**Exception Handling**

Exception handling is used to handle run time errors. We know that errors are of 3 types those are

1. Syntax errors (compile time errors)
2. Run time errors (occurs during program execution)
3. Logical error (if logic is wrong you will get wrong output)

**What happens when a runtime error is occurred?**

|  |
| --- |
| The program will be terminated from the statement where the error is occurred and JVM collects the error information and put that information in an object and throws that object to us. |

**What is an Exception?**

|  |
| --- |
| Exception is an object thrown by JVM during run time error occurs. It contains error information like where and why it was occurred etc.. |

**We can handle the exceptions by using the following 5 keywords**

1. try
2. catch
3. finally
4. throw
5. throws

try

|  |
| --- |
| 1. By using try keyword we can write a block, it is called as try block 2. Try block is used to put the statements which might cause runtime errors. |

­catch block

|  |
| --- |
| 1. Catch block is used to catch the exceptions thrown by the JVM. 2. We can write a catch block either after try or catch block |

Example-1

import java.util.Scanner;

public class ExHandling1

{

    public static void main(String[] args)

    {

        System.out.println("Start of the program...............");

        Scanner scanner=new Scanner(System.in);

        System.out.print("Enter two int values:\t");

        int a=scanner.nextInt();

        int b=scanner.nextInt();

        int c=0;

        try{

            c=a/b;  //100/0 = infinite

        }catch(ArithmeticException ae)

        {   ae.printStackTrace();

        }

        //CIT TEAM CREATES OBJECT FOR ArithmeticException class and puts

        //the error information into that object like why it was occured(cause of the error)

        //,where it was occurred( in which class, in which methods, in which line etc..)

        // and JVM throws that ArithmeticException object to us

        System.out.println("c:\t"+c);

        System.out.println("End of the program...............");

    }

}

//ArithmeticException->RuntimeException->Exception->Throwable

Output:

Start of the program...............

Enter two int values: 10

0

java.lang.ArithmeticException: / by zero

at ExHandling1.main(ExHandling1.java:13)

c: 0

End of the program...............

Example on multiple catch blocks for a single try block

import java.util.InputMismatchException;

import java.util.Scanner;

public class ExHandling1

{

    public static void main(String[] args)

    {

        System.out.println("Start of the program...............");

        Scanner scanner=new Scanner(System.in);

        int c=0;

        try{

            System.out.print("Enter two int values:\t");

            int a=scanner.nextInt();

            int b=scanner.nextInt();

            c=a/b;  //100/0 = infinite

            System.out.println("c:\t"+c);

        }catch(ArithmeticException ae)

        {   ae.printStackTrace();

        }

        catch(InputMismatchException ex)

        {   ex.printStackTrace();

        }

        //CIT TEAM CREATES OBJECT FOR ArithmeticException class and puts

        //the error information into that object like why it was occured(cause of the error)

        //,where it was occurred( in which class, in which methods, in which line etc..)

        // and JVM throws that ArithmeticException object to us

        System.out.println("End of the program...............");

    }

}

//ArithmeticException->RuntimeException->Exception->Throwable

Finally block

1. It is a block which executes every time
2. It doesn’t matter whether the error is occurred or not or error is handled or not but it will get execute every time.
3. In the finally block we write the statements to close the resources like stream closings, database connection closings, socket closing etc..

Example on finally block

import java.util.InputMismatchException;

import java.util.Scanner;

public class ExHandling1

{

    public static void main(String[] args)

    {

        System.out.println("Start of the program...............");

        Scanner scanner=new Scanner(System.in);

        int c=0;

        try{

            System.out.print("Enter two int values:\t");

            int a=scanner.nextInt();

            int b=scanner.nextInt();

            c=a/b;  //100/0 = infinite

            System.out.println("c:\t"+c);

        }catch(ArithmeticException ae)

        {   ae.printStackTrace();

        }

        finally{

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

        }

        //CIT TEAM CREATES OBJECT FOR ArithmeticException class and puts

        //the error information into that object like why it was occured(cause of the error)

        //,where it was occurred( in which class, in which methods, in which line etc..)

        // and JVM throws that ArithmeticException object to us

        System.out.println("End of the program...............");

    }

}

//ArithmeticException->RuntimeException->Exception->Throwable

Output:

Start of the program...............

Enter two int values: 10

2a

finally......block......

Exception in thread "main" java.util.InputMismatchException

at java.base/java.util.Scanner.throwFor(Scanner.java:964)

at java.base/java.util.Scanner.next(Scanner.java:1619)

at java.base/java.util.Scanner.nextInt(Scanner.java:2284)

at java.base/java.util.Scanner.nextInt(Scanner.java:2238)

at ExHandling1.main(ExHandling1.java:14)

Nested try ... catch

|  |
| --- |
| We can write a try block within another try block it is called as nested try |

Example

import java.util.InputMismatchException;

import java.util.Scanner;

public class ExHandling1

{

    public static void main(String[] args)

    {

        System.out.println("Start of the program...............");

        Scanner scanner=new Scanner(System.in);

        int a=0,b=0,c=0;

        try{

            try{

                System.out.print("Enter an int value:\t");

                a=scanner.nextInt();   //10a

            }catch(InputMismatchException ae)

            {   System.out.println("Enter int value for a(last chance:)");

                scanner.next();

                a=scanner.nextInt();

            }

            System.out.print("Enter another int value:\t");

            b=scanner.nextInt();

            c=a/b;  //100/0 = infinite

            System.out.println("c:\t"+c);

        }catch(ArithmeticException ae)

        {   ae.printStackTrace();

        }

        finally{

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

        }

        System.out.println("End of the program...............");

    }

}

Output:

Start of the program...............

Enter an int value: 10a

Enter int value for a(last chance:)

10

Enter another int value: 2

c: 5

finally......block......

End of the program...............

Example to handle all the exceptions by using single catch block

import java.util.InputMismatchException;

import java.util.Scanner;

public class ExHandling1

{

    public static void main(String[] args)

    {

        System.out.println("Start of the program...............");

        Scanner scanner=new Scanner(System.in);

        int c=0;

        try{

            System.out.print("Enter two int values:\t");

            int a=scanner.nextInt();

            int b=scanner.nextInt();

            c=a/b;  //100/0 = infinite

            System.out.println("c:\t"+c);

        }catch(Exception ae)

        {   ae.printStackTrace();

        }

        finally{

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

        }

        System.out.println("End of the program...............");

    }

}

//ArithmeticException->RuntimeException->Exception->Throwable

//InputMismatchException->RuntimeException->Exception->Throwable

**Exception Types**

**There are 2 types of exceptions**

1. Checked exceptions (Unreported Exceptions)
2. Unchecked Exceptions (Reported Exceptions)

Checked Exceptions

|  |
| --- |
| 1. All checked exceptions are direct sub classes of Exception |

**What is the use of throw key word?**

|  |
| --- |
| Which is used to throw the exception by us |

**What is the use of throws keyword?**

|  |
| --- |
| The throws clause or keyword is used to throw an un reported exception out of a method. |

**Example on usage of UserDefined and Checked exception**

import java.util.Scanner;

//checked exception

class MyException extends Exception

{

    MyException(String message)

    {   super(message);

    }

}

class Authentication

{

    public static void loginCheck()throws MyException

    {

        Scanner scan=new Scanner(System.in);

        System.out.print("Email:\t");

        String email=scan.next();

        System.out.print("Password:\t");

        String password=scan.next();

        if(email.equals("madhu@gmail.com") && password.equals("123456"))

        {

            System.out.println("You are authorized user...");

        }else

        {

            throw new MyException("Sachinoda Hacking sestaava...nee lathkoor mokhamla naa cheppu...");

        }

    }

}

public class UserDefinedEx1

{

    public static void main(String[] args) //throws MyException

    {

        System.out.println("start.........................");

        try{

            Authentication.loginCheck();

        }catch(MyException ex)

        {   ex.printStackTrace();

            System.out.println("end.........................");

        }

    }

}

//

Output:

start.........................

Email: madhu@gmail.com

Password: 123

MyException: Sachinoda Hacking sestaava...nee lathkoor mokhamla naa cheppu...

at Authentication.loginCheck(UserDefinedEx1.java:23)

at UserDefinedEx1.main(UserDefinedEx1.java:34)

end.........................

**Creating Reported Exceptions**

|  |
| --- |
| To create Reported Exception Just you need to write a class which is the child class of RuntimeException class |

**Example on creating and using Reported exceptins**

import java.util.Scanner;

//Unchecked exception (reported exception)

class MyException extends RuntimeException

{

    MyException(String message)

    {   super(message);

    }

}

class Authentication

{

    public static void loginCheck()

    {

        Scanner scan=new Scanner(System.in);

        System.out.print("Email:\t");

        String email=scan.next();

        System.out.print("Password:\t");

        String password=scan.next();

        if(email.equals("madhu@gmail.com") && password.equals("123456"))

        {

            System.out.println("You are authorized user...");

        }else

        {

            throw new MyException("Sachinoda Hacking sestaava...nee lathkoor mokhamla naa cheppu...");

        }

    }

}

public class UserDefinedEx1

{

    public static void main(String[] args)

    {

        System.out.println("start.........................");

        try{

             Authentication.loginCheck();

        }catch(MyException ex)

        {   ex.printStackTrace();

        }

        //Authentication.loginCheck();

        System.out.println("end...............................");

    }

}

Output:

start.........................

Email: madhu

Password: 11

MyException: Sachinoda Hacking sestaava...nee lathkoor mokhamla naa cheppu...

at Authentication.loginCheck(UserDefinedEx1.java:23)

at UserDefinedEx1.main(UserDefinedEx1.java:34)

end...............................

//

Output-2:

start.........................

Email: madhu@gmail.com

Password: 123456

You are authorized user...

end...............................

**Catching multiple exceptions by using single catch block**

import java.util.InputMismatchException;

import java.util.Scanner;

public class ExHandling1

{

    public static void main(String[] args)

    {

        System.out.println("Start of the program...............");

        Scanner scanner=new Scanner(System.in);

        int c=0;

        try{

            System.out.print("Enter two int values:\t");

            int a=scanner.nextInt();

            int b=scanner.nextInt();

            c=a/b;  //100/0 = infinite

            System.out.println("c:\t"+c);

        }catch(ArithmeticException | InputMismatchException ae)

        {   ae.printStackTrace();

        }

        finally{

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

        }

        System.out.println("End of the program...............");

    }

}

1. **Chained Exceptions**
2. **Rethrowing an Exception**

**Object class**

|  |
| --- |
| 1. It is the base for all classes in Java 2. This class is existed in java.lang package   Object class methods  public class java.lang.Object {  public java.lang.Object();  public final native java.lang.Class<?> getClass();  public native int hashCode();  public boolean equals(java.lang.Object);  protected native java.lang.Object clone() throws java.lang.CloneNotSupportedException;  public java.lang.String toString();  public final native void notify();  public final native void notifyAll();  public final void wait() throws java.lang.InterruptedException;  public final native void wait(long) throws java.lang.InterruptedException;  public final void wait(long, int) throws java.lang.InterruptedException;  protected void finalize() throws java.lang.Throwable;  } |

**Why we override the toString() method of Object class**

|  |
| --- |
| To see the object state when you print the object |

**Example on overriding toString() method**

class One

{

    int a,b;

    One(int a,int b)

    {   this.a=a;

        this.b=b;

    }

    @Override

    public String toString()

    {

        return "One@[a="+a+",b="+b+"]";

    }

}

public class Demo

{

    public static void main(String[] args)

    {

        One o1=new One(100, 200);

        One o2=new One(1000, 2000);

        System.out.println(o1);

        System.out.println(o2);

    }

}

Output:

One@[a=100,b=200]

One@[a=1000,b=2000]

**Concepts we have covered till now**

* + 1. **Introduction**
    2. **Features**
    3. **Jdk installations**
    4. **VSCode installation**
    5. **JCL**
    6. **Java API**
    7. **Escape sequences**
    8. **Naming conventions**
    9. **Compilation**
    10. **Execution**
    11. **Data types**
    12. **If**
    13. **If..else**
    14. **If..else..if**
    15. **Nested if**
    16. **Switch**
    17. **Array**
    18. **1D arrays**
    19. **2d arrays**
    20. **3d arrays**
    21. **Jagged arrays**
    22. **For loop**
    23. **For each**
    24. **While**
    25. **Do..while..**
    26. **Functions(methods)**
    27. **Class**
    28. **Object**
    29. **Instance variables**
    30. **Static variables**
    31. **Instance methods**
    32. **Static methods**
    33. **Instance block**
    34. **Static block**
    35. **Constructors**
    36. **This keyword**
    37. **String constant pool**
    38. **Accessor and mutator methods**
    39. **Inheritance**
    40. **Super()**
    41. **Method overloading**
    42. **Method overriding**
    43. **Upcasting & Down casting**
    44. **Static and dynamic binding**
    45. **Compile time and runtime Polymorphism**
    46. **Abstract classes**
    47. **Interfaces**
    48. **Inner classes**
    49. **Anonymous inner classes**
    50. **Functional interfaces**
    51. **Lambda expressions**
    52. **Singleton**
    53. **Immutable object**
    54. **Creating record using contextual keyword called ‘record’**
    55. **Exception Handling**
    56. **Try**
    57. **Catch**
    58. **Finally**
    59. **Throw**
    60. **Throws**
    61. **Nested try**
    62. **Types of exceptions**
    63. **User defined exceptions**
    64. **Mult catch exceptions**
    65. **Object class**