Todays Agenda: 16-04-2024

|  |
| --- |
| **Exception Handling** |
| Exception Handling |
| try-catch Blocks: Basic Exception Handling |
| Creating Custom Exceptions: When and How to Create |

Exception Handling:

An exception is an unwanted or unexpected event which occurs during the execution of a program i.e at run time that disrupts the normal flow of the program’s instructions.

**Error:**An Error indicates serious problem that a reasonable application should not try to catch.  
**Exception:**Exception indicates conditions that a reasonable application might try to catch.

All exception and errors types are sub classes of class **Throwable**.

### Types of Java Exceptions:

There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error

1) Checked Exception// compile time

The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

### 2) Unchecked Exception// run time

The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time but they are checked at runtime.

### 3) Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Java Exception Keywords:

|  |  |
| --- | --- |
| Keyword | Description |
| try | The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means we can't use try block alone. |
| catch | The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later. |
| finally | The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not. |
| throw | The "throw" keyword is used to throw an exception. |
| throws | The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that an exception may occur in the method. It is always used with method signature. |

Common Scenarios:

int a=50/0;//ArithmeticException

String s=null;

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

String s="abc";

int i=Integer.parseInt(s);//NumberFormatException

int a[]=new int[5];

a[10]=50; //ArrayIndexOutOfBoundsException

Java try block

Java try block is used to enclose the code that might throw an exception. It must be used within the method.

Java try block must be followed by either catch or finally block.

### Syntax of Java try-catch:

try{

//code that may throw an exception

}

catch(Exception\_class\_Name ref) {

}

### Syntax of try-finally block

try{

//code that may throw an exception

}

finally { }

Java catch block:

Java catch block is used to handle the Exception by declaring the type of exception within the parameter. The declared exception must be the parent class exception ( i.e., Exception) or the generated exception type.

The catch block must be used after the try block only. You can use multiple catch blocks with a single try block.

Java throw keyword:

The Java throw keyword is used to explicitly throw an exception.

We can throw either checked or unchecked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception.

Syntax:

throw exception;

Ex of IOException:

throw new IOException("sorry device error”);

Ex: Here, we have created the validate method that takes integer value as a parameter. If the age is less than 18, we are throwing the ArithmeticException otherwise print a message “welcome to vote”.

public class TestThrow1{

   static void validate(int age) {

     if(age<18)

      throw new ArithmeticException("not valid");

     else

      System.out.println("welcome to vote");

   }

   public static void main(String args[]){

      validate(13);

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

  }

}

# Java throws keyword

The Java throws keyword is used to declare an exception. It gives information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

Syntax: return\_type method\_name() throws exception\_class\_name{

//method code

}

|  |  |  |
| --- | --- | --- |
| **No.** | **throw** | **throws** |
| 1) | Java throw keyword is used to explicitly throw an exception. | Java throws keyword is used to declare an exception. |
| 2) | Checked exceptions cannot be propagated using throw only. | Checked exceptions can be propagated with throws. |
| 3) | Throw is followed by an instance. | Throws is followed by Exception class. |
| 4) | Throw is used within the method. | Throws is used with the method signature. |

# **Java Custom Exception:**

Java provides us the facility to create our own exceptions which are basically derived classes of Exception. Creating our own Exception is known as a custom exception or user-defined exception.

Why use custom exceptions?

To catch and provide specific treatment to a subset of existing Java exceptions.

Business logic exceptions: These are the exceptions related to business logic and workflow. It is useful for the application users or the developers to understand the exact problem.

In order to create a custom exception, we need to extend the Exception class that belongs to java.lang package.

Example: We pass the string to the constructor of the superclass- Exception which is obtained using the “getMessage()” function on the object created.

class MyException extends Exception {

String message;

public MyException(String s)

{

super(s);

}

public String getMessage() {

return this.message;

}

}

public class Main {

public static void main(String args[])

{

try {

throw new MyException("User Defined Exception ");

}

catch (MyException ex) {

System.out.println("Caught");

System.out.println(ex.getMessage());

}

}

}

Output

Caught

User Defined Exception

In the above code, the constructor of MyException requires a string as its argument. The string is passed to the parent class Exception’s constructor using super(). The constructor of the Exception class can also be called without a parameter and the call to super is not mandatory.

Recursion:

A function in java can call itself and that calling is called as Recursion

Eg: factorial of n

5 = 5\*4\*3\*2\*1 4,3,2 🡪 f(n) \* f(n-1)

Fact(0),fact(1) = 1

Public int fact(int num){

If(n==0||n==1)

Return 1;

}else{

n

}

Fact(n) = n\* fact(n-1)

n!= 1;