User-Defined Custom Exception in Java

Last Updated : 27 Dec, 2024

In Java, an Exception is an issue (run time error) that occurred during the execution of a program. When an exception occurred the program gets terminated abruptly and, the code past the line that generated the exception never gets executed.

Java provides us the facility to create our own exceptions which are basically derived classes of Exception. Creating our own Exception is

Java. Basically, **Java custom exceptions** are used to customize the exception according to user needs. In simple words, we can say that a **User-Defined Custom Exception** or custom exception is creating your own exception class and throwing that exception using the '**throw**' keyword.

Example: In this example, a custom exception **MyException** is created and thrown in the program.

```java
// A Class that represents user-defined exception
class MyException extends Exception {
    public MyException(String m) {
        super(m);
    }
}

// A Class that uses the above MyException
public class setText {
    public static void main(String args[]) {
        try {
            // Throw an object of user-defined exception
            throw new MyException("This is a custom exception");
        }
        catch (MyException ex) {
            System.out.println("Caught");  // Catch and print message
            System.out.println(ex.getMessage());
        }
    }
}
```

Output

```
Caught
This is a custom exception
```

Java Custom Exception

A **custom exception in Java** is an exception defined by the user to handle specific application requirements. These exceptions extend either the Exception class (for checked exceptions) or the RuntimeException class (for unchecked exceptions).

Why Use Java Custom Exceptions?

- To represent application-specific errors.
- To add clear, descriptive error messages for better debugging.
- To encapsulate business logic errors in a meaningful way.

Types of Custom Exceptions

There are two types of custom exceptions in Java.

- Checked Exceptions: It extends the **Exception** class. and it must be declared in the **throws** clause of the method signature.
- Unchecked Exceptions: It extends the RuntimeException class.

Create a User-Defined Custom Exception

1. Create a new class that extends Exception (for checked exceptions) or RuntimeException (for unchecked exceptions).
2. Provide constructors to initialize the exception with custom messages.
3. Add methods to provide additional details about the exception. (this is optional)

Example: Checked Custom Exception

```
// Custom Checked Exception
```

```
class InvalidAgeException extends Exception {
    public InvalidAgeException(String m) {
        super(m);   //message
    }
}

// Using the Custom Exception
public class Geeks {
    public static void validate(int age)
      throws InvalidAgeException {
        if (age < 18) {
            throw new InvalidAgeException("Age must be 18 or above.");
        }
        System.out.println("Valid age: " + age);
    }

    public static void main(String[] args) {
        try {
            validate(12);
        } catch (InvalidAgeException e) {
            System.out.println("Caught Exception: " + e.getMessage());
        }
    }
}
```

Output

```
Caught Exception: Age must be 18 or above.
```

Explanation: The above example defines a custom checked exception **InvalidAgeException** that is thrown when an age is below 18. The **validate()** method checks the age and throws the exception if the age is invalid. In the main() method, the exception is caught and the error message is printed.

Example 3: Unchecked Custom Exception

```
// Custom Unchecked Exception
class DivideByZeroException extends RuntimeException {
    public DivideByZeroException(String m) {
        super(m);
    }
}

// Using the Custom Exception
public class Geeks {
    public static void divide(int a, int b) {
        if (b == 0) {
            throw new DivideByZeroException("Division by zero is not
allowed.");
        }
        System.out.println("Result: " + (a / b));
    }
    public static void main(String[] args) {
        try {
            divide(10, 0);
        } catch (DivideByZeroException e) {
            System.out.println("Caught Exception: " + e.getMessage());
        }
    }
}
```

Output

```
Caught Exception: Division by zero is not allowed.
```

Explanation: The above example defines a custom unchecked exception **DivideByZeroException** that is thrown when we are trying to divide by zero. The **divide()** method checks if the denominator is zero and throws the exception if true. In the **main()** method, the exception is caught and the error message is printed.