Exception Handling

Christian Rodríguez Bustos
Edited by Juan Mendivelso
Object Oriented Programming

Agenda


1.
Exception Handling Basis
2.
Exception Handling Flow
3.
Exception Rules

No matter how good developer you are, you can not control everything.

Your code must be prepared to *handle exceptional situations*

1. Exception Handling Basis

1.1 Exceptions

1.2 Try and Catch

1.3 Finally

1.1 Exceptions

Exceptions are Java run-time errors

Run-time errors (**exceptions**) are common if we do not use **exception handling** to deal with them

Out of bounds array access

Operate null objects

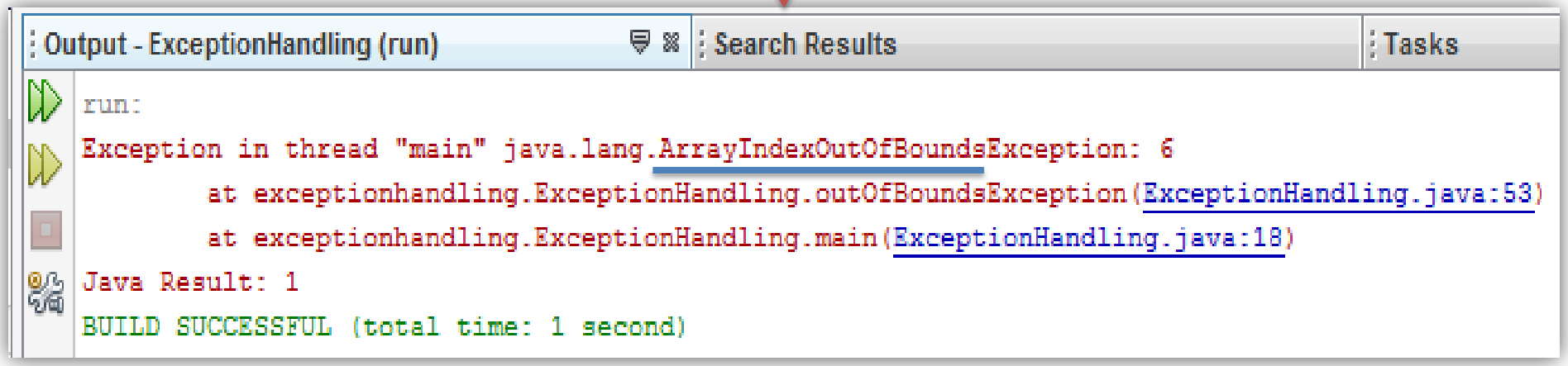
Division by zero

Invalid casting

Exceptions are Java run-time errors

```
// ...
private static void outOfBoundsException() {

    // this code generates
    // a null pointer exception
    int array[] = {0, 1, 2, 3, 4};

    System.out.println(array[6]);
}
// ...
```

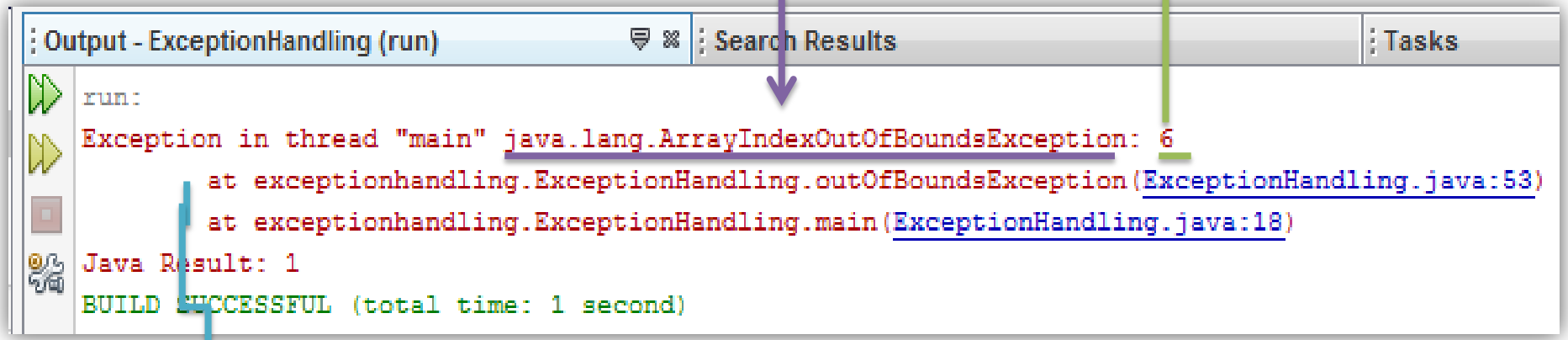
Out of bounds array access throws an Exception



Exceptions are Java run-time errors

Exception Name

Exception details



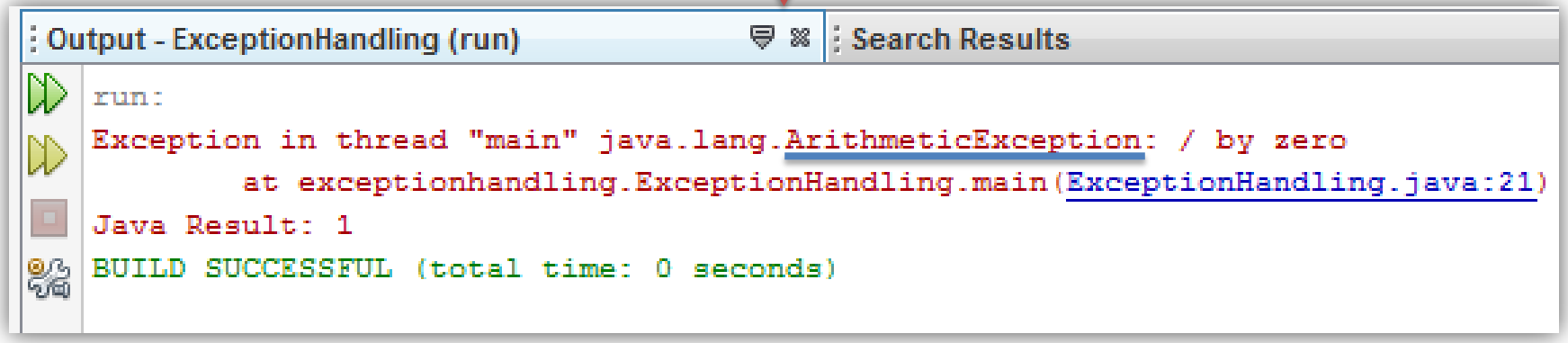
Stack Trace

A user-friendly snapshot of the threads and monitors in a JVM

Exceptions are Java run-time errors

```
// ...
public static void divisionByZeroException() {

    // this code generates
    // a division by zero exception
    int numberA = 5;
    int numberB = 0;

    int result = numberA / numberB;

    System.out.println(result);
}
// ...
```

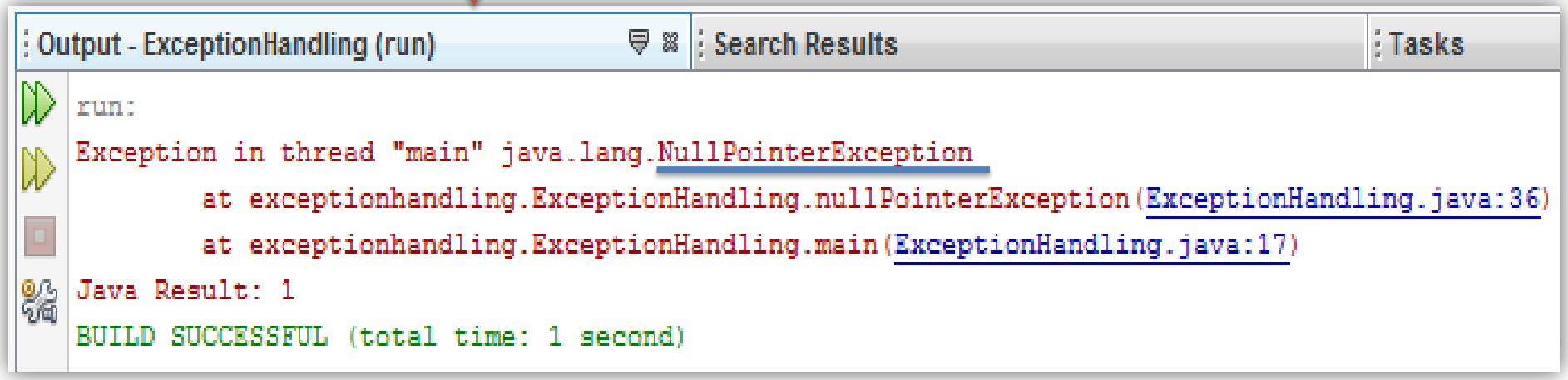
Division by zero throws an Exception (run time error)



Exceptions are Java run-time errors

Operate null objects throws an Exception

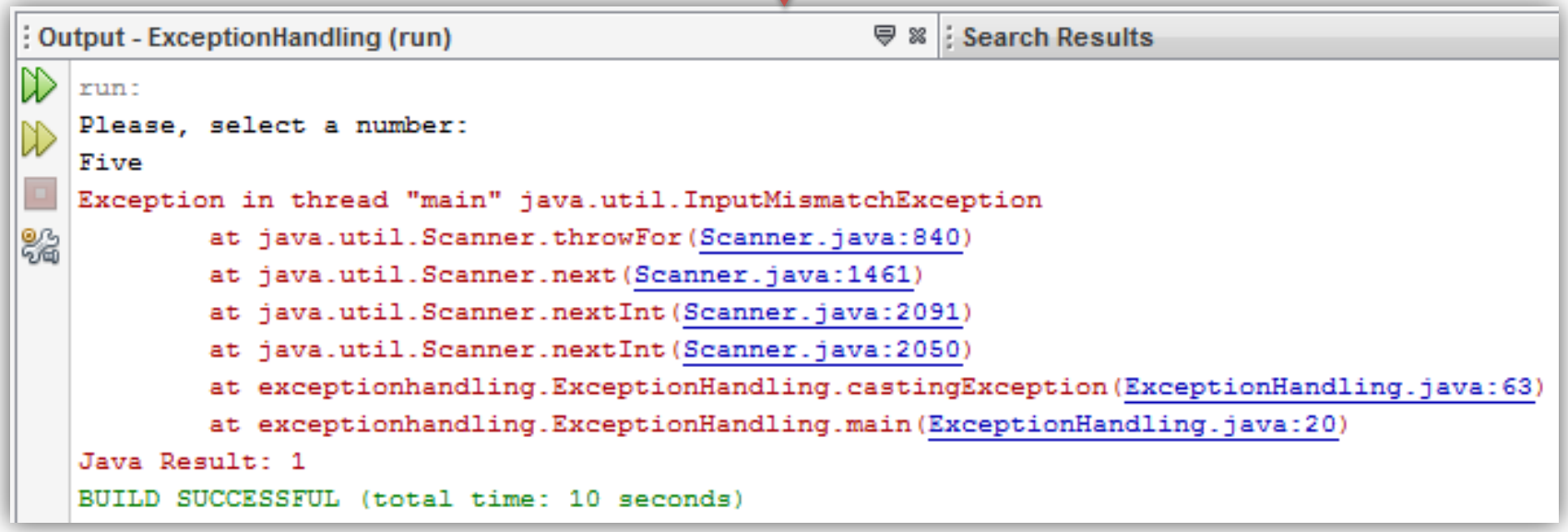
```
// ...
private static void nullPointerException() {

    // this code generates
    // a null pointer exception
    String name = null;

    name.length();

    System.out.println(name);
}
// ...
```



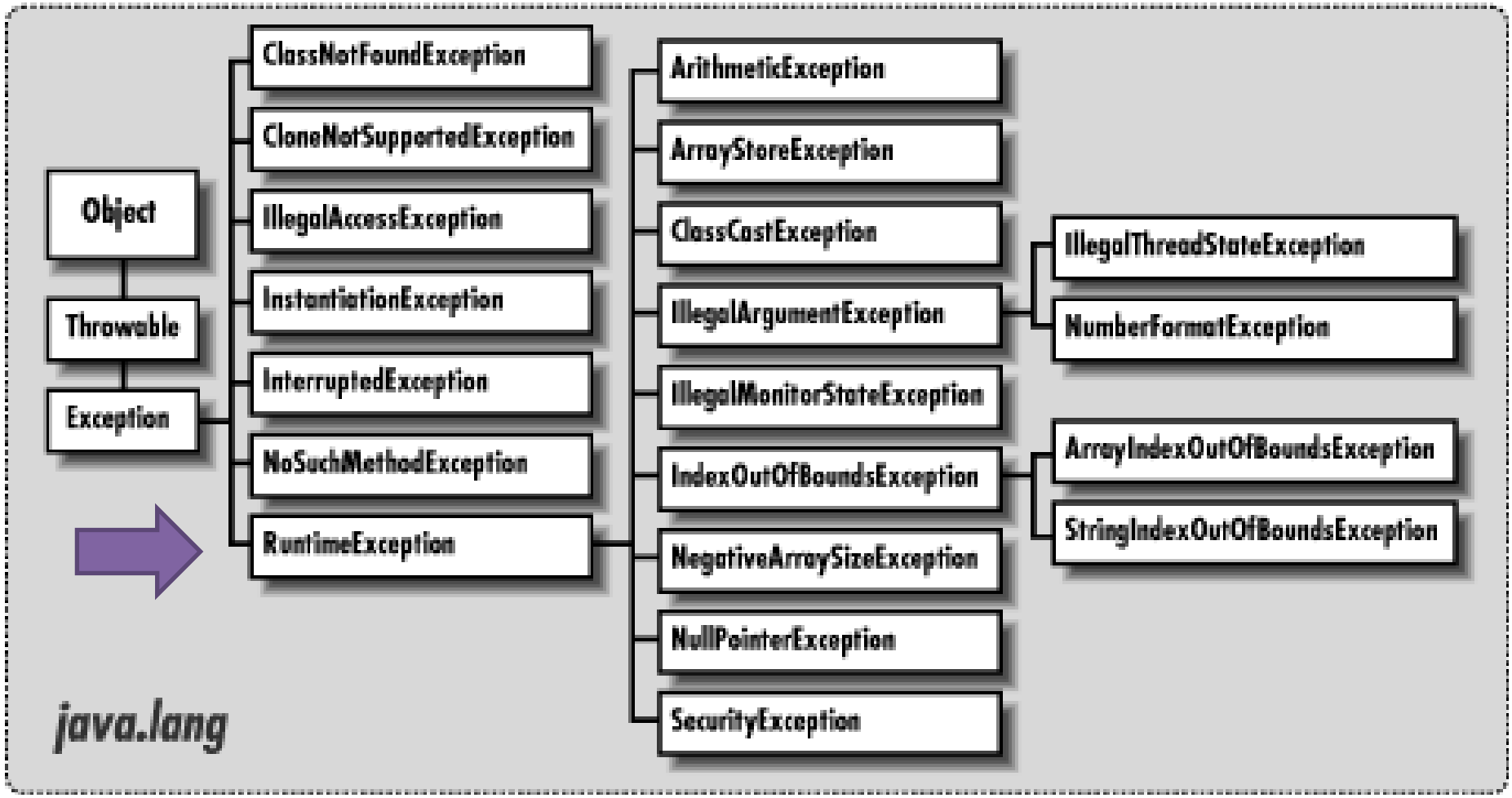
Exceptions are Java run-time errors

```
// ...
private static void castingException() {

    Scanner reader = new Scanner(System.in);

    System.out.println("Please, select a number: ");
    int userOption = reader.nextInt();
}
// ...
```

Invalid casting throws an Exception



The compiler checks for everything except RuntimeExceptions.



http://docstore.mik.ua/orelly/java/langref/ch09_04.htm

Exception Handling is “Plan B” to handle errors

Exception Handling allows us to be prepared for **unexpected run-time errors**

1.2 Try and Catch

Try and catch are used to handle exceptions

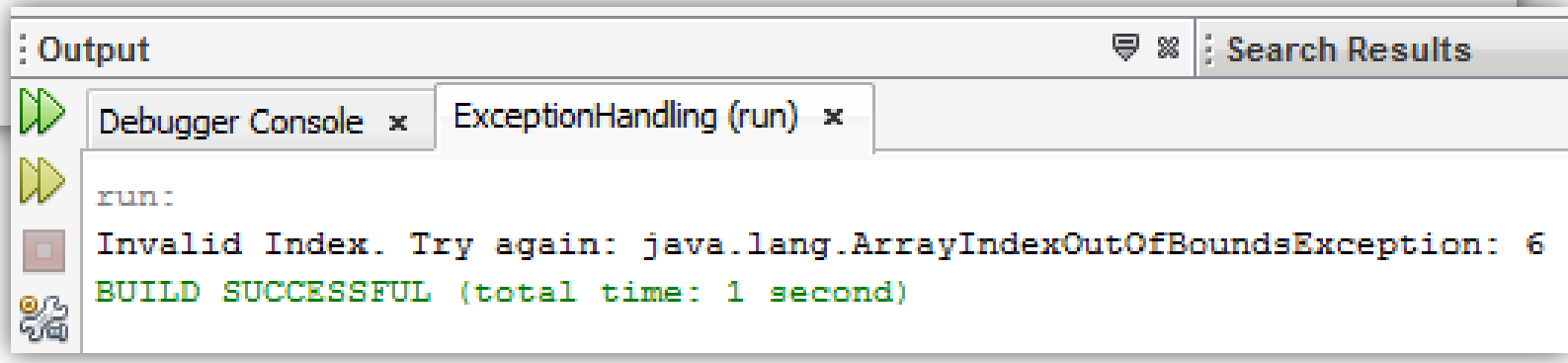
```
// ...
private static void TryAndCatch() {

    try {

        // To do some risky things and

    } catch (Exception exception) {
        // If something exceptional happens
        // execute the plan B (try to recover)

        // This code only runs if an Exception
        // is thrown
    }
}
// ...
```

Java *Try and Catch* structure example

Try and catch are used to handled exceptions

```
// ...
private static void outOfBoundsException() {

    try {

        int array[] = {0, 1, 2, 3, 4};
        System.out.println(array[6]);

    } catch (IndexOutOfBoundsException exception) {

        System.out.println("Invalid Index. Try again: "+exception);
    }
}
// ...
```

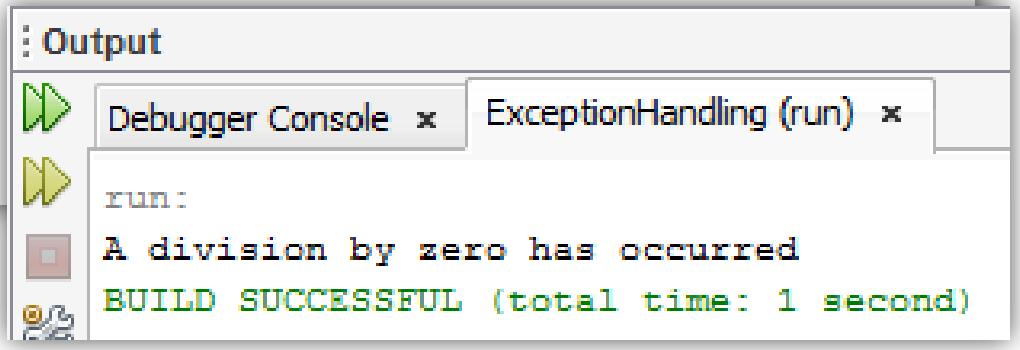
Exception class name



Try and catch are used to handled exceptions

```
// ...
public static void divisionByZeroException() {

    try {
        int numberA = 5;
        int numberB = 0;

        int result = numberA / numberB;

        System.out.println(result);

    } catch (ArithmeticException exception) {

        System.out.println("A division by zero has occurred");

    }
}
// ...
```

Exception class name



Try and catch are used to handled exceptions

```
// ...
private static void nullPointerException() {

    String name = null;

    try {
        name.length();
    } catch (NullPointerException exception) {
        name = "NAME";
    }

    System.out.println(name);
}
// ...
```

Exception class name

```
Output
Debugger Console × ExceptionHandling (run) ×
run:
NAME
BUILD SUCCESSFUL (total time: 1 second)
```

Try and catch are used to handled exceptions

```
// ...
private static void castingException() {

    Scanner reader = new Scanner(System.in);

    System.out.println("Please, select a number: ");

    try {

        int userOption = reader.nextInt();

    } catch (InputMismatchException exception) {
        System.out.println("Invalid number, select a number between "
                + Integer.MIN_VALUE + " and " + Integer.MAX_VALUE);
    }
}
// ...
```

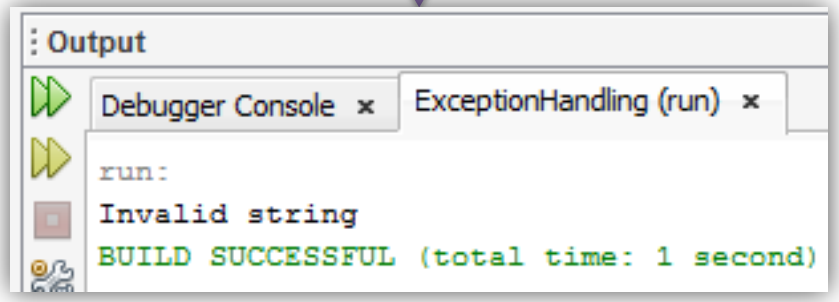
Exception class name

```
Output                                                    Search Results
Debugger Console × | ExceptionHandling (run) ×
run:
Please, select a number:
FIVE
Invalid number, select a number between -2147483648 and 2147483647
BUILD SUCCESSFUL (total time: 3 seconds)
```

Try and catch are used to handled exceptions

```
// ...
private static void castingException() {

    try {

        String numberString = "five";
        int number = Integer.parseInt(numberString);

    } catch (NumberFormatException ex) {

        System.out.println("Invalid string");

    }
}
// ...
```

Invalid casting throws an Exception



1.3 Finally

Finally block is used to do things no matter what happen

```
public static void main(String[] args) throws IOException {

    FileWriter writer = null;

    try {

        writer = new FileWriter("c:\\Windows");
        writer.write("My Test Text file");

    } catch (FileNotFoundException ex) {

        System.out.println("An Error has occurred trying to open the file: ");
        ex.printStackTrace();

    } catch (IOException ex) {

        ex.printStackTrace();

    } finally {

        System.out.println("File will be released");
        writer.close();

    }
}
```

This block will be executed whether an exception occurs or not

It is possible to handle to different Exceptions

```
public static void main(String[] args) throws IOException {

    FileWriter writer = null;

    try {

        writer = new FileWriter("c:\\Windows");
        writer.write("My Test Text file");

    } catch (FileNotFoundException ex) {

        System.out.println("An Error has occurred trying to open the file: ");
        ex.printStackTrace();

    } catch (IOException ex) {

        ex.printStackTrace();

    } finally {

        System.out.println("File will be released");
        writer.close();

    }
}
```

1

2

2. Exception Handling Flow

2.1 Who throws exceptions?

2.2 Exception handling flow

2.1 Who throws exceptions?

2.2 Exception handling flow

One method will catch what another method throws

nextInt

`public int nextInt()`

Scans the next token of the input as an `int`.

An invocation of this method of the form `nextInt()` behaves in exactly the same way as the invocation `nextInt(radix)`, where `radix` is the default radix of this scanner.

Returns:

the `int` scanned from the input

Throws:

`InputMismatchException` - if the next token does not match the *Integer* regular expression, or is out of range

`NoSuchElementException` - if input is exhausted

`IllegalStateException` - if this scanner is closed

Methods can throw more than one exception

Class Scanner – Method nexInt

One method will catch what another method throws

FileWriter

```
public FileWriter(File file)
            throws IOException
```

Constructs a FileWriter object given a File object.

Parameters:

`file` - a File object to write to.

Throws:

`IOException` - if the file exists but is a directory rather than a regular file, does not exist but cannot be created, or cannot be opened for any other reason

Class FileWritter – Constructor FileWriter(File)

One method will catch what another method throws

exists

```
public boolean exists()
```

Tests whether the file or directory denoted by this abstract pathname exists.

Returns:

`true` if and only if the file or directory denoted by this abstract pathname exists; `false` otherwise

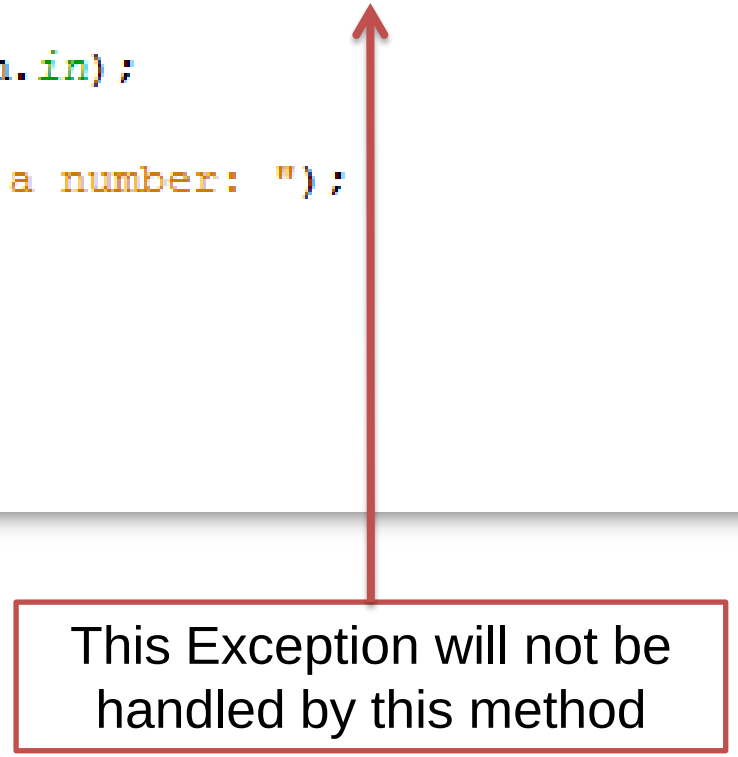
Throws:

`SecurityException` - If a security manager exists and its `SecurityManager.checkRead(java.lang.String)` method denies read access to the file or directory

Class File – method exists

Exceptions can be thrown by user defined methods

```
// ...
private static int castingException() throws InputMismatchException {

    Scanner reader = new Scanner(System.in);

    System.out.println("Please, select a number: ");

    return reader.nextInt();

}
// ...
```



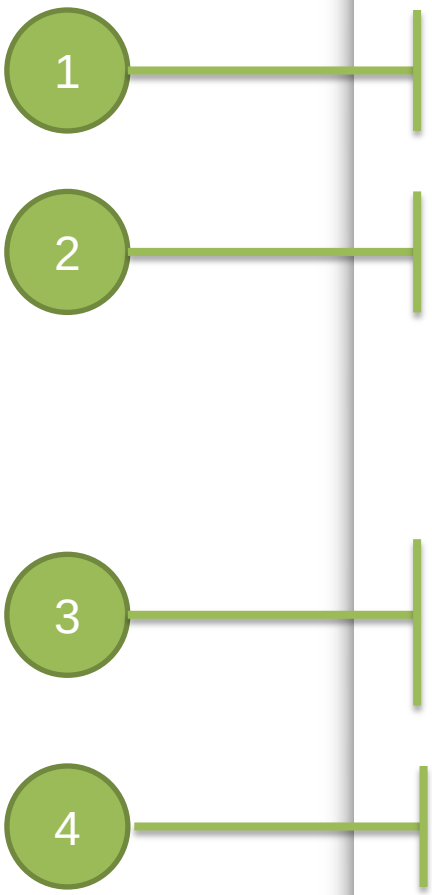
Exceptions can be thrown by user defined methods

```
 // ...
public static void main(String[] args) {

    try {

        castingException();

    } catch (InputMismatchException ex) {
        System.out.println("Invalid input.");
    }
}

private static int castingException() throws InputMismatchException {

    Scanner reader = new Scanner(System.in);

    System.out.println("Please, select a number: ");

    return reader.nextInt();

}
// ...
```

This Exception will be handled by this method

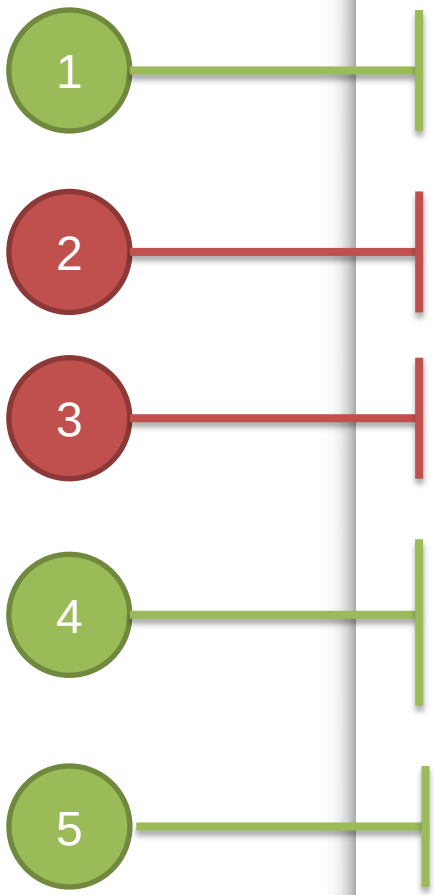
2.2 Exception handling flow

Normal Execution flow




```
// ...
private static void castingException() {

    System.out.println("Please, select a number: ");
    Scanner reader = new Scanner(System.in);
    int number = -1;

    try {

        number = reader.nextInt();

    } catch (InputMismatchException ex) {

        System.out.println("Error has ocurred");
        number = 0;

    } finally {

       executeTask(number);

    }

    System.out.println("Operation finished");

}
// ...
```

Error Execution flow



Exception thrown

```
// ...
private static void castingException() {

    System.out.println("Please, select a number: ");
    Scanner reader = new Scanner(System.in);
    int number = -1;

    try {

        number = reader.nextInt();

    } catch (InputMismatchException ex) {

        System.out.println("Error has ocurred");
        number = 0;

    } finally {

       executeTask(number);

    }

    System.out.println("Operation finished");


}
// ...
```

3. Exception Rules

You cannot have a catch or finally without a try

```
// ...
private static void TryAndCatch(
    System.out.println("Try and
    catch (Exception exception) {
    }
}
// ...
```

'catch' without 'try'

';' expected

(Alt-Enter shows hints)

```
// ...
private static void TryAndCatch() {
    System.out.println("Try and catch example");
    finally {
    }
}
// ...
```

You cannot put code between the try and the catch

```
// ...
private static void TryAndCatch() {
        'try' without 'catch' or 'finally'
        ----
        (Alt-Enter shows hints)
    Scan                              nner(System.in);

    try {

        int number = reader.nextInt();

    }
    System.out.println("We can not code here");
    catch (Exception exception) {

        exception.printStackTrace();

    }
}
```

A try must be followed by either a catch or a finally

```
// ...
private static void TryAndCatch() {
        Scan                                    nner(System.in);
    try {
        int number = reader.nextInt();

    }
}
// ...
```

'try' without 'catch' or 'finally'

(Alt-Enter shows hints)

References

[Deitel] H.M. Deitel and P.J. Deitel, *Java How to Program: Early Objects Version*, Prentice Hall, 2009.

[Sierra] K. Sierra and B. Bates, *Head First Java*, 2nd Edition, O'Reilly Media, 2005.