try-catch blocks at We can provide the throws keyword only at method & constructor & blocks level. method & constructor level but not block level.

**Example 1:** class Test

{ void studentDetails() **throws InterruptedException**

{ System.out.println("suneel babu is sleeping");

Thread.sleep(3000);

System.out.println("do not disturb sir......");

}

void hod()**throws InterruptedException**

{ studentDetails();

}

void principal()

{ try{hod();}

catch(InterruptedException ie)

{ ie.printStackTrace(); }

}

void officeBoy()

{ principal();

}

public static void main(String[] args)

{ Test t = new Test(); t.officeBoy();

}

}

* in above example exception raised in studentDetails() method but it delegating responsibilities of exception handling to hod() method by using throws keyword.
* But hod() method delegating responsibilities of exception handling to principal() method by using throws now principal handing this exception by using try-catch blocks.

**Example 2:-**

class Test

{ void studentDetails() **throws InterruptedException**

{ System.out.println("suneel babu is sleeping");

Thread.sleep(3000);

System.out.println("do not disturb sir......");

}

void hod()**throws InterruptedException**

{ studentDetails();

}

void principal()**throws InterruptedException**

{ hod();

}

void officeBoy()**throws InterruptedException**

{ principal();

}

public static void main(String[] args) **throws InterruptedException**

{ Test t = new Test(); t.officeBoy();

}

}

* In above example method-by-method using throws keyword to delegate responsibilities of exception handling to caller method.
* At final main() method uses throws keyword to delegate the responsibilities of exception handling to JVM.

**Example 3:- One method is able to throws more than one exception.**

import java.io.\*; class Test

{ void m2()throws FileNotFoundException,InterruptedException

{ FileInputStream fis = new FileInputStream("abc.txt");

Thread.sleep(2000);

System.out.println("Exceptions are handled");

}

void m1() { try{m2();}

catch(FileNotFoundException | InterruptedException f){f.printStackTrace();}

}

public static void main(String[] args)

{ Test t = new Test();

t.m1();

}

}

**Example 4:- The root class is able to throws all exceptions (Exception root class)**

import java.io.\*;

class Test

{ void m2()**throws Exception**

{ FileInputStream fis = new FileInputStream("abc.txt");

Thread.sleep(2000);

System.out.println("Exceptions are handled");

}

void m1() { try{m2(); }

catch(Excetpion e){ e.printStackTrace(); }

}

public static void main(String[] args)

{ Test t = new Test();

t.m1();

}

}

**Example 5:-** import java.io.\*; class Test

{ void m2()throws FileNotFoundException,InterruptedException

{ FileInputStream fis = new FileInputStream("abc.txt");

Thread.sleep(2000);

System.out.println("Exceptions are handled");

}

void m1()throws InterruptedException

{ try{m2();}

catch(FileNotFoundException fn){fn.printStackTrace();}

}

public static void main(String[] args)

{ Test t = new Test();

try{ m1();}

catch(InterruptedException ie){ie.printStackTrace();}

}

}

* m2() method delegated two exceptions to caller method.
* The caller method is handled one exception(FileNotFoundException)& delegated one exception(InterruptedException) to caller method.
* The main method is handled that exception.

##### Throw keyword:-

In the application whenever the exception raised the JVM will create the exception object & JVM will print predefined exception message.

**Case 1:** It is possible to create the exception object explicitly by developer,

Step 1:- create the Exception object explicitly by the developer by using new keyword.

**new ArithmeticException("balu not eligible");**

Step 2:- handover user created Exception object to jvm by using throw keyword.

**throw new ArithmeticException("balu not eligible");**

The above approach is not recommended because ArithmeticException is predefined exception & it contains some fixed meaning(/ by zero)

**Case 2: InvalidAgeException : userdefined excpetion :-**

Step 1:- create the Exception object explicitly by the developer by using new keyword.

**new InvalidAgeException ("balu not eligible");**

Step 2:- handover user created Exception object to jvm by using throw keyword.

**throw new InvalidAgeException ("balu not eligible");**

the above approach is recommended because user defined must handled by user only.

Note: - throw keyword is used to handover user created exception object to JVM whether it is predefined exception class or user defined exception class but it is always recommended throw custom exception. **Example:-** import java.util.\*; class Test

{ static void validate(int age)

{ if (age>18)

{ System.out.println("eligible for mrg");

}

else

{ throw new ArithmeticException("not eligible for marriage");

}

}

public static void main(String[] args)

{ Scanner s=new Scanner(System.in);

System.out.println("please enter your age ");

Int age = s.nextInt();

Test.validate(age);

System.out.println("rest of the code");

}

}

E:\>java Test E:\>java Test

please enter your age please enter your age

45 10

Check the output Check the output

The above example is not recommended because we are creating object of ArithmeticException but is contains some fixed meaning(/ by zero).

##### Customization of exception handling : user defined exception:-

There are two types of user defined exceptions

1. User defined checked exception.
   1. Default constructor approach.
   2. Parameterized constructor approach.

1. User defined un-checked Exception.
   1. Default constructor approach.
   2. Parameterized constructor approach.

Note: - while declaring user defined exceptions: the naming conventions are every exception suffix must be the word Exception.

##### Creation of userdefined checked Exception by using default constructor approach:-

**Step-1:- create the user defined checked Exception**

Normal java class will become Exception class whenever we are extends Exception class.

**InvaliedAgeException.java:-** package com.tcs.userexceptions;

public class InvalidAgeExcepiton extends Exception

{//default constructor

};

**Step-2:- use the user created Exception in our project.**

##### Test.java

package com.tcs.project;

import com.tcs.userexceptions.InvalidAgeExcepiton;

import java.util.Scanner;

class Test

{ static void status(int age)throws InvalidAgeExcepiton

{ if (age>25)

{ System.out.println("eligible for mrg");

}

else

{ throw new InvalidAgeExcepiton();**//default constructor executed**

}

}

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

{ Scanner s = new Scanner(System.in); System.out.println("enter u r age");//23 int age = s.nextInt();

Test.status(age);

}

}

D:\morn11>java com.tcs.project.Test

enter u r age

20

Exception in thread "main" com.tcs.userexceptions.InvalidAgeExcepiton

**Creation of userdefined checked exception by using parameterized constructor approach.**

**step-1:- create the userdefined checked exception class.**

Normal java class will become checked exception class when we extend Exception class.

**InvalidAgeException.java** package com.tcs.userexceptions; public class InvalidAgeExcepiton extends Exception

{ public InvalidAgeExcepiton(String str)

{super(str); **//super constructor calling in order to print your information**

}

};

**Step-2:- use user created Exception in our project.**

##### Test.java

package com.tcs.project;

import com.tcs.userexceptions.InvalidAgeExcepiton;

import java.util.Scanner;

class Test

{ static void status(int age)throws InvalidAgeExcepiton

{ if (age>25)

{ System.out.println("eligible for mrg");

}

else

{ throw new InvalidAgeExcepiton("not eligible try after some time");

}

}

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

{ Scanner s = new Scanner(System.in); System.out.println("enter u r age"); int age = s.nextInt();

Test.status(age);

}

}

D:\morn11>javac -d . InvalidAgeExcepiton.java

D:\morn11>javac -d. Test.java D:\morn11>java com.tcs.project.Test

enter u r age 28

eligible for mrg

D:\morn11>java com.tcs.project.Test

enter u r age 20

Exception in thread "main" com.tcs.userexceptions.InvalidAgeExcepiton: not eligible try after some time

##### Differences between checked Exception & unchecked Exception:-

**User checked Exception**

1. Our normal java class will become checked Exception class when extends Exception class. class InvalidAgeException extends Exception

{ //logics here

}

1. Must handle the checked Exceptions by using try-catch block or throws keyword.

**User un-checked Exception**

1. Our normal java class will become unchecked Exception class when extends Exception class.

class InvalidAgeException extends RuntimeException

{ //logics here

}

1. Handling unchecked Exceptions is optional but it is recommended.

**Different types of exceptions**

**ArrayIndexOutOfBoundsException:-**  int[] a={10,20,30};

System.out.println(a[4]);//ArrayIndexOutOfBoundsException

**NumberFormatException:-**

String str1="abc"; int b=Integer.parseInt(str1);

System.out.println(b);//NumberFormatException

**NullPointerException:-**

String str1=null;

System.out.println(str1.length());//NullPointerException

**ArithmeticException:-**

int b=10/0;

System.out.println(b);//ArithmeticExceptiom

**IllegalArgumentException:-**

Thread priority range is 1-10

1--->low priority 10--->high priority

Thread t=new Thread();

t.setPriority(11);//IllegalArgumentException

**IllegalThreadStateException:-**  Thread t=new Thread(); t.start();

t.start();//IllegalThreadStateException

**StringIndexOutOfBoundsException:-**  String str="rattaiah";

System.out.println(str.charAt(13));//StringIndexOutOfBoundsException

**NegativeArraySizeException:-**

int[] a=new int[-9];

System.out.println(a.length);//NegativeArraySizeException

**InputMismatchException:-**

Scanner s=new Scanner(System.in); System.out.println("enter first number");

int a=s.nextInt();

**D:\>java Test enter first number**

**balu**

**Exception in thread "main" java.util.InputMismatchException**

**ClassCastException:-**

String s = new String("balu");

Object o = (Object)s;

Object oo = new Object();

String str = (String)oo; **// java.lang.ClassCastException**

**java.lang.NoClassDefFoundError vs java.lang.ClassNotFoundException:-** class Test1

{ void m1(){ System.out.println("Test1 class m1()"); }

}

class Test

{ public static void main(String[] args) throws ClassNotFoundException { Test1 t = new Test1();

t.m1();

Class.forName("Emp");

}

}

**Observation-1:-** In Test class we are hard coding Test1 object but in target location Test1.class file is not available it will generate **java.lang.NoClassDefFoundError.**

**Observation-2:-** In java to load .class file dynamically at runtime we are using forName() method but if runtime the class is not available it generate **java.lang.ClassNotFoundException.**

**Different types of Errors:-**

**StackOverflowError**

**OutOfMemoryError:-**

class Test

{ public static void main(String[] args)

{ int[] a=new int[100000000]; **//OutOfMemoryError**

}

}

**ExceptionInInitializerError:-** static int a=10/0;

ExceptionInInitializerErrorCaused by: java.lang.ArithmeticException: / by zero

**Exception handling interview Questions**

1. What do you mean by Exception?
2. How many types of exceptions in java?
3. What is the difference between Exception and error?
4. What is the difference between checked Exception and un-checked Exception?
5. Is it possible to handle Errors in java?
6. Explain exception handling hierarchy?
7. What the difference is between partially checked and fully checked Exception?
8. What do you mean by exception handling?
9. How many ways are there to handle the exception?
10. What is the root class of Exception handling?
11. Can you please write some of checked and un-checked exceptions in java?
12. What are the keywords present in Exception handling?
13. What is the purpose of try block?
14. In java is it possible to write try without catch or not?
15. What is the purpose catch block?
16. What is the difference between try-catch?
17. Is it possible to write normal code in between try-catch blocks?
18. What are the methods used to print exception messages?
19. What is the purpose of printStackTrace( ) method?
20. What is the purpose of finally block?
21. If the exception raised in catch block what happened?
22. Independent try blocks are allowed or not allowed?
23. Once the control is out of try , is it remaining statements of try block is executed or not?
24. Try-catch , try-catch-catch , catch-catch , catch-try how many combinations are valid?
25. Try-catch-finally , try-finally ,catch-finally , catch-catch-finally how many combinations are valid?
26. Is possible to write code in between try-catch-finally blocks?
27. Is it possible to write independent catch blocks?
28. Is it possible to write independent finally block?
29. What is the difference between try-catch –finally?
30. Is it allows to use nested try-catch in java?
31. For the method argument it us possible to provide exceptions?
32. What do you mean by exception propagation?
33. is the checked Exceptions are propagated or not?
34. If the exception raised in finally block what happened?
35. What are the situations finally block is not executed?
36. What is the relation of finally & return statement.
37. Try,catch,finally three blocks are returning value then which one taken as a final value.
38. What is the purpose of throws keyword?
39. What is the difference between try-catch blocks and throws keyword?
40. What do you mean by default exception handler and what is the purpose of default exception handler?
41. What is the purpose of throw keyword?
42. If we are writing the code after throw keyword usage then what happened?
43. What is the difference between throw and throws keyword?
44. How to create user defined checked exceptions?
45. How to create user defined un-checked exceptions?
46. Where we placed clean-up code like resource release, database closeting inside the try or catch or finally and why ?
47. Write the code of ArithmeticException?
48. Write the code of NullPointerException?
49. Write the code of ArrayIndexOutOfBoundsException & StringIndexOutOfBoundsException?
50. Write the code of IllegalThreadStateException?
51. When we will get InputMisMatchException?
52. When we will get IllegalArgumantException?
53. When we will get ClassCastException?
54. When we will get OutOfMemoryError?
55. What is the difference between ClassNotFoundException & NoClassDefFoundError?
56. When we will get compilation error like “unreportedException must be catch”?
57. When we will get compilation error like “Exception XXXException has already been caught”?
58. When we will get compilation error like “try without catch or finally”?
59. How many approaches are there to create user defined unchecked exceptions and un-checked exceptions?
60. How to create object of user defined exceptions?
61. How to handover user created exception objects to JVM?
62. Is it possible to handle different exceptions by using single catch block yes-->how nowhy?
63. What is the purpose of try with resources?
64. Relation with Exception handling & overriding?
65. How to propagated checked & unchecked Exceptions?

\*\*\*\*\*\*\*\*\*\*\* Thank you \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*