

EXPERIMENT-13

Aim: Program on user defined exception.

Theory:

User-defined Exception -

In Java, we have some built-in exception classes like NullPointerException,

ArithmeticException. These exceptions are restricted to trigger on some predefined conditions. In Java, we can write our own exception class by extends the Exception class. We can throw our own exception on a particular condition using the throw keyword. For creating a user-defined exception, we should have basic knowledge of the try-catch block and throw keyword.

In Java, 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 user defined exception; we can have your own exception and message.

Here, we have passed a string to the constructor of superclass i.e. Exception class that can be obtained using getMessage() method on the object we have created.

Why we use custom exceptions -

Java exceptions cover almost all the general type of exceptions that may occur in the programming. However, we sometimes need to create custom exceptions.

Following are few of the 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.

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

To create your own User-defined Exception in Java -

There are the following steps that are followed in creating user-defined exception or custom exception in Java. They are as follows:

Step 1: User-defined exceptions can be created simply by extending Exception class. This is done as -

```
class OwnException extends Exception
```

Step 2: If you do not want to store any exception details, define a default constructor in your own exception class. This can be done as follows -

```
OwnException(){  
}
```

Step 3: If you want to store exception details, define a parameterized constructor with string as a parameter, call superclass (Exception) constructor from this, and store variable “str”. This can be done as follows -

```
OwnException(String str){  
    super(str); //Call super class exception constructor and store the variable “str” in  
    it.  
}
```

Step 4: In the last step, we need to create an object of user-defined exception class

and throw it using throw clause.

```
OwnException obj = new OwnException(“Exception details”);
```

```
throw obj;
```

or,

```
throw new OwnException(“Exception details”);
```

CODE:

```
class MyCustomException extends Exception {

    public MyCustomException(String message) {
        super(message);
    }
}

public class CustomExceptionExample {

    public static void main(String[] args) {

        try {

            int age = 15;

            if (age < 18) {

                throw new MyCustomException("You must be at least 18 years old.");
            }

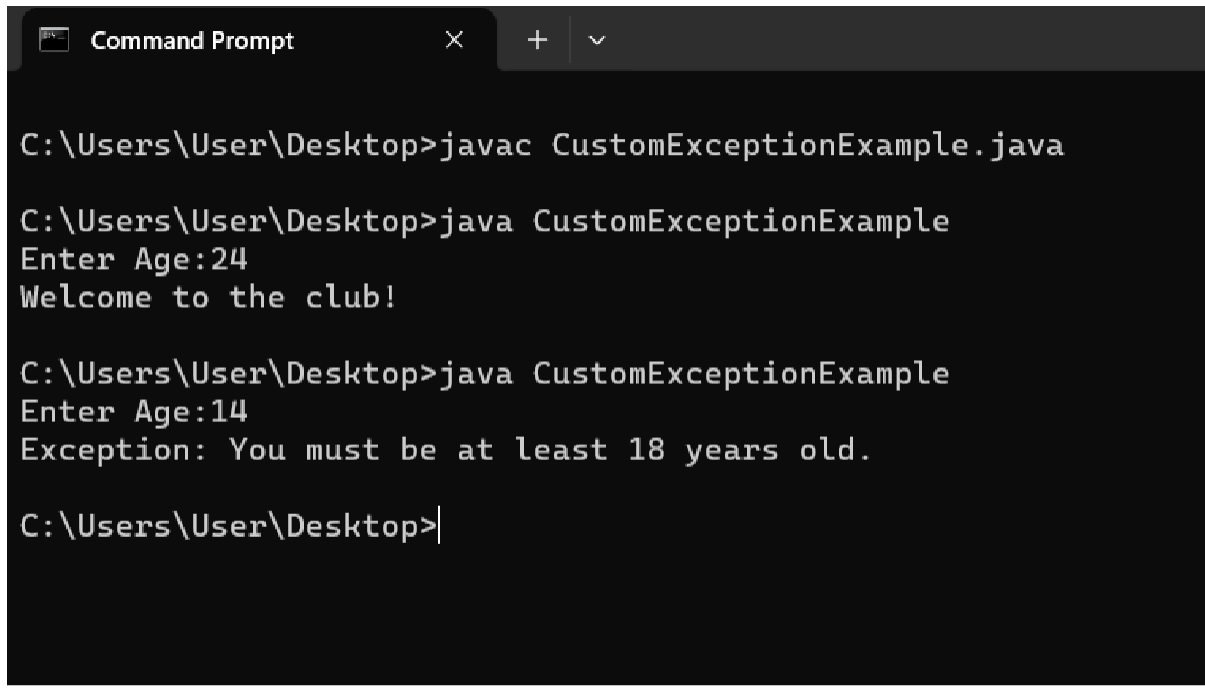
            System.out.println("Welcome to the club!");

        }

        catch (MyCustomException e) { System.out.println("Exception: " +
            e.getMessage());
        }

    }
}
```

OUTPUT:



```
C:\Users\User\Desktop>javac CustomExceptionExample.java

C:\Users\User\Desktop>java CustomExceptionExample
Enter Age:24
Welcome to the club!

C:\Users\User\Desktop>java CustomExceptionExample
Enter Age:14
Exception: You must be at least 18 years old.

C:\Users\User\Desktop>|
```

Conclusion: In conclusion, a program on user-defined exceptions is a valuable tool for improving the robustness and reliability of software systems. By allowing developers to create custom exception classes that capture specific error conditions and provide meaningful error messages, user-defined exceptions enhance the clarity of code and simplify debugging.