

UNIT 5 & 6

HANDLING EXCEPTIONS / HANDLING STRINGS

By Yuba Raj Devkota

Exception Handling in Java

to handle the runtime errors so that normal flow of the application can be maintained.

- **Dictionary Meaning:** Exception is an abnormal condition.
- In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.
- Exception Handling is a mechanism to handle runtime errors such as `ClassNotFoundException`, `IOException`, `SQLException`, `RemoteException`, etc.
- Suppose there are 10 statements in your program and there occurs an exception at statement 5, the rest of the code will not be executed i.e. statement 6 to 10 will not be executed. If we perform exception handling, the rest of the statement will be executed. That is why we use exception handling in Java.

Types of Java Exceptions

1.Checked Exception:

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:

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. finally block in Java can be used to put " cleanup " code such as closing a file, closing connection, etc.
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 there may occur an exception in the method. It is always used with method signature.

Example

```
public class JavaExceptionExample{  
    public static void main(String args[]){  
        try{  
            int data=100/0;  
        }catch(ArithmeticException e){System.out.println(e);}  
        System.out.println("rest of the code...");  
    } finally{System.out.println("finally block is always executed");}  
    System.out.println("finally code...");  
}  
  
}
```

Exception in thread main java.lang.ArithmeticException:/ by zero

rest of the code...

Finally code ...

Finally block in java can be used to put "cleanup" code such as closing a file, closing connection etc.

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. We will see custom exceptions later.

Output:

Exception in thread main java.lang.ArithmeticException: not valid

```
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 an 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.
- Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing check up before the code being used.

Output:

exception handled
normal flow...

```
import java.io.IOException;
class Testthrows1{
    void m()throws IOException{
        throw new IOException("device error");//checked exception
    }
    void n()throws IOException{
        m();
    }
    void p(){
        try{
            n();
        }catch(Exception e){System.out.println("exception handled");
        }
    }
    public static void main(String args[]){
        Testthrows1 obj=new Testthrows1();
        obj.p();
        System.out.println("normal flow...");
    }
}
```

```
public class TestThrows {  
    //defining a method  
    public static int divideNum(int m, int n) throws ArithmeticException {  
        int div = m / n;  
        return div;  
    }  
    //main method  
    public static void main(String[] args) {  
        TestThrows obj = new TestThrows();  
        try {  
            System.out.println(TestThrows.divideNum(45, 0));  
        }  
        catch (ArithmeticException e){  
            System.out.println("\nNumber cannot be divided by 0");  
        }  
  
        System.out.println("Rest of the code..");  
    }  
}
```

Java Custom Exception

- we can create our own exceptions that are derived classes of the Exception class.
- Creating our own Exception is known as custom exception or user-defined exception.
- Basically, Java custom exceptions are used to customize the exception according to user need.
- Using the custom exception, we can have your own exception and message. Here, we have passed a string to the constructor of superclass.

Reasons to 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.

```
class InvalidAgeException extends Exception
{
    public InvalidAgeException (String str)
    {
        super(str);
    }
}

public class TestCustomException1
{
    static void validate (int age) throws InvalidAgeException
    {
        if(age < 18){
            throw new InvalidAgeException("age is not valid to vote");
        }
        else {
            System.out.println("welcome to vote");
        }
    }
}
```

```
public static void main(String args[])
{
    try
    {
        validate(13);
    }
    catch (InvalidAgeException ex)
    {
        System.out.println("Caught the exception");
        System.out.println("Exception occurred: " + ex);
    }
    System.out.println("rest of the code...");
}
}
```

Handling Strings

```
public class StringDemo {  
  
    public static void main(String args[]) {  
        char[] helloArray = { 'h', 'e', 'l', 'l', 'o', '.' };  
        String helloString = new String(helloArray);  
        System.out.println( helloString );  
    }  
}
```

Java String Methods

char charAt(int index): It returns the character at the specified index. Specified index value should be between 0 to length() -1 both inclusive. It throws IndexOutOfBoundsException if index<0 || >= length of String.

boolean equalsIgnoreCase(String string): It works same as equals method but it doesn't consider the case while comparing strings. It does a case insensitive comparison.

String substring(int beginIndex): It returns the substring of the string. The substring starts with the character at the specified index.

String substring(int beginIndex, int endIndex): Returns the substring. The substring starts with character at beginIndex and ends with the character at endIndex.

String concat(String str): Concatenates the specified string "str" at the end of the string.

String replace(char oldChar, char newChar): It returns the new updated string after changing all the occurrences of oldChar with the newChar.

boolean contains(CharSequence s): It checks whether the string contains the specified sequence of char values. If yes then it returns true else false. It throws NullPointerException if 's' is null.

String toUpperCase(): Equivalent to toUpperCase(Locale.getDefault()).

Contd...

```
class TestStringConcatenation3{  
  public static void main(String args[]){  
    String s1="Aashish";  
    String s2="Gurung";  
    String s3=s1.concat(s2);  
    System.out.println(s3);// Aashish  Gurung  
  }  
}
```

Contd...

```
public class StringUpperExample{  
public static void main(String args[]){  
String s1="hello string";  
String s1upper=s1.toUpperCase();  
System.out.println(s1upper);  
}}
```

Assignment 3

- Why it is important to handle exception in java? Write a program to illustrate the use of exception handling.
- Define the use of static keyword. Write any four String methods used in java with example.
- Explain different keywords used in exception handling.