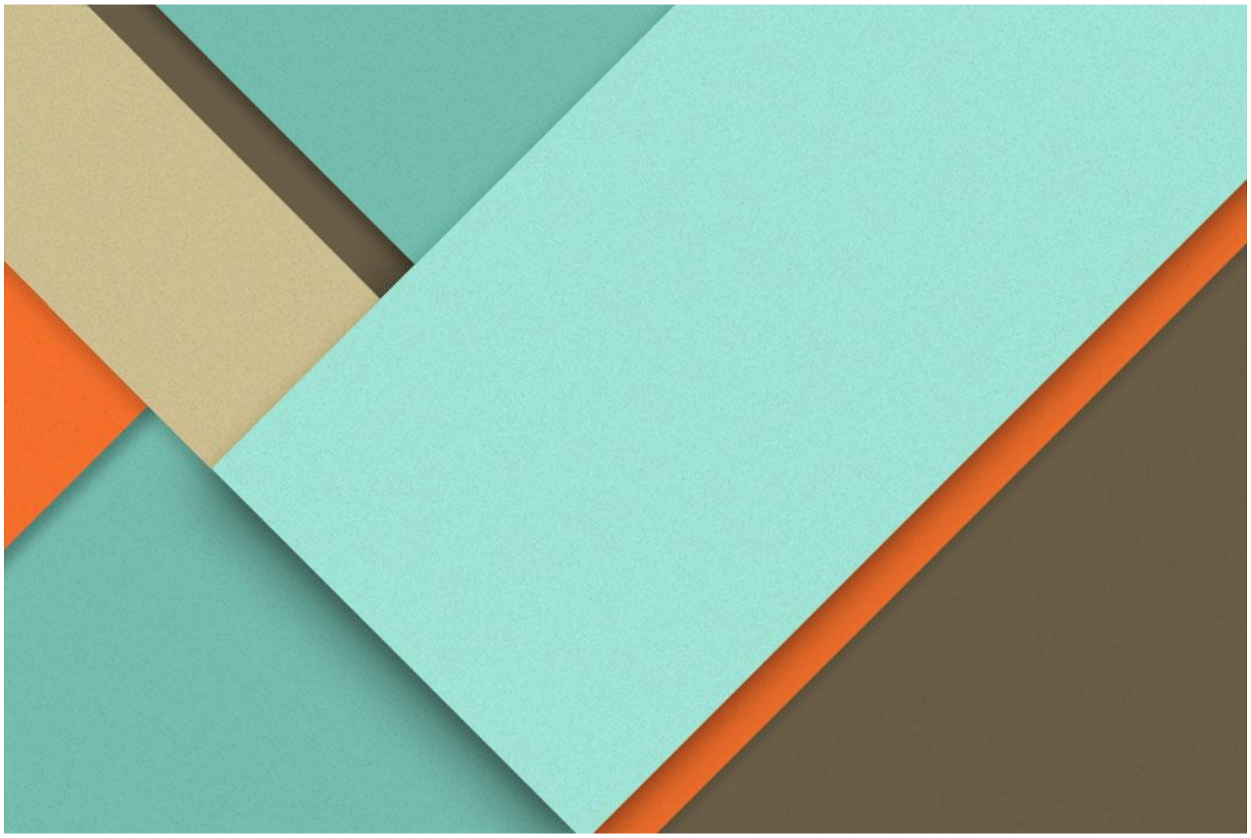


Java Exceptions

09.05.2022

—

B27
Group 4
Viacheslav

What's for?

To avoid program (app) crush, we can predict some problems and let the app keep running or close it correctly.

What should we do?

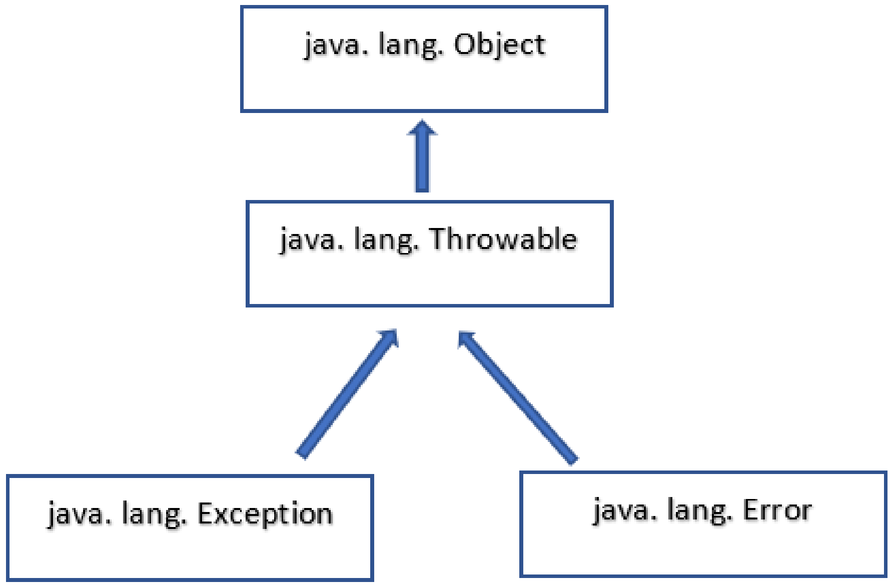
We should divide Errors and Exceptions because we can do nothing with errors but can handle exceptions.

Why?

Errors and Exceptions are subclasses of the Throwable Java class.

The **Error class** represents critical conditions that can not be caught and handled by the program's code.

Exception class represents concerning conditions raised by the application itself; these can be caught and handled within the code to ensure that the application continues to run smoothly.



Errors class:

AnnotationFormatError, AssertionError, AWTError, CoderMalfunctionError, FactoryConfigurationError, FactoryConfigurationError, IOError, LinkageError, ServiceConfigurationError, ThreadDeath, TransformerFactoryConfigurationError,

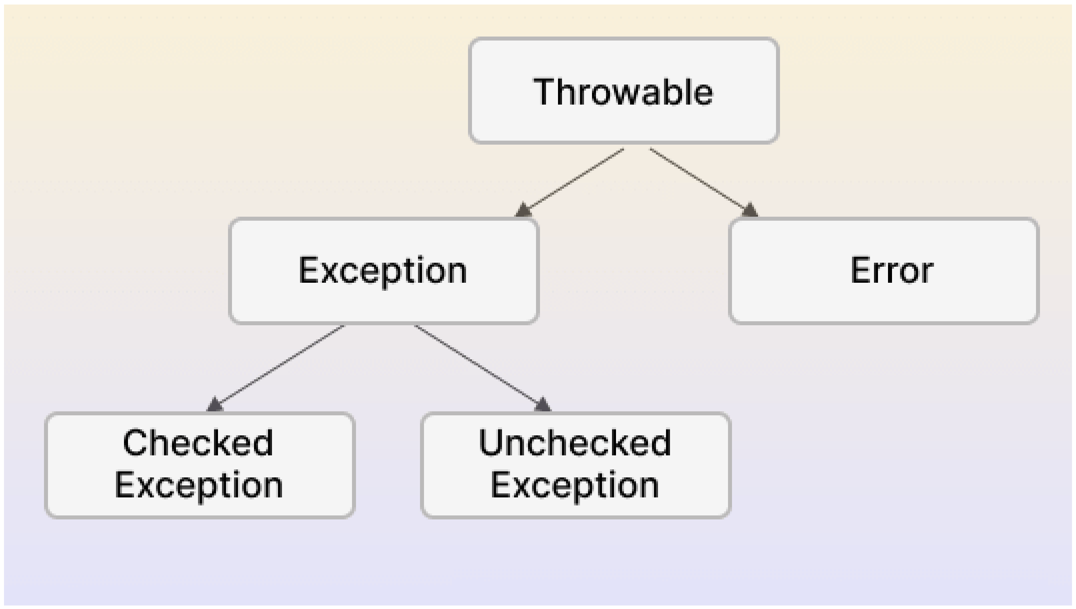
VirtualMachineError → InternalError, OutOfMemoryError, StackOverflowError, UnknownError.

Oracle docs link: https://docs.oracle.com/javase/7/docs/api/java/lang/Error.html

Exception class:

Exception class has checked and unchecked exceptions.

Oracle docs link: https://docs.oracle.com/javase/7/docs/api/java/lang/Exception.html



Checked Exceptions:

These are the exceptions that are checked at compile time, before running programm.

Unchecked Exceptions:

These are the exceptions that are not checked at compile time, the program starts running even if the code is not correct.

Unchecked exceptions located in **java.lang.RuntimeException**, a subclass of class Exceptions.

Oracle docs link:

https://docs.oracle.com/javase/7/docs/api/java/lang/RuntimeException.html

☝ We can handle Checked Exceptions and Unchecked Exceptions.

Checked Exceptions handling:

throws keyword that is used to declare the exception in method's signature.

‼ Can be used with checked exceptions only.

Syntax:

```
methodName() throws Exception1,Exception2,Exception3...{
//code
}
```

Example:

```
public static void main(String[] args)throws InterruptedException
{
   Thread.sleep(10000);
   System.out.println("Hello");
}
```

InterruptedException here will be ignored, and the program will run.

Unchecked and Checked Exceptions handling:

try → catch & finally

The **try** statement allows you to define a block of code to be tested while it is being executed.

The **catch** statement allows you to define a block of code to be executed if an exception occurs in the **try** block.

Syntax:

```
try {
  // Block of code to try
}
catch(Exception e) {
  // Block of code to handle errors
}
```

Example:

```
public static void main(String[ ] args) {
  try {
    int[] myNumbers = {1, 2, 3};
    System.out.println(myNumbers[10]);
  } catch (ArrayIndexOutOfBoundsException e) {
   e.getMessage();
   e.printStackTrace();
   e.getCause();
  }
}
```

Output is

```
java.lang.ArrayIndexOutOfBoundsException: Index 10 out of bounds for length 3
	at TryCatch.main(TryCatch.java:5)
Something went wrong.
```

The program was not interrupted due to the wrong index 10.
ArrayIndexOutOfBoundsException was caught.

In block **catch** instance of exception class was declared, and Exception class methods were called:

```
e.getMessage();
e.printStackTrace();
e.getCause();
```

These methods give us the required information about the caught exceptions.
Here is a link to check other methods of Exception class:
https://docs.oracle.com/javase/7/docs/api/java/lang/Exception.html

finally keyword defines a block of code we use along with the **try** keyword. It defines code that's always run after the try and any catch block before the method is completed. The finally block executes regardless of whether an exception is thrown or caught. It's an optional keyword that can be used to close the program correctly.

```
try {
  int[] myNumbers = {1, 2, 3};
  System.out.println(myNumbers[10]);
} catch (Exception e) {
  System.out.println("Something went wrong.");
} finally {
  System.out.println("The 'try catch' is finished.");
}
```

Unchecked and Checked Exceptions handling:

throw keyword

The **throw** keyword is used to create a custom error.

The **throw** defines **new** instance of the exception class, and we can add a String message. See example:

```
public class Main {
  static void checkAge(int age) {
    if (age < 18) {
      throw new ArithmeticException("Access denied - You must be
at least 18 years old.");
    }
    else {
      System.out.println("Access granted - You are old enough!");
    }
  }
  public static void main(String[] args) {
    checkAge(15);
  }
```

```
    }
```

```
Exception in thread "main" java.lang.ArithmeticException:
Access denied - You must be at least 18 years old.
```