# Java catch Multiple Exceptions

In this tutorial, we will learn to handle multiple exceptions in Java with the help of examples.

Before Java 7, we had to write multiple exception handling codes for different types of exceptions even if there was code redundancy.

Let's take an example.

### Example 1: Multiple catch blocks

```
class Main {
  public static void main(String[] args) {
    try {
      int array[] = new int[10];
      array[10] = 30 / 0;
    } catch (ArithmeticException e) {
      System.out.println(e.getMessage());
    } catch (ArrayIndexOutOfBoundsException e) {
      System.out.println(e.getMessage());
    }
  }
}
```

### Output

```
/ by zero
```

In this example, two exceptions may occur:

- `ArithmeticException` because we are trying to divide a number by 0.
- `ArrayIndexOutOfBoundsException` because we have declared a new integer array with array bounds 0 to 9 and we are trying to assign a value to index 10.

We are printing out the exception message in both the `catch` blocks i.e. duplicate code.

The associativity of the assignment operator `=` is right to left, so an `ArithmeticException` is thrown first with the message `/ by zero`.

---

## Handle Multiple Exceptions in a catch Block

In Java SE 7 and later, we can now catch more than one type of exception in a single `catch` block.

Each exception type that can be handled by the `catch` block is separated using a vertical bar or pipe `|`.

Its syntax is:

```
try {
  // code
} catch (ExceptionType1 | Exceptiontype2 ex) {
  // catch block
}
```

### Example 2: Multi-catch block

```
class Main {
  public static void main(String[] args) {
    try {
      int array[] = new int[10];
      array[10] = 30 / 0;
    } catch (ArithmeticException | ArrayIndexOutOfBoundsException e) {
      System.out.println(e.getMessage());
    }
  }
}
```

### Output

```
/ by zero
```

Catching multiple exceptions in a single `catch` block reduces code duplication and increases efficiency.

The bytecode generated while compiling this program will be smaller than the program having multiple `catch` blocks as there is no code redundancy.

**Note:** If a `catch` block handles multiple exceptions, the catch parameter is implicitly `final`. This means we cannot assign any values to catch parameters.

---

## Catching base Exception

When catching multiple exceptions in a single `catch` block, the rule is generalized to specialized.

This means that if there is a hierarchy of exceptions in the `catch` block, we can catch the base exception only instead of catching multiple specialized exceptions.

Let's take an example.

### Example 3: Catching base exception class only

```
class Main {
  public static void main(String[] args) {
    try {
      int array[] = new int[10];
      array[10] = 30 / 0;
    } catch (Exception e) {
      System.out.println(e.getMessage());
    }
  }
}
```

### Output

```
/ by zero
```

We know that all the exception classes are subclasses of the `Exception` class. So, instead of catching multiple specialized exceptions, we can simply catch the `Exception` class.

---

If the base exception class has already been specified in the `catch` block, do not use child exception classes in the same `catch` block. Otherwise, we will get a compilation error.

Let's take an example.

### Example 4: Catching base and child exception classes

```
class Main {
  public static void main(String[] args) {
    try {
      int array[] = new int[10];
      array[10] = 30 / 0;
    } catch (Exception | ArithmeticException | ArrayIndexOutOfBoundsException e) {
      System.out.println(e.getMessage());
    }
  }
}
```

**Output**

**"Main.java:6: error: Alternatives in a multi-catch statement cannot be related by subclassing"**

In this example, `ArithmeticException` and `ArrayIndexOutOfBoundsException` are both subclasses of the `Exception` class. So, we get a compilation error.