Java Exceptions

Table of contents

Introduced in 2026-03-19-exceptions[1] (Lecture, Week 4).

What is an exception?

An unexpected event that occurs during program execution, e.g.:

```
System.out.println(5 / 0); // ArithmeticException: / by zero (NOT Infinity, unlike
↪  floating-point division)
```

Common causes: invalid user input, loss of network connection, physical limitations (e.g. out of disk space), code errors, opening an unavailable file.

When an exception occurs, Java creates an **exception object** containing information about the failure (Effective Java, Item 75: include failure-capture information in detail messages).

[1] courses/csse2002/2026-03-19-exceptions.pdf

Exception hierarchy

```
Throwable
   Error
   Exception
        RuntimeException (unchecked)
        IOException (checked)
```

`RuntimeException` — unchecked

Caused by a programming error; the compiler does **not** force you to handle these:

- Improper use of an API → `IllegalArgumentException`
- Null pointer access → `NullPointerException`
- Out-of-bounds array access → `ArrayIndexOutOfBoundsException`
- Dividing by 0 → `ArithmeticException`

`IOException` — checked

Checked by the compiler at compile-time — the programmer is prompted (via `throws`) to handle these:

- Opening a file that doesn't exist → `FileNotFoundException`
- Reading past the end of a file → `EOFException`
- A connection attempt to a remote host fails → `ConnectException`

`try/catch`

Place code that might throw inside `try`; every `try` is followed by a `catch`:

```
int result;
try {
    result = 5 / 0;
} catch (ArithmeticException e) {
    System.out.println(e); // print out the error message
    result = Integer.MAX_VALUE; // fallback value
}
```

1. When an exception occurs, the rest of the `try` block is skipped.
2. The `catch` block catches it and its statements execute.
3. If nothing in `try` throws, `catch` is skipped entirely.

`throw` and `throws`

- **`throw`** explicitly throws a single exception:

```
public static void checkEntry(int age) {
    if (age < 18) {
        throw new IllegalArgumentException("Access denied");
    }
}
```

- **`throws`** (in a method declaration) declares the exception types a method might produce, so callers are prompted to handle them:

```
public static void findFile() throws FileNotFoundException {
    File newFile = new File("file.txt");
    FileInputStream stream = new FileInputStream(newFile);
}
```

`finally`

Runs regardless of whether an exception was thrown; optional.

```
try {
    // code
} catch (ExceptionType1 e1) {
    // catch block
} finally {
    // always executes
}
```

Worked example — what does this print?

```
try {
    System.out.println("Good Morning");
    throw new FileNotFoundException();
    System.out.println("the earth says"); // never reached
} catch (Exception e) {
    System.out.println("hello");
} finally {
    System.out.println("world");
}
```

Prints `Good Morning`, `hello`, `world` — the `throw` immediately skips the rest of `try` (so `"the earth says"` never prints), `catch` runs since `FileNotFoundException` is an `Exception`, and `finally` always runs last.

Custom exceptions

Extend `Exception` (or a subclass) to define your own:

```
public class ItemNotFoundException extends Exception {
    public ItemNotFoundException() {
        super("Item not found");
    }
    public ItemNotFoundException(String message) {
        super(message);
    }
}

public class Store {
    private List<String> items;

    public Store() {
        items = new ArrayList<>();
        items.add("apple");
        items.add("banana");
        items.add("orange");
    }

    public String findItem(String itemName) throws ItemNotFoundException {
        for (String item : items) {
            if (item.equals(itemName)) {
                return item;
            }
        }
        throw new ItemNotFoundException("Item '" + itemName + "' not found in the
        ↪  list.");
    }
}
```