Exception Handling

Handling Errors During the Program Execution

SoftUni Team

Technical Trainers

Software University

https://softuni.bg

Table of Contents

Have a Question?

Software University

sli.do

#java-advanced

What Are Exceptions?

What Are Exceptions?

- Simplify code construction and maintenance
- Allow the problematic situations to be processed at multiple levels
- Exception objects have detailed information about the error

There are two ways to write error-free programs; only the third one works. (Alan J. Perlis)

The Throwable Class

- Exceptions in Java are **objects**
- The `Throwable` class is a base for all exceptions in JVM
  - Contains information for the cause of the error
  - `Message` – a text description of the exception
    - `StackTrace` – the snapshot of the stack at the moment of exception throwing

Types of Exceptions

- Java exceptions inherit from `Throwable`
- Below `Throwable` are:
  - `Error` - not expected to be caught under normal circumstances from the program
    - Example - `StackOverflowError`
  - `Exception`
    - Used for exceptional conditions that user programs should catch
    - User-defined exceptions

Exceptions

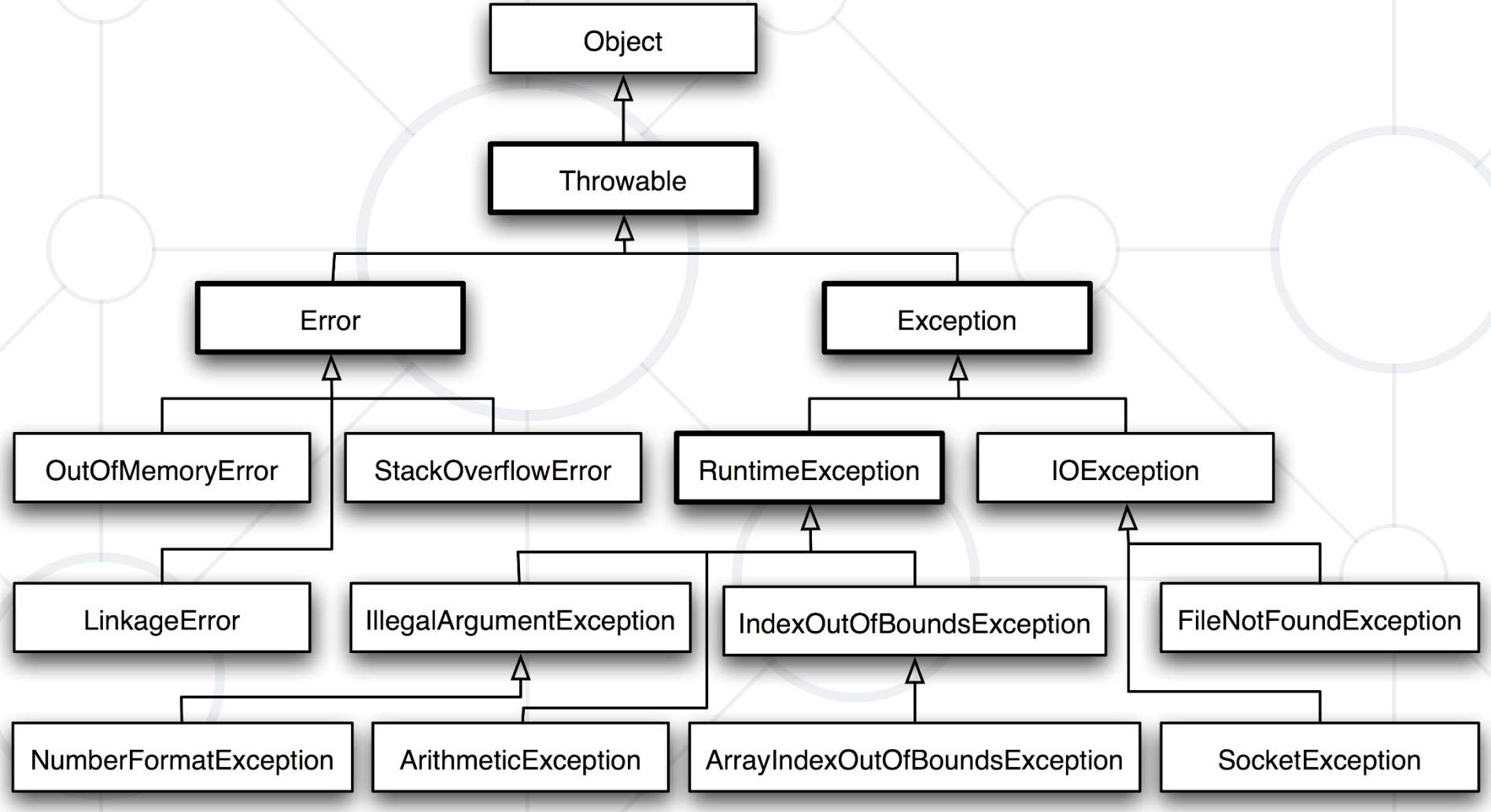
- **Exceptions** are two types:
  - **Checked** - an exception that is checked (notified) by the compiler at compilation-time
    - Also called as **Compile Time** exceptions

```
public static void main(String args[]) {
  File file = new File("E://file.txt");
  FileReader fr = new FileReader(file);
}
```

FileNotFoundException

  - **Unchecked** - an exception that occurs at the time of execution
    - Also called as **Runtime Exceptions**

Exception Hierarchy



Handling Exceptions

Handling Exceptions

- In Java exceptions can be handled by the `try-catch` construction

```
try {
  // Do some work that can raise an exception
} catch (SomeException) {
  // Handle the caught exception
}
```

- `catch` blocks can be used multiple times to process different exception types

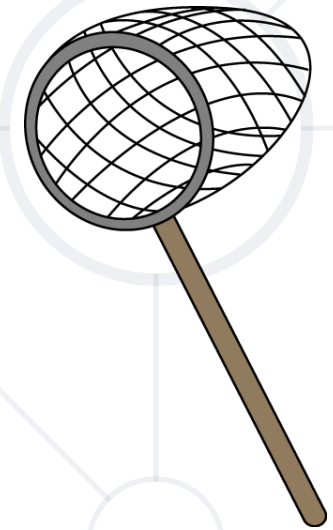
Multiple Catch Blocks – Example

```
String s = sc.nextLine();
try {
    Integer.parseInt(s);
    System.out.printf(
      "You entered a valid integer number %s.", s);
} catch (NumberFormatException ex) {
    System.out.println("Invalid integer number!");
}
```

Handling Exceptions

- When catching an exception of a particular class, all its inheritors (child exceptions) are caught too, e.g.

```
try {
  // Do some work that can cause an exception
} catch (IndexOutOfBoundsException ae) {
  // Handle the caught arithmetic exception
}
```



- Handles **IndexOutOfBoundsException** and its descendants **ArrayIndexOutOfBoundsException** and **StringIndexOutOfBoundsException**

Find the Mistake!

```
String str = sc.nextLine();

try {
    Integer.parseInt(str);
} catch (Exception ex) {
    System.out.println("Cannot parse the number!");
} catch (NumberFormatException ex) {
    System.out.println("Invalid integer number!");
}
```

Should be last

Unreachable code

Handling All Exceptions

- Unmanaged code can throw other exceptions
- For handling all exceptions (even unmanaged) use the construction:

```
try {
  // Do some work that can raise any exception
} catch (Exception ex) {
  // Handle the caught exception
}
```

The Try-finally Statement

- The statement:

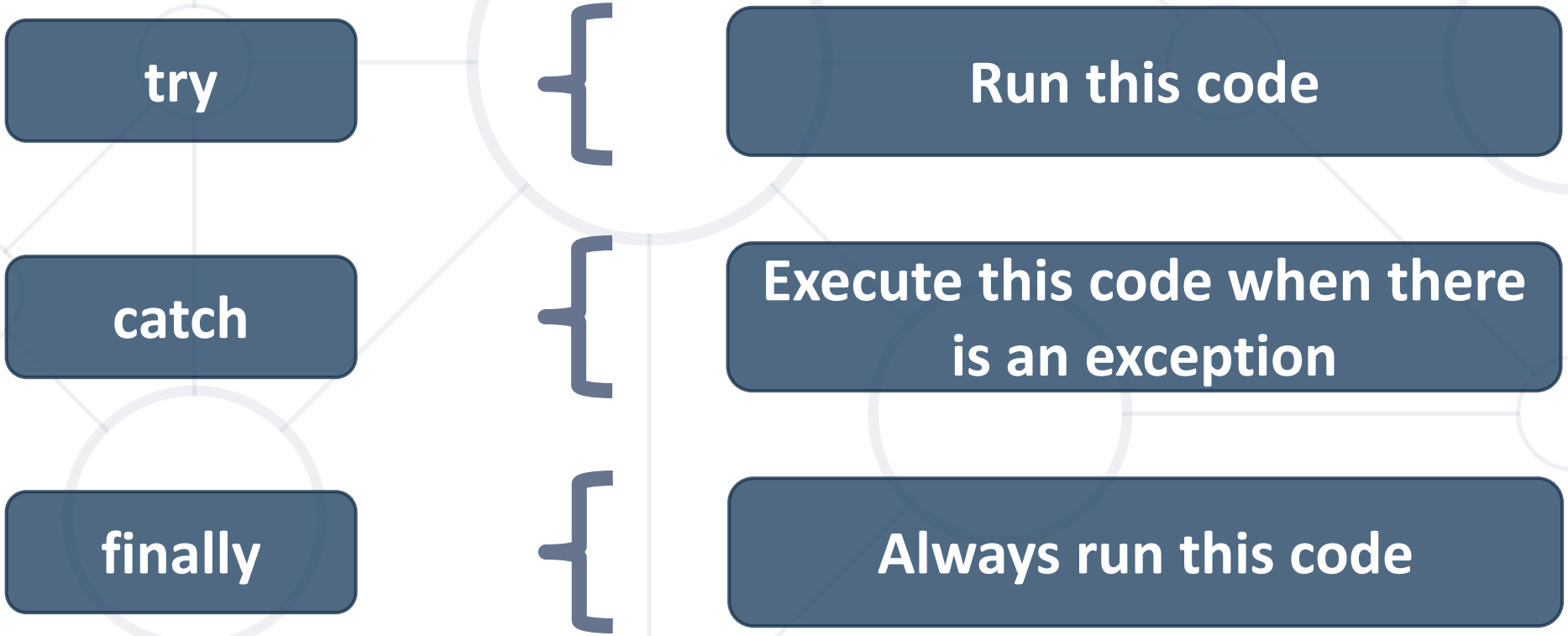
```
try {
    // Do some work that can cause an exception
} finally {
    // This block will always execute
}
```

- Ensures execution of a given block in all cases
  - When exception is raised or not in the `try` block
- Used for execution of cleaning-up code, e.g. releasing resources

Try-finally – Example

```
static void testTryFinally() {
  System.out.println("Code executed before try-finally.");
  try {
     String str = sc.nextLine();
     Integer.parseInt(str);
     System.out.println("Parsing was successful.");
     return; // Exit from the current method
  } catch (NumberFormatException ex) {
     System.out.println("Parsing failed!");
  } finally {
     System.out.println("This cleanup code is always executed.");
  }
  System.out.println("This code is after the try-finally block.");
}
```

How Do Exceptions Work?



Handling Exceptions

Throwing Exceptions

Throwing Exceptions

- Exceptions are thrown (raised) by the `throw` keyword
- Used to notify the calling code in case of an error or unusual situation
- When an exception is thrown:
  - The program execution stops
  - The exception travels over the stack
    - Until a matching `catch` block is reached to handle it
- Unhandled exceptions display an error message

Using Throw Keyword

- Throwing an exception with an error message:

```
throw new IllegalArgumentException("Invalid amount!");
```

- Exceptions can accept message and cause:

```
try {
  …
} catch (SQLException sqlEx) {
  throw new IllegalStateException("Cannot save invoice.", sqlEx);
}
```

- **Note**: if the original exception is not passed, the initial cause of the exception is lost

Re-Throwing Exceptions

- Caught exceptions can be re-thrown again:

```
try {
    Integer.parseInt(str);
} catch (NumberFormatException ex) {
    System.out.println("Parse failed!");
    throw ex; // Re-throw the caught exception
}
```

Throwing Exceptions – Example

```
public static double sqrt(double value) {
  if (value < 0)
    throw new IllegalArgumentException(
                  "Sqrt for negative numbers is undefined!");
  return Math.sqrt(value);
}
public static void main(String[] args) {
  try {
     sqrt(-1);
  } catch (IllegalArgumentException ex) {
     System.err.println("Error: " + ex.getMessage());
     ex.printStackTrace();
  }
}
```

Throwing Exceptions

Best Practices

Using Catch Block

- `Catch` blocks should:
  - Begin with the exceptions lowest in the hierarchy
  - Continue with the more general exceptions
  - Otherwise a compilation error will occur
- Each `catch` block should handle only these exceptions which it expects
  - If a method is not competent to handle an exception, it should leave it unhandled
  - Handling all exceptions disregarding their type is a popular **bad practice** (anti-pattern)!

Choosing the Exception Type (1)

- When an application attempts to use `null` in a case where an object is required – `NullPointerException`
- An array has been accessed with an illegal index – `ArrayIndexOutOfBoundsException`
- An index is either negative or greater than the size of the string – `StringIndexOutOfBoundsException`
- Attempts to convert a inappropriate string to one of the numeric types - `NumberFormatException`

Choosing the Exception Type (2)

- When an exceptional arithmetic condition has occurred – `ArithmeticException`
- Attempts to cast an object to a subclass of which it is not an instance – `ClassCastException`
- A method has been passed an illegal or inappropriate argument - `IllegalArgumentException`

Exceptions – Best Practices (1)

- When raising an exception, always pass to the constructor a **good explanation message**
- When throwing an exception always pass a good description of the problem
  - The exception message should explain what causes the problem and how to solve it
    - Good: "Size should be integer in range [1…15]"
    - Good: "Invalid state. First call Initialize()"
    - Bad: "Unexpected error"
    - Bad: "Invalid argument"

Exceptions – Best Practices (2)

- Exceptions can decrease the application performance
  - Throw exceptions only in situations which are really exceptional and should be handled
  - Do not throw exceptions in the normal program control flow
  - JVM could throw exceptions at any time with no way to predict them
    - E.g. `StackOverflowError`

Custom Exceptions

Creating Custom Exceptions

- Custom exceptions inherit an exception class (commonly – **Exception**)

```
public class TankException extends Exception {
   public TankException(String msg) {
      super(msg);
   }
}
```

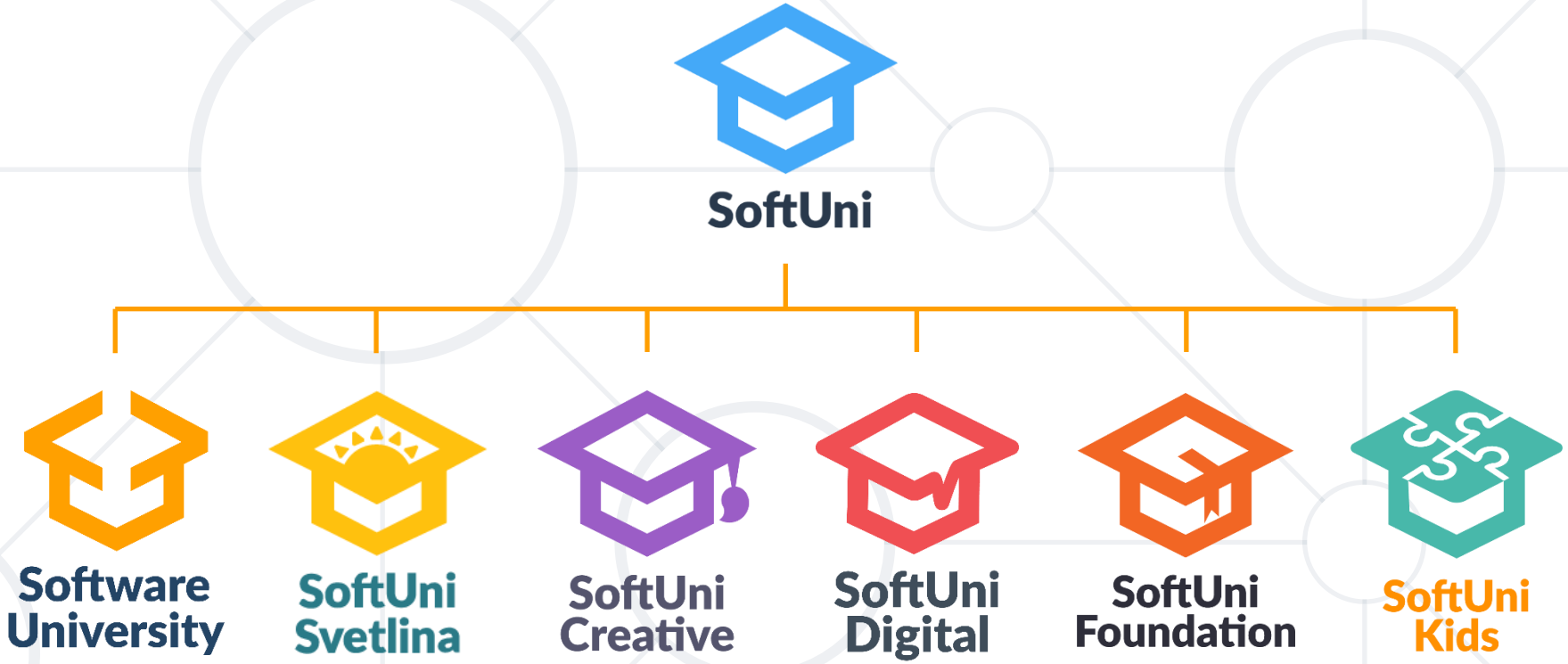
- Thrown just like any other exception

```
throw new TankException("Not enough fuel to travel");
```

Summary

- Exceptions provide a **flexible** error handling mechanism
- Unhandled exceptions cause error messages
- `try-finally` ensures a given code block is always executed
  - Even when an exception is thrown

Questions?



SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

INDEAVR
Serving the high achievers

SmartIT

XS SOFTWARE

zühlke
empowering ideas

INFRAGISTICS®

Coca-Cola HBC Bulgaria

DRAFT KINGS

Postbank
Решения за твоето утре

SOFTWARE GROUP

Educational Partners

Software University

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

Software University

FORUM

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg
- © Software University – https://softuni.bg