***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.***

*If the application contains exception then the program terminated abnormally then the 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.*

***Reasons for exceptions:-***

 *opening a non-existing file.*

 *Network connection problems.*

 *Values are out of range values*

 *End user input mistakes…….etc*

***Exception Handling:-***

 ***The main objective of exception handling is to get normal termination of the application in order to execute 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***

*javax.servlet.ServletException* ***javax.servlet***

***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*

***Checked Exception:-***

 *The Exceptions which are checked by the compiler at the time of compilation are called Checked Exceptions.*

*Examples:- IOException,SQLException,InterruptedException,ClassNotFoundException……..etc*

 *If the application contains checked Exception the compiler is able to check it and it will give intimation to developer regarding Exception in the form of compilation error.*

 *Handle the checked Exception in two ways*

o *By using try-catch block.*

o *By using throws keyword.*

*In our application whenever we are using exceptional methods the code is not compiled because these methods throws checked exception hence must handle the exception by using try-catch or throws keywords. And no need to remember the methods just use the method compile it then compiler is saying exception information handle it.*

*Note: - If application contains checked Exception then compile time just compiler is displays exception information for handling* ***but the exception raised at runtime*** *if the required resources are not available.*

***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 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.*

***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.*

***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.*

***Note: In above scenarios compile time compiler is display just exception information but exception raised at runtime.***

***Example:-***

*import java.io.\*;*

*class Test*

*{ public static void main(String[] args)*

*{ File f = new File("abc.txt");*

*FileReader fr = new FileReader(f);*

*}*

*}*

*If you are trying to compile the above compilation the compiler will show the compilation error.*

***G:\>javac Test.java***

***Test.java:5: error: unreported exception FileNotFoundException; must be caught or declared to be thrown***

***FileReader fr = new FileReader(f);***

Unchecked Exception:-

 *The exceptions which are not checked by the compiler at the time of compilation are called unchecked Exception.*

*ArithmeticException,ArrayIndexOutOfBoundsException,NumberFormatException….etc*

 *If the application contains un-checked Exception code is compiled but at runtime JVM (Default Exception handler) display exception message then program terminated abnormally.*

 *To overcome runtime problem must handle the exception in two ways.*

o *By using try-catch blocks.*

o *By using throws keyword.*

*class Test*

*{ public static void main(String[] args)*

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

*System.out.println(a[6]);*

*}*

*}*

*G:\>java Test*

*Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 6*

***Note-1:-*** *Whether it is a checked Exception or unchecked exception exceptions are raised at runtime but not compile time.*

***Note 2:-*** *In java 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.*

***Exception vs Error:-***

*The exception are occurred due to several reasons*

*a. Developer mistakes*

*b. End-user input mistakes.*

*c. Resource is not available*

*d. Networking problems.*

 *Exceptions are caused to several reasons like developer mistakes, end user input mistakes, network problems. But error is caused due to lack of system resources.*

*Example: - 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.*

***Example:-***

*class Test*

*{ public static void main(String[] args)*

*{ Test[] t = new Test[100000000];*

*}*

*};*

***Exception in thread "main" java.lang.OutOfMemoryError: Java heap space***



***Exception handling key words:-***

*1) try*

*2) catch*

*3) finally*

*4) throw*

*5) throws*

*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 (ExceptionName reference\_variable)*

*{ Code to run if an exception is raised (alternate code);*

*}*

***Example-1 :-***

***Application without try-catch blocks***

*class Test*

*{ public static void main(String[] args)*

*{ System.out.println("ratan");*

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

*System.out.println("rest of the application");*

*}*

*}*

*E:\>java Test*

*ratan*

Exception in Thread “main” java.lang.ArithmeticException: / by zero

Handled by JVM type of the Exception description

*In above example exception raised program is terminated abnormally & rest of the application is not executed*

*Whenever the exception raised the default exception handler is responsible to handle the exception & it is component of the JVM.*

***Application with try-catch blocks:-***

*Whenever the exception is raised in the try block JVM won’t terminate the program immediately it will search corresponding catch block.*

*a. If the catch block is matched then that block will be executed & rest of the application executed & program is terminated normally.*

*b. If the catch block is not matched program is terminated abnormally.*

*class Test*

*{ public static void main(String[] args)*

*{ System.out.println("ratan");*

*try*

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

*}*

*catch (ArithmeticException ae)*

*{ System.out.println(10/2);*

*}*

*System.out.println("rest of the application");*

*}*

*}*

***In above example we are handling exception by using try-catch block hence the program is terminated normally & rest of the application is executed.***

***Example-2 :-***

*Whenever the exception is raised in the try block JVM won’t terminate the program immediately it will search corresponding catch block.*

*a. If the catch block is matched then that block will be executed & rest of the application executed & program is terminated normally.*

*b. If the catch block is not matched program is terminated abnormally.*

*In below example catch block is not matched hence program is terminated abnormally.*

*class Test*

*{ public static void main(String[] args)*

*{ try*

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

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

*}*

*catch(NullPointerException e)*

*{ System.out.println(10/2);*

*}*

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

*}*

*}*

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

*class Test*

*{ public static void main(String[] args)*

*{ try*

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

*System.out.println("anu");*

*}*

*catch(NullPointerException e)*

*{ System.out.println(10/2);*

*}*

*System.out.println("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-catch-finally.***

*class Test*

*{ public static void main(String[] args)*

*{ try*

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

*System.out.println("anu");*

*}*

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

*}*

*}*

***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 exception handling must declare try with immediate catch block.*

*class Test*

*{ public static void main(String[] args)*

*{ try*

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

*}*

*System.out.println("anu");*

*catch(ArithmeticException e)*

*{ System.out.println(10/2);*

*}*

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

*}*

*}*

***Example 6:-***

 *If the exception raised in try block jvm will search corresponding catch block but if the exception raised other than try-catch blocks it is always abnormal termination.*

 *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.*

*class Test*

*{ public static void main(String[] args)*

*{ try*

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

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

*}*

*catch(ArithmeticException e)*

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

*}*

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

*}*

*}*

***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(10/0);*

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

*System.out.println("ratan");*

*}*

*catch(ArithmeticException e)*

*{ System.out.println(10/2);*

*}*

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

*}*

*}*

***Example 8:-*** *The way of handling the exception is varied from exception to the exception hence it is recommended to provide try with multiple number of 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("ratan".charAt(10));*

*}*

*catch (ArithmeticException ae)*

*{ System.out.println("surya t");*

*}*

*catch (StringIndexOutOfBoundsException se)*

*{ System.out.println("Ravi ");*

*}*

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

*}*

*}*

***Example 9:-*** *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("ratan".charAt(10));*

*}*

*catch (Exception ae)*

*{ System.out.println("Ratansoft");*

*}*

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

*}*

*}*

***Example 10:-*** *When we declare multiple catch blocks then the catch block order must be* ***child-parent*** *but if we are declaring parent to child compiler will generate error message.*

***No compilation error (catch block order child to parent type)***

*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);*

*String str=null;*

*System.out.println(str.length());*

*}*

***//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");*

*}*

*}*

***Compilation error (catch block order is parent to child)***

***//catch block order is parent to child***

*catch (Exception ae)*

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

*}*

*catch (ArithmeticException ne)*

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

*}*

***Example 11:-*** *There are three methods to print Exception information*

 *toString()*

 *getMessage()*

 *printStackTrace()*

*class Test*

*{ void m1()*

*{ m2(); }*

*void m2()*

*{ m3(); }*

*void m3()*

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

*catch(ArithmeticException ae)*

*{ System.out.println(ae.toString());*

*System.out.println(ae.getMessage());*

*ae.printStackTrace();*

*}*

*}*

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

*{ Test1 t = new Test1();*

*t.m1();*

*}*

*};*

***Example 12:-*** *Internally JVM uses printStackTrace() method to print exception information.*

*class Test*

*{ void m3() { System.out.println(10/0); }*

*void m2() { m3(); }*

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

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

*{ new Test().m1();*

*}*

*};*

***Example 13:-***

 *It is possible to combine more than one exception in single catch block.*

 *This is introduced In 1.7 version the code is compiled & executed in 1.7 & above 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("ratan".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");*

*}*

*}*

***Example14:-*** *multiple exception in single catch for checked Exception.*

*import java.io.\*;*

*public class Test*

*{ public static void main(String[] args)*

*{ try {*

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

*Thread.sleep(2000);*

*}*

*catch(FileNotFoundException | InterruptedException a)*

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

*}*

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

*}*

*}*

***Example 15*** *: Above example Observation*

*If you are declared exceptions in catch block but these exceptions are not raised in try block then it raise an error message.*

*import java.io.\*;*

*public class Test*

*{ public static void main(String[] args)*

*{ try*

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

*Thread.sleep(2000);*

*}*

*catch(FileNotFoundException | ClassCastException a)*

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

*}*

*catch(InterruptedException | ClassNotFoundException a)*

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

*}*

*}*

*}*

***Example 16: 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 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();*

*}*

*}*

***Example 17:-***

*Generally finally block is used to close the resources but some time we forgot to close the resources then we will get problems.*

*You can use a finally block to ensure that a resource is closed regardless of whether the try statement completes normally or abruptly. The following example uses a finally block instead of a try-with-resources statement:*

*BufferedReader br = new BufferedReader(new FileReader(path));*

*try { return br.readLine();*

*} finally {*

*if (br != null) br.close();*

*}*

*}*

*To overcome above problem use try-with resources when we declare the resource by using try, when the try-catch block completed resource is automatically closed.*

*try (BufferedReader br = new BufferedReader(new FileReader(path))) {*

*return br.readLine();*

*}*

*BufferedReader will be closed regardless of whether the try statement completes normally or abnormally.*

*import java.util.\*;*

*class Test*

*{ public static void main(String[] args)*

*{ Scanner s=null;*

*try*

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

*System.out.println("enter id");*

*int a = s.nextInt();*

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

*}*

*catch (InputMismatchException ae)*

*{ System.out.println("entered input wrong .......");*

*}*

*finally*

*{ s.close();*

*System.out.println("scanner is closed");*

*}*

***//after closing scanner object***

*System.out.println("enter second id");*

*int b = s.nextInt();*

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

*}*

*}*

***Example 18:-***

*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);*

*}*

*catch (InputMismatchException ae)*

*{ System.out.println("entered input wrong .......");*

*}*

*}*

*}*

***example 19:-*** *while declaring try with multiple resource every resource separated with semicolon.*

*import java.util.\*;*

*import java.io.\*;*

*class Test*

*{ public static void main(String[] args)*

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

*{ System.out.println("enter id");*

*int a = s.nextInt();*

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

*}*

*catch (Exception e)*

*{ System.out.println("entered input wrong .......");*

*}*

*}*

*}*

***Example 20:-***

*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());*

*}*

*}*

Finally block:-

1) *Finally block code is always executed irrespective of try and catch block code*.

*2) It is used to provide clean-up code*

***a.*** *connection closing.* ***Connection.close();***

***b.*** *streams closing.* ***Scanner.close();***

***c.*** *Object destruction .* ***Test t = new Test(); t=null;***

*3) It is not possible to write finally alone.*

*a. try-catch-finally ---> valid*

*b. try-catch ---> valid*

*c. catch-finally ---> invalid*

*d. try-catch-catch-finally ---> valid*

*e. try-finally ---> valid*

*f. catch-catch-finally ---> invalid*

*g. Try ---> invalid*

*h. Catch ---> invalid*

*i. Finally ---> invalid*

***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)*

*}*

***Example:- final vs return statement***

*If the return statement present in try or catch block but finally block executed then only return statement will be executed.*

*class Test*

*{ int m1()*

*{ try*

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

*return 10;*

*}*

*finally*

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

*}*

*}*

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

*{ int a = new Test().m1();*

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

*}*

*}*

***Example*** *:- if the try & catch & finally contains return statement then finally block return statement value is printed.*

*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);*

*}*

*}*

***Example :-*** *when the exception raised in try,catch finally but default exception handler is able to handle only one exception at a time that is most recently raised.*

*class Test*

*{ void m1()*

*{ try*

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

*}*

*catch(Exception e)*

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

*}*

*finally*

*{ System.out.println("ratan".charAt(20));*

*}*

*}*

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

*{ new Test().m1();*

*}*

*}*

***Example:-in two cases finally block won’t be executed***

**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("ratan"); }*

*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("ratan");*

*System.exit(0); }*

*finally*

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

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

*}*

*};*

***Throws keyword:-***

*There are two approaches two handle the exceptions in java*

*a. By using try-catch blocks.*

*b. By using throws keyword.*

***Handling exception by using Try-catch***

*1. Try-catch blocks are used to write the exception handling code.*

*2. By using try-catch blocks it is possible to handle multiple exceptions by using multiple catch blocks.*

*3. We can write the try-catch blocks at method implementation level.*

*4. We can provide the try-catch blocks at method & constructor & blocks level.*

***Handling Exception by using throws keyword***

*Throws keyword is used to delegate the responsibilities of exception handling to caller method*.

*By using throws it is possible to handle multiple exceptions because one method is able to throws multiple exceptions at time.*

*We can write the throws keyword at method declaration level.*

*We can provide the throws keyword only at method & constructor level but not block level.*

***Example 1:-***

 *in below 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.*

*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();*

*}*

*}*

***Example 2:-***

 *In below 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.*

*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();*

*}*

*}*

***Example 3:-***

*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 f) { f.printStackTrace(); }*

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

*}*

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

*{ Test t = new Test();*

*t.m1();*

*}*

*}*

***Example 4:-***

*import java.io.\*;*

*class Test*

*{ void m2()****throws Exception*** *//FileNotFoundException,InterruptedException*

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

*Thread.sleep(2000);*

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

*}*

*void m1()****throws Exception***

*{ m2();*

*}*

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

*{ Test t = new Test();*

*t.m1();*

*}*

*}*

***Exception Handling with Method overriding in java:-***

***Example 1:-***

*If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but it can declare unchecked exception.*

***Case 1:-***

*import java.io.\*;*

*class Parent*

*{ void m1()*

*{ System.out.println("Parent m1()");*

*}*

*}*

*class Child extends Parent*

*{ void m1()throws IOException*

*{ System.out.println("child m1()");*

*}*

*}*

***case 2:-***

*class Parent*

*{ void m1()*

*{ System.out.println("Parent m1()");*

*}*

*}*

*class Child extends Parent*

*{ void m1()throws ArithmeticException*

*{ System.out.println("child m1()");*

*}*

*}*

***Example :2*** *If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.*

***Case 1: Overriding & Overridden method same type of exception.***

*class Parent*

*{ void m1()throws ArithmeticException*

*{ System.out.println("Parent m1()");*

*}*

*}*

*class Child extends Parent*

*{ void m1()throws ArithmeticException*

*{ System.out.println("child m1()");*

*}*

*}*

***Case 2: Overriden method contains Exception but overriding no exception***

*class Parent*

*{ void m1()throws ArithmeticException*

*{ System.out.println("Parent m1()");*

*}*

*}*

*class Child extends Parent*

*{ void m1()*

*{ System.out.println("child m1()");*

*}*

*}*

***Case 3: overriding method is super class & overridden method is sub class***

*class Parent*

*{ void m1()throws Exception*

*{ System.out.println("Parent m1()");*

*}*

*}*

*class Child extends Parent*

*{ void m1()throws ArithmeticException*

*{ System.out.println("child m1()");*

*}*

*}*

***Case 4: overridden method Is sub-type & overriding method is parent-type not allowed generate Compilation error.***

*class Parent*

*{ void m1()throws ArithmeticException*

*{ System.out.println("Parent m1()");*

*}*

*}*

*class Child extends Parent*

*{ void m1()throws Exception*

*{ System.out.println("child m1()");*

*}*

*}*

***Throw keyword:-***

 *It is used to handover user created Exception object to JVM.*

 *It is used to throw exception explicitly.*

 *By using throw keyword it is possible throw predefined Exceptions & custom exception but it is always recommended to throw custom exceptions.*

*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:- throw statement throw an predefined exception.***

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

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

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

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

***Example:-***

*import java.util.\*;*

*class Test*

*{ static void validate(int age)*

*{ if (age<18)*

*{ throw new ArithmeticException("not eligible for vote");*

*}*

*else*

*{ System.out.println("welcome to the voting");*

*}*

*}*

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

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

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

*validate(s.nextInt());*

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

*}*

*}*

***Example: - throw statement throw a user defined exception.***

*To achieve this mechanism first we must know how to create user defined exception then we are able to use this throw keyword.*

*There are two types of exceptions present in the java language*

*1) Predefined Exceptions.*

*ArithmeticException,IOException,NullPointerException…………..etc*

*2) User defined Exceptions.(created by user)*

*InvalidAgeException,MyException…etc*

***Customization of exception handling :-( creation of predefined exceptions)***

*There are two types of user defined exceptions*

*1. User defined checked exception.*

*a. Default constructor approach.*

*b. Parameterized constructor approach.*

*2. User defined un-checked Exception.*

*a. Default constructor approach.*

*b. Parameterized constructor approach.*

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

***Creation of user defined 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);*

*}*

*}*

*Example :-****Creation of user defined checked exception by using parameterized constructor approach.***

***step-1:- create the user defined checked exception class.***

*Normal java class will become checked exception class when we extends 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*

*{* ***//using user created Exception***

*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***

***at com.tcs.project.Test.status(Test.java:11)***

***at com.tcs.project.Test.main(Test.java:18)***

*Example:-****creation of user defined un-checked exception by using default constructor approach***

***Step-1:- create user defined un-checked exception.***

*Normal java class will become un-checked exception class when we exetends RuntimeException class.*

***InvalidAgeException.java***

*package com.tcs.userexceptions;*

*public class InvalidAgeExcepiton extends RuntimeException*

*{ //default constructor*

*}*

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

***Test.java***

*package com.tcs.project;*

*import com.tcs.userexceptions.InvalidAgeExcepiton;*

*import java.util.Scanner;*

*class Test*

*{ static void status(int age)*

*{ if (age>25)*

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

*}*

*else*

*{ //using user created Exception*

*throw new InvalidAgeExcepiton();*

*}*

*}*

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

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

*System.out.println("enter u r age");//23*

*int age = s.nextInt();*

*Test.status(age);*

*}*

*}*

***Example: - creation of user defined un-checked exception by using parameterized constructor approach***

***Step 1:- create user defined un-checked exception classs.***

*Normal java class will become un-checked exception class when we exetends RuntimeException class.*

***InvalidAgeException.java***

*package com.tcs.userexceptions;*

*public class InvalidAgeExcepiton extends RuntimeException*

*{ public InvalidAgeExcepiton(String str)*

*{ super(str);*

*}*

*};*

***Step2:- use user created exception object in your project.***

***Test*.java**

*package com.tcs.project;*

*import com.tcs.userexceptions.InvalidAgeExcepiton;*

*import java.util.Scanner;*

*class Test*

*{ static void status(int age)*

*{ if (age>25)*

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

*}*

*else*

*{* ***//using user created Exception***

*throw new InvalidAgeExcepiton("not eligible for mrg");*

*}*

*}*

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

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

*System.out.println("enter u r age");*

*int age = s.nextInt();*

*Test.status(age);*

*}*

*}*

***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*

*}*

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

***User un-checked Exception***

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

*class InvalidAgeException extends RuntimeException*

*{ //logics here*

*}*

*2. 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***

***ratan***

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

***ClassCastException:-***

*class Test*

*{ public static void main(String[] args)*

*{ String s = new String("ratan");*

*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:-***

*class Test*

*{ void m1()*

*{ m2();*

*System.out.println("this is Rattaiah");*

*}*

*void m2()*

*{ m1();*

*System.out.println("from Sravyasoft");*

*}*

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

*{ new Test().m1();*

*}*

*}*

***OutOfMemoryError:-***

*class Test*

*{ public static void main(String[] args)*

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

*}*

*}*

***ExceptionInInitializerError:-***

*class Test*

*{ static int a=10/0;*

*public static void main(String[] args) { }*

*}*

*Exception in thread "main" java.lang.ExceptionInInitializerError*

*Caused by: java.lang.ArithmeticException: / by zero*

|  |  |
| --- | --- |
| ***Different types of Exceptions in java:- Checked Exception*** | ***Description*** |
| ClassNotFoundException | If the loaded class is not available |
| CloneNotSupportedException | Attempt to clone an object that does not implement the Cloneable interface. |
| IllegalAccessException | Access to a class is denied. |
| InstantiationException | Attempt to create an object of an abstract class or interface. |
| InterruptedException | One thread has been interrupted by another thread. |
| NoSuchFieldException | A requested field does not exist. |
| NoSuchMethodException | If the requested method is not available. |
| ***UncheckedException*** | ***Description*** |
| ArithmeticException | Arithmetic error, such as divide-by-zero. |
| ArrayIndexOutOfBoundsException | Array index is out-of-bounds.(out of range) |
| InputMismatchException | If we are giving input is not matched for storing input. |
| ClassCastException | If the conversion is Invalid. |
| IllegalArgumentException | Illegal argument used to invoke a method. |
| IllegalThreadStateException | Requested operation not compatible with current thread state. |
| IndexOutOfBoundsException | Some type of index is out-of-bounds. |
| NegativeArraySizeException | Array created with a negative size. |
| NullPointerException | Invalid use of a null reference. |
| NumberFormatException | Invalid conversion of a string to a numeric format. |
| StringIndexOutOfBoundsException | Attempt to index outside the bounds of a string. |