# Exception Handling

1) Dictionary meaning of the exception is abnormal termination.

2) An exception is a problem occurred during execution time of the program.

3) An *unwanted* *unexpected* event that disturbs normal flow of execution called exception.

4) Exception is nothing but an object.

5) Exception is a class present **in java.lang package**.

6) All the exceptions are nothing but objects called classes.

7) Whenever user enters invalid data then Exception occurs.

8) A file that needs to be opened can’t found then Exception is occurred.

9) Exception is occurred when the network has disconnected at the middle of the communication.

# Types of Exceptions:-

As per 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 compilation time, for the proper execution of a program at runtime are called CheckedExceptions.

Ex:- IOException,SQLException etc……….

### Some of the checked Exceptions in the java language

1. **ClassNotFoundException** If the loaded class is not available
2. **CloneNotSupportedException** Attempt to clone an object that does not implement the *Cloneable interface*.
3. **IllegalAccessException** Access to a class is denied.
4. **InstantiationException** Attempt to create an object of an abstract class or interface.
5. **InterruptedException** One thread has been interrupted by another thread.
6. **NoSuchFieldException** A requested field does not exist.
7. **NoSuchMethodException** If the requested method is not available.

## Unchecked Exception:-

The exceptions which are not checked by the compiler at compilation time is called unchecked Exception. These checking down at run time only.

Ex:- ArithmeticException,NullPointerException, etc……….

|  |  |
| --- | --- |
| **Exception** | **Description** |
| 1. ArithmeticException | Arithmetic error, such as divide-by-zero. |
| 1. ArrayIndexOutOfBoundsException | Array index is out-of-bounds.(out of range) |
| 1. InputMismatchException | If we are giving input is not matched for storing input. |
| 1. ClassCastException | If the conversion is Invalid. |
| 1. IllegalArgumentException | Illegal argument used to invoke a method. |
| 1. IllegalThreadStateException | Requested operation not compatible with current thread state. |
| 1. IndexOutOfBoundsException | Some type of index is out-of-bounds. |
| 1. NegativeArraySizeException | Array created with a negative size. |
| 1. NullPointerException | Invalid use of a null reference. |
| 1. NumberFormatException | Invalid conversion of a string to a numeric format. |
| 1. StringIndexOutOfBoundsException | Attempt to index outside the bounds of a string. |

## Error:-

Errors are caused by lack of system resources. These are non-recoverable.

Ex:- StackOverFlowError,OutOfMemoryError,AssertionError etc…………

### Good point:-

The Exception whether it is checked or unchecked the Exceptions are occurred at runtime.

## Difference between Exception and Error:-

|  |  |
| --- | --- |
| Exception | Error |
| An exception is unwanted unexpected event. | Errors are caused by lack of system resources . |
| These are caused by the programmer mistake. Exceptions are recoverable. | These are not caused by the programmer mistake. These are non-recoverable. |
| Ex:- IOException,SQLExcetion,RuntimeExecption etc…………… | Ex:- StackOverFlowError,AssertionError etc………… |

Note:-

1) RuntimeException and its child classes and Error and its child classes are Unchecked remaining all are checked Exceptions.

2) Root class for all Exception hierarchy is **Throwable** class.

|  |
| --- |
|  |

## In java Exception handling we are having 5 key words:-

1) try

2) catch

3) finally

4) throw

5) throws

#### We are able to handle the exception in two ways.

1. by using try-catch blocks.

2. by using throws keyword.

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

1. **try** block contains risky code of the program and catch block contains handling code of the program.
2. **catch** block code is an alternative code for Exceptional code. If the exception is raised the alternative code is executed fine then rest of the code is executed normally.

### Before try and catch:-The program goes to abnormal termination.

**Syntax:-**

**try**

{

Code to run [break;]

}

**catch** (**ExceptionName** **reference\_variable**)

{

Code to run if an exception is raised

}

Note:- if we are not using try-catch it is always abnormal termination if an Exception raised.

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

Handled by JVM 🡪 type of the Exception 🡪 Description

### With try catch:

1. If we are taking try-catch the program goes to normal termination. Because the risky code we are taking inside the try block and handling code we are taking inside the catch block.
2. If the exception is raised in the try block the corresponding catch block is executed.
3. If the corresponding catch block is not there program goes to abnormal termination.

|  |
| --- |
| **try**{  **int** b = 10/0;  System.***out***.println(b);    //implicitly Exception object is created by JVM and it is used by the catch to handle it  } **catch**(ArithmeticException ae){    System.***out***.println("Please enter valid number");  **throw** **new** NullPointerException();  } **finally**{  System.***out***.println("Finally ");  } |

|  |
| --- |
| Ex.1  **package** exceptionHandling.examples;  **public** **class** Example1MatchingException{  **public** **static** **void** main(String[] args) {    **try**{  **int** a=10/0;  System.***out***.println(a);  }**catch**(ArithmeticException ae){  System.***out***.println("Enter valid number");  }  }  } |

|  |
| --- |
| Ex.2  **package** exceptionHandling.examples;  **public** **class** Example2NotMatchingException {  **public** **static** **void** main(String[] args) {  **try** {  **int** a = 10/0;  System.***out***.println(a);  } **catch** (NullPointerException e) {  // **TODO**: handle exception  }  **catch** (ArithmeticException e) {  // **TODO**: handle exception  System.***out***.println("Divide by zero");  }  }  } |

|  |
| --- |
| Ex.3  **package** exceptionHandling.examples;  **public** **class** Example3NoException {  **public** **static** **void** main(String[] args) {  **try** {  System.***out***.println("try-->No exception occured.");  } **catch** (Exception e) {  System.***out***.println("catch-->Exception has occured.");  }  }  } |

|  |
| --- |
| Ex.4  **package** exceptionHandling.examples;  **public** **class** Example4OnlyTryOrOnlyCatch {  **public** **static** **void** main(String[] args) {  //Only try is not allowed.  /\*try{  System.out.println("Only try is not allowed.");  }\*/  }  } |

|  |
| --- |
| Ex.5  **package** exceptionHandling.examples;  **public** **class** Example5IndependentStatementsBetweenTryAndCatch {  **public** **static** **void** main(String[] args) {  **try** {  System.***out***.println("try");  }  // Independent statements are not allowed in between try and catch.  // System.out.println("Independent statements");  **catch** (ArithmeticException ae) {  }  // System.out.println("Independent statement not allowed.");  **catch** (Exception e) {  System.***out***.println("Catch");  }  // System.out.println("Independent statement not allowed in between try,  // catch, finally");  **finally** {  System.***out***.println("Finally");  }  System.***out***.println("After finally");  // unreachable exception as parent is coming.  }  } |

|  |
| --- |
| Ex.6   1. Inside the try block once we are getting exception the JVM search for the corresponding catch block. 2. If the catch block is matched then it will be executed. The program is terminated normally the control never goes try block once again. 3. Once the control is out of the try block the control never enters into try block once again.   **package** exceptionHandling.examples;  **public** **class** Example6ControlNeverGoesBack {  **public** **static** **void** main(String[] args) {  **try** {  System.***out***.println("Before exception");  System.***out***.println(10 / 0);  System.***out***.println("After exception ");  } **catch** (Exception e) {  System.***out***.println("catch block");  }  System.***out***.println("After try catch");  }  } |

|  |
| --- |
| Ex.7  **package** exceptionHandling.examples;  **public** **class** Example7RootException {  **public** **static** **void** main(String[] args) {  /\*  \* By using root class (Exception) we are able to hold any type of  \* exceptions. import java.util.\*;  \*/  **try** {  System.***out***.println(10 / 0);  String str = **null**;  System.***out***.println(str.length());  } **catch** (Exception e) {  // **TODO**: handle exception  System.***out***.println("All exceptions are handled by root");  }  }  } |

|  |
| --- |
| **Ex.8**  **package** exceptionHandling.examples;  **public** **class** Example8OrderOfException {  **public** **static** **void** main(String[] args) {  /\*  \* If we are declaring multiple catch blocks, at that situation, the  \* catch block order should be child to parent. shouldn’t be parent to  \* the child.  \*/  **try** {  System.***out***.println(10 / 9);  } **catch** (Exception e) {  System.***out***.println("Root exception");  }  //This is already handled by root.  /\*  \* catch (ArithmeticException ae) {  \* System.out.println("Child Exception"); }  \*/  }  } |

|  |
| --- |
| **Ex.9**  **package** exceptionHandling.examples;  **public** **class** Example9CatchException {  **public** **static** **void** main(String[] args) {  // The exception raised in catch block it is always abnormal  // termination.  **try** {  System.***out***.println(10 / 0);  } **catch** (Exception e) {  // **TODO**: handle exception  System.***out***.println(10 / 0);  }  }  } |

# Possibilities of try-catch blocks:-

1. single time try-catch
2. multiple times try-catch:-
3. try with multiple catches:-
4. nested try-catch
5. catch with try-catch

# Finally block

1. finally is a block it is always executed irrespective of try and catch.
2. Finally contains clean-up code.
3. It is not possible to write finally alone.
4. We must take try-catch-finally otherwise take the try-finally these two are the possibilities.
5. If we are taking any other we are getting compilation error saying finally without try block.

#### Syntax:-

try

{

risky code;

}

catch (Exception obj)

{

handling code;

}

finally

{

free code;

}

|  |
| --- |
| Ex.10: **Exception rose in try block and corresponding catch block is matched then rest of the code is executed normally.**  **package** exceptionHandling.examples;  **public** **class** Example10FinallyDemo {  **public** **static** **void** main(String[] args) {  /\*  \* Exception raised in try block and corresponding catch block is  \* matched then rest of the code is executed normally.  \*/  **try** {  System.***out***.println(10 / 0);  } **catch** (Exception e) {  // **TODO**: handle exception  System.***out***.println("Exception occured");  } **finally** {  System.***out***.println("Finally ");  }  System.***out***.println("Rest of the code");  }  } |

# The only one situation the finally block is won’t be executed:-

In your program whenever we are using **System.exit(0)** the JVM is shut downing hence the rest of the code won’t be executed .

|  |
| --- |
| **Ex.11:- finally block is not executed vs finally block is executed**  **package** exceptionHandling.examples;  **public** **class** Example11FinallyNotRunning {  **public** **static** **void** main(String[] args) {  /\*  \* Exception raised in try block and corresponding catch block is  \* matched then rest of the code is executed normally.  \*/  **try** {  System.***out***.println(10 / 0);  } **catch** (Exception e) {  // **TODO**: handle exception  System.***out***.println("Exception occured");  System.*exit*(0);  } **finally** {  System.***out***.println("Finally ");  }  // this will not be executed.  System.***out***.println("Rest of the code");  }  } |

# Throw keyword

1. The main purpose of the throw keyword is to create an Exception object explicitly either for predefined or user defined.
2. Throw keyword works like a try block. The difference is try block is automatically find the situation and creates an Exception object ***implicitly***. Whereas throw keyword creates an Exception object ***explicitly.***
3. Mostly the main method is responsible to create an exception object. So the main method is creating exception object implicitly. A programmer is not responsible person to create an exception object.

|  |
| --- |
| **Ex.12**  **package** exceptionHandling.examples;  **public** **class** Example12ThrowDemo {  **public** **static** **void** main(String[] args) {  //explicitly programmer is throwing the exception object.  **throw** **new** ArithmeticException("we are getting Exception / by zero man");  }  } |

# Throws keyword

1. Throw keyword is used to create an exception object explicitly. But the main purpose of the **throws** keyword is bypassing the generated exception from present method to caller method.
2. Throw keyword is used in the method body. But **throws** keyword we have to use in the **method declaration**.
3. It is possible to throws any number of exceptions at a time based on the programmer requirement.
4. In the java language we are handling the Exception in two ways
   1. By using try-catch blocks
   2. By using throws keyword

# Exceptions:-

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

1) Predefined Exceptions.

2) User defined Exceptions.

## Predefined Exception:-

Predefined classes come along with the software based on your requirement we have to create objects.

Ex:- ArithmeticException,IOException,NullPointerException…………..etc

## User defined Exceptions:-

Based on the user requirement user can create an Exception is called user defined Exception.

Ex:InvaliedAgeException,BombBlostException………..etc

### To create user defined Exceptions:-

1. To create user defined exception we have to take an user defined class that is a sub class to the **RuntimeException**(***for creation of unchecked Exceptions***) .
2. To create user defined exception we have to take user defined class that is subclass to the **Exception**(***for creation of checked Exceptions***)
3. Each and every Exception contains two constructors
   1. default constructor
   2. parameterized constructor
4. The naming conventions we have to follow
   1. Every exception suffix must be the word **Exception**.
   2. Exception is a class so we have to follow class coding conventions.

# For the creation of Unchecked Exception

### Default constructor approach

Class InvaliedAgeException extends RuntimeException

{

}

### Parameterized constructor approach

Class XXXException extends RuntimeException

{

XXXException(String str)

{

Super(str);

}

}

### Note:-

For this type of user defined Exceptions no need of handling the Exception. Hence try-catch [or] throws keyword is not required.

# For the creation of checked Exception

### Default constructor approach

Class InvaliedAgeException extends Exception

{

}

### Parameterized constructor approach

Class XXXException extends Exception

{

XXXException(String str)

{

Super(str);

}

}

### Note:-

For these type of user defined Exceptions we have to handle the Exception otherwise we will be getting compilation error. Hence try-catch [or] throws keyword is required for the handling of Exceptions.

# Ex:-preparation of custom checked exceptions

|  |
| --- |
| Ex.14  **package** exceptionHandling.examples;  **import** java.util.Scanner;  **public** **class** Example14CustomChecked {  **public** **static** **void** main(String[] args) **throws** InvalidAgeException {  // user defined Exception class preparation  Scanner s = **new** Scanner(System.***in***);  System.***out***.println("please enter age");  **int** age = s.nextInt();  *validateAge*(age);  }  **public** **static** **void** validateAge(**int** age) **throws** InvalidAgeException {  **if** (age < 18) {  System.***out***.println("Not eligible for vote");  } **else** {  System.***out***.println("Eligible fot Vote");  }  }  }  **class** InvalidAgeException **extends** Exception {  InvalidAgeException(String str) {  **super**(str);  }  } |

# Ex:-preparation of custom unchecked exceptions

|  |
| --- |
| package exceptionHandling.examples;  import java.util.Scanner;  class InvalidAgeException extends RuntimeException {  InvalidAgeException(String str) {  super(str);  }  }  public class Example15CustomUnChecked {  public static void main(String[] args) {  // user defined Exception class preparation  Scanner s = new Scanner(System.in);  System.out.println("please enter age");  int age = s.nextInt();  validateAge(age);  }  public static void validateAge(int age) {  if (age < 18) {  //System.out.println("Not eligible for vote");  throw new InvalidAgeException("not elgible for vote");  } else {  System.out.println("Eligible fot Vote");  }  }  } |