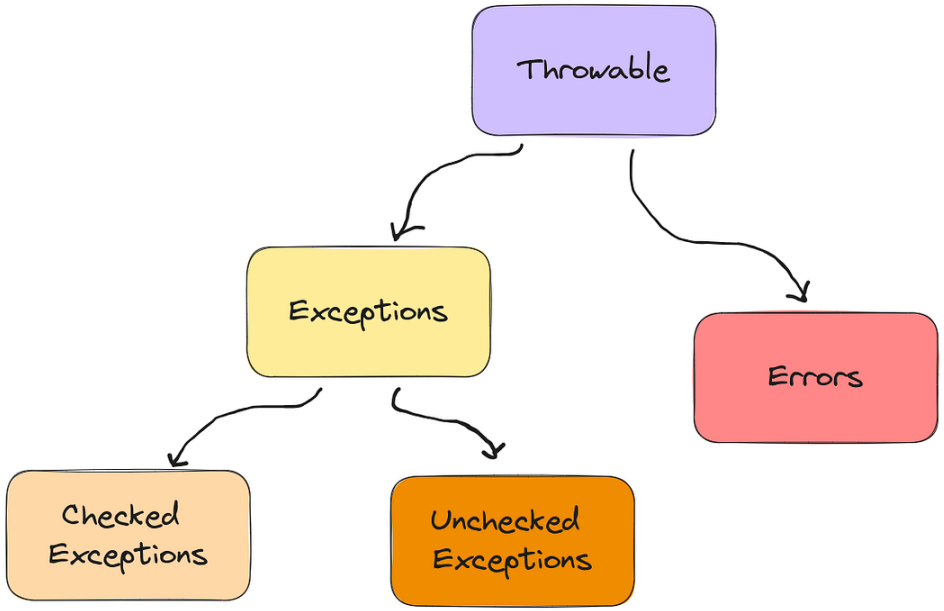
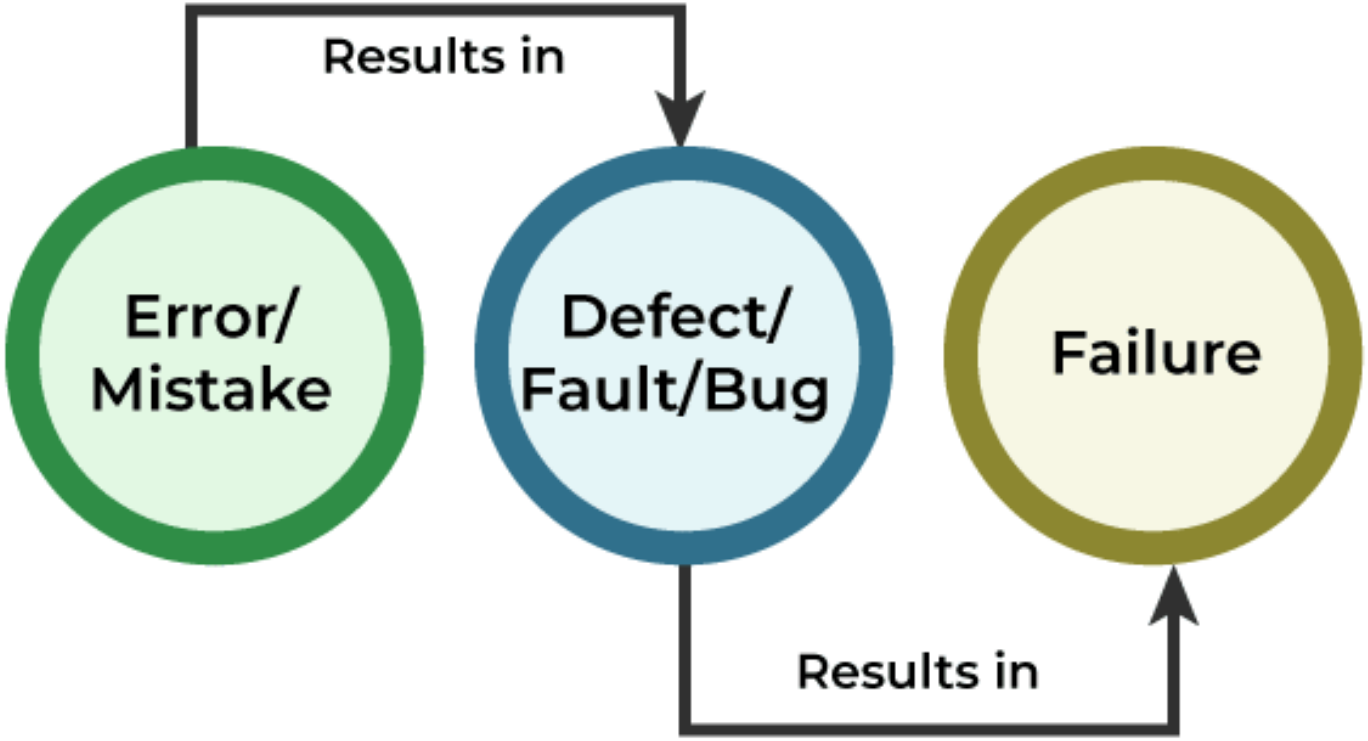


Exception Handling

Nimesha Hewawasam
Lecturer
nimesha.h@nsbm.ac.lk

Terms in Software Engineering



Error vs Exception

Error

Should not be caught, means an error in the program code

User can’t handle

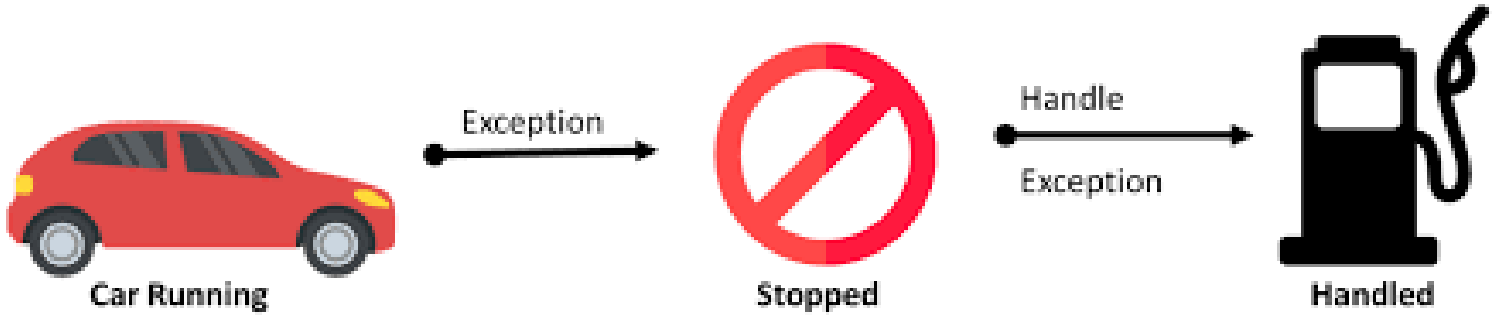
Exception

Encouraged to be caught, provide useful data

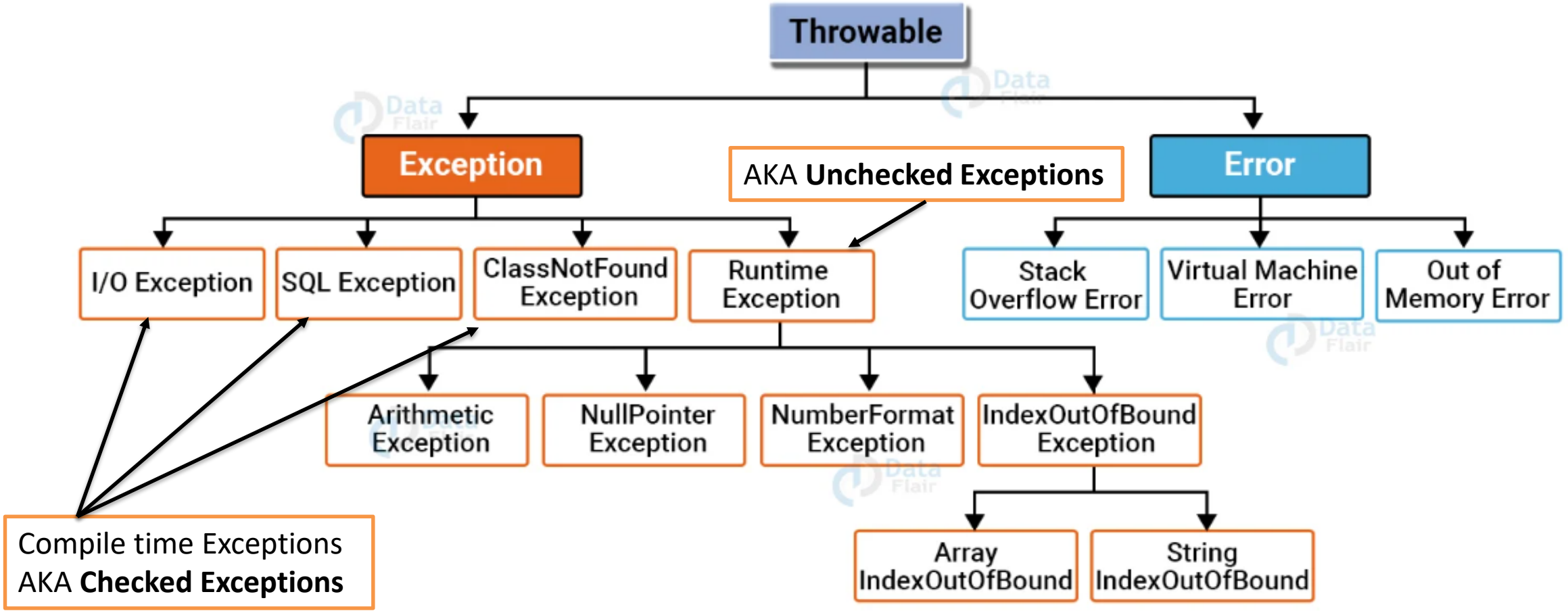
User can handle

Introduction to Exceptions

- An ***exception*** is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.
- The Java uses ***exceptions*** to handle errors and other exceptional events.

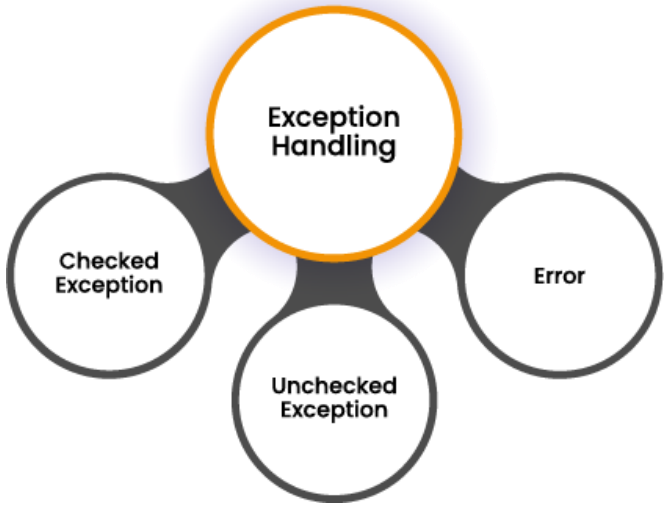


Hierarchy of Java Exceptions



Types of Exceptions

- There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception.
- According to Oracle, there are three types of exceptions:
  - Checked Exception
  - Unchecked Exception
  - Error



Checked Exception vs Unchecked Exception vs Error

1) Checked Exception

- The classes which directly **inherit Throwable class** except *RuntimeException and Error* are known as checked exceptions e.g. IOException, SQLException etc. **Checked exceptions are checked at compile-time**.

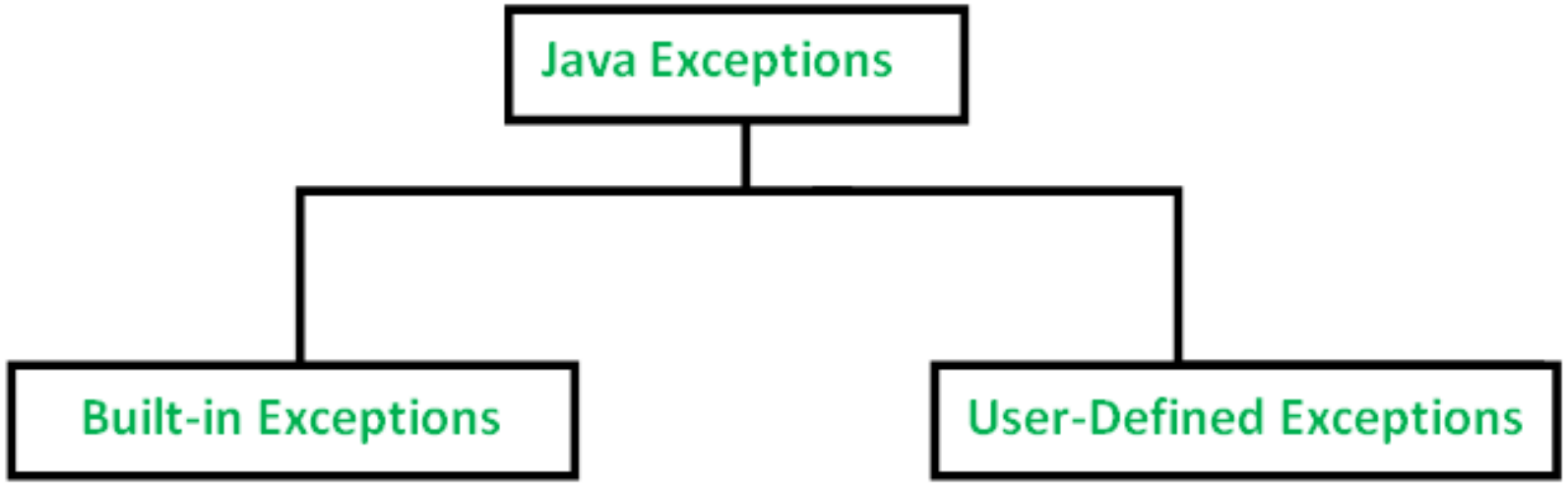
2) Unchecked Exception

- The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, **but they are checked at runtime**.

3) Error

- Error is **irrecoverable** e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Types of Exception



The Catch or Specify Requirement

This means that code that might throw certain exceptions must be enclosed by either of the following:

1. A **try statement** that catches the exception.
2. A method that specifies that it can **throw** the exception. (The method must provide a **throws** clause that lists the exception.)

Catching Exception

- A method catches an exception using a combination of the **try** and **catch** keywords.
- A try/catch block is placed around the code that might generate an exception.
- Code within a try/catch block is referred to as protected code

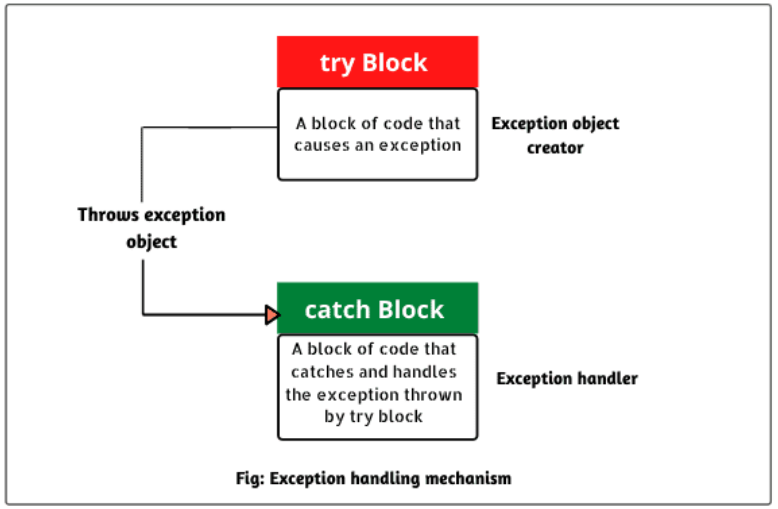
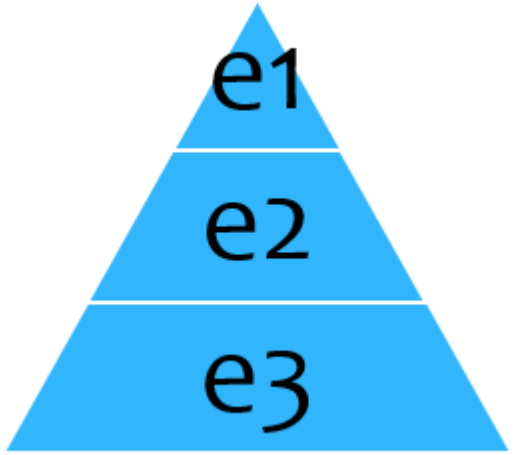


Fig: Exception handling mechanism

Multiple Catch Blocks:

- A try block can be followed by multiple catch blocks.

```
try{
        // protected code
}
catch (ExceptionName1  e1){
}
catch (ExceptionName2   e2){
}
catch (ExceptionName3   e3){
}
```



Finally Clause:

```
finally {
        // this block always executes
}
```

- The finally keyword is used to create a block of code that follows a try block.
- A finally block of code always executes, whether or not an exception has occurred.

Common Scenarios of Java Exception

- A scenario where **ArithmeticException** occurs

```java
public class Example {
    public static void main(String[] args) {
        int a = 10;
        int b = 0;
        int c = a / b;  // ArithmeticException: / by zero
        System.out.println(c);
    }
}
```

```java
public class Example {
    public static void main(String[] args) {
        int a = 10, b = 0;
        try {
            int c = a / b;
            System.out.println(c);
        } catch (ArithmeticException e) {
            System.out.println("Error: Cannot divide by zero.");
        }
    }
}
```

- A scenario where **NullPointerException** occurs

```java
public class Example {
    public static void main(String[] args) {
        String s = null;
        System.out.println(s.length()); // NullPointerException
    }
}
```

```java
public class Example {
    public static void main(String[] args) {
        String s = null;
        try {
            System.out.println(s.length());
        } catch (NullPointerException e) {
            System.out.println("Error: String is null.");
        }
    }
}
```

- A scenario where **NumberFormatException** occurs

```java
public class Example {
    public static void main(String[] args) {
        String s = "abc";
        int num = Integer.parseInt(s); // NumberFormatException
        System.out.println(num);
    }
}
```

```java
public class Example {
    public static void main(String[] args) {
        String s = "abc";
        try {
            int num = Integer.parseInt(s);
            System.out.println(num);
        } catch (NumberFormatException e) {
            System.out.println("Error: Invalid number format.");
        }
    }
}
```

- A scenario where **ArrayIndexOutOfBoundsException** occurs

```java
public class Example {
    public static void main(String[] args) {
        int[] arr = new int[3];
        arr[5] = 50; // ArrayIndexOutOfBoundsException
        System.out.println(arr[5]);
    }
}
```

```java
public class Example {
    public static void main(String[] args) {
        int[] arr = new int[3];
        try {
            arr[5] = 50;
            System.out.println(arr[5]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Error: Array index out of range.");
        }
    }
}
```

Built in Exceptions

- **ArithmeticException :** It is thrown when an exceptional condition has occurred in an arithmetic operation.
- **ArrayIndexOutOfBoundException :** It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.
- **ClassNotFoundException :** This Exception is raised when we try to access a class whose definition is not found
- **FileNotFoundException :** This Exception is raised when a file is not accessible or does not open.
- **IOException :** It is thrown when an input-output operation failed or interrupted
- **InterruptedException :** It is thrown when a thread is waiting , sleeping , or doing some processing , and it is interrupted.
- **NoSuchFieldException :** It is thrown when a class does not contain the field (or variable) specified
- **NoSuchMethodException :** It is thrown when accessing a method which is not found.
- **NullPointerException :** This exception is raised when referring to the members of a null object. Null represents nothing
- **NumberFormatException :** This exception is raised when a method could not convert a string into a numeric format.
- **RuntimeException :** This represents any exception which occurs during runtime.
- **StringIndexOutOfBoundsException :** It is thrown by String class methods to indicate that an index is either negative than the size of the string

Throws / throw:

```
public  void  doAddition()  throws  MyException {

      throw new MyException();
}
```

- If a method *does not handle a checked exception*, the method must declare it using the **throws** keyword.
- The throws keyword appears at the end of a method's signature.
- You *can throw an exception by using* the **throw** keyword.

Declaring your own Exception Class

```
Public  class  MyException  extends  Exception {
}
```

- You can create your own exceptions in Java.
- Note:
  - **All exceptions must be a child of Throwable.**
  - If you want to write a checked exception that is automatically enforced by the Handle or Declare Rule, you need to **extend the Exception** class.
  - If you want to write a runtime exception, you need to extend the RuntimeException class.

Thank you!