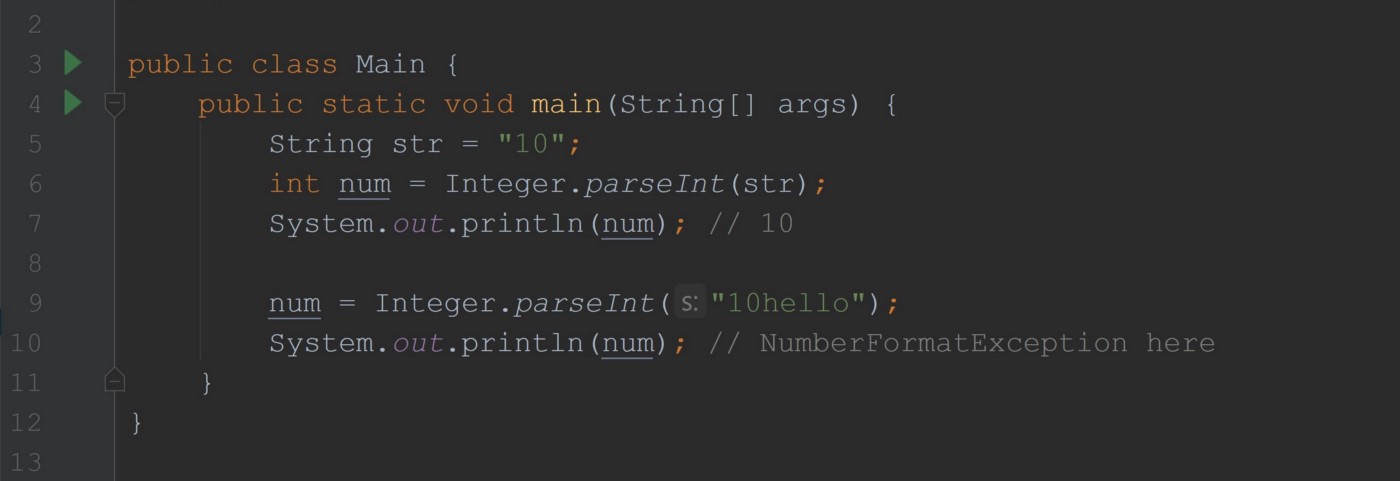
Exception in Java:

1. What is an exception?
2. Exception types.
3. How to handle exceptions?
4. How to throw an exception?
5. Custom exceptions.

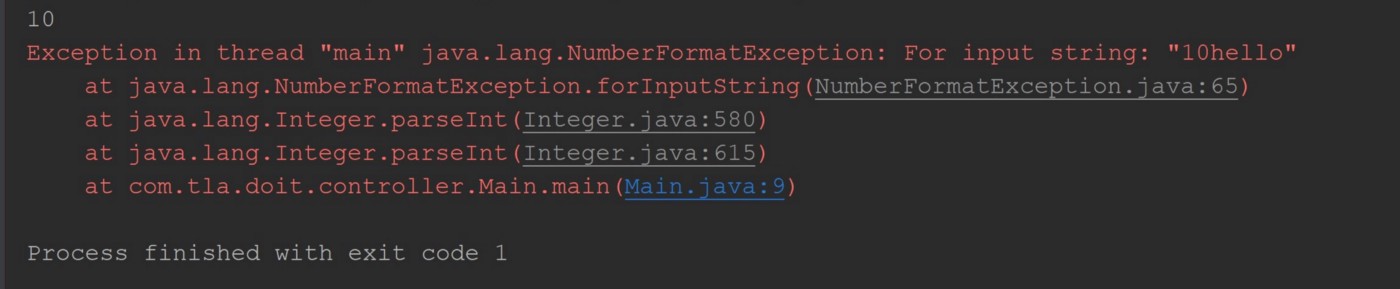
**What is an exception?**

An exception is a mechanism in Java to handle unexpected situations. Most of the time unexpected situation is unexpected input.



in Main.java file

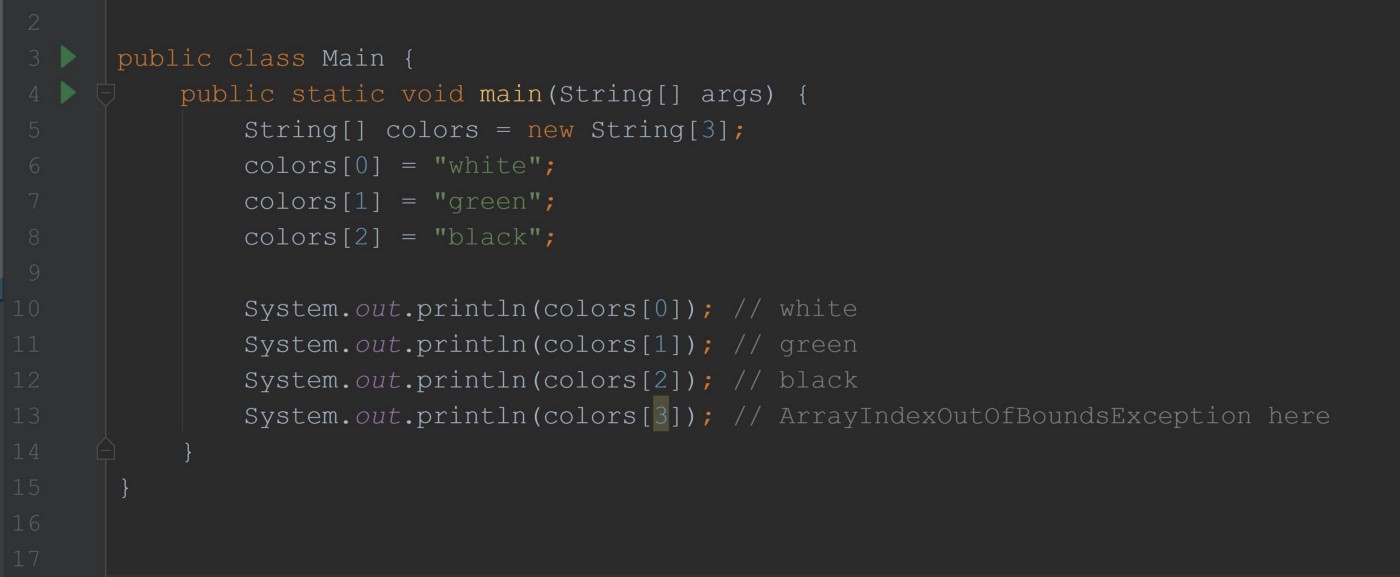
For example, to convert string number to int number we can use *Integer.parseInt(str)* method. It takes a string as an argument and expects we will pass the number as a string to get it as int. But no one can stop us from passing any string. If we will pass the string “10hello”, what program should do? Yes, throw an exception and let you know that input was wrong.



The output from the above program

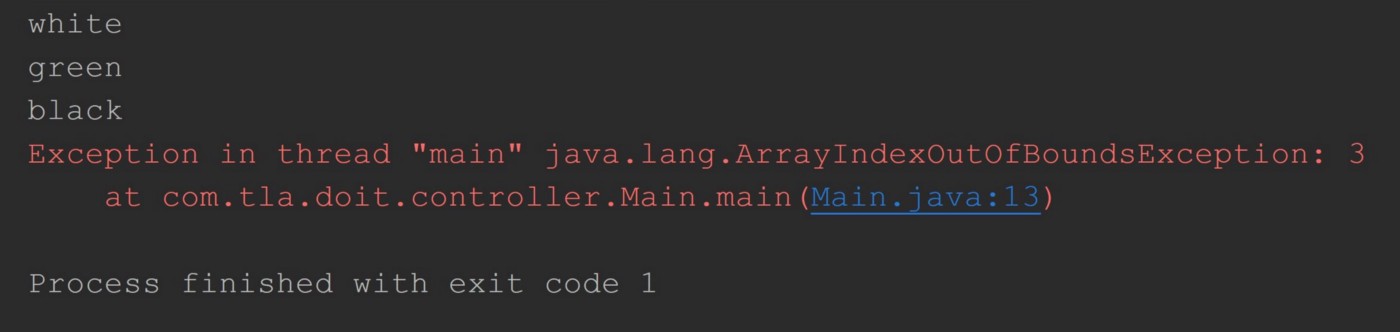
The above picture shows an example of the thrown exception by our program.

Let’s take a look at one more example.



in Main.java file

In the above example, we will ArrayIndexOutOfBoundsException. We created an array of size 3 and trying to access the 4th element. There is no 4th element, our array has only 3 elements. So java will throw an exception which totally makes sense.



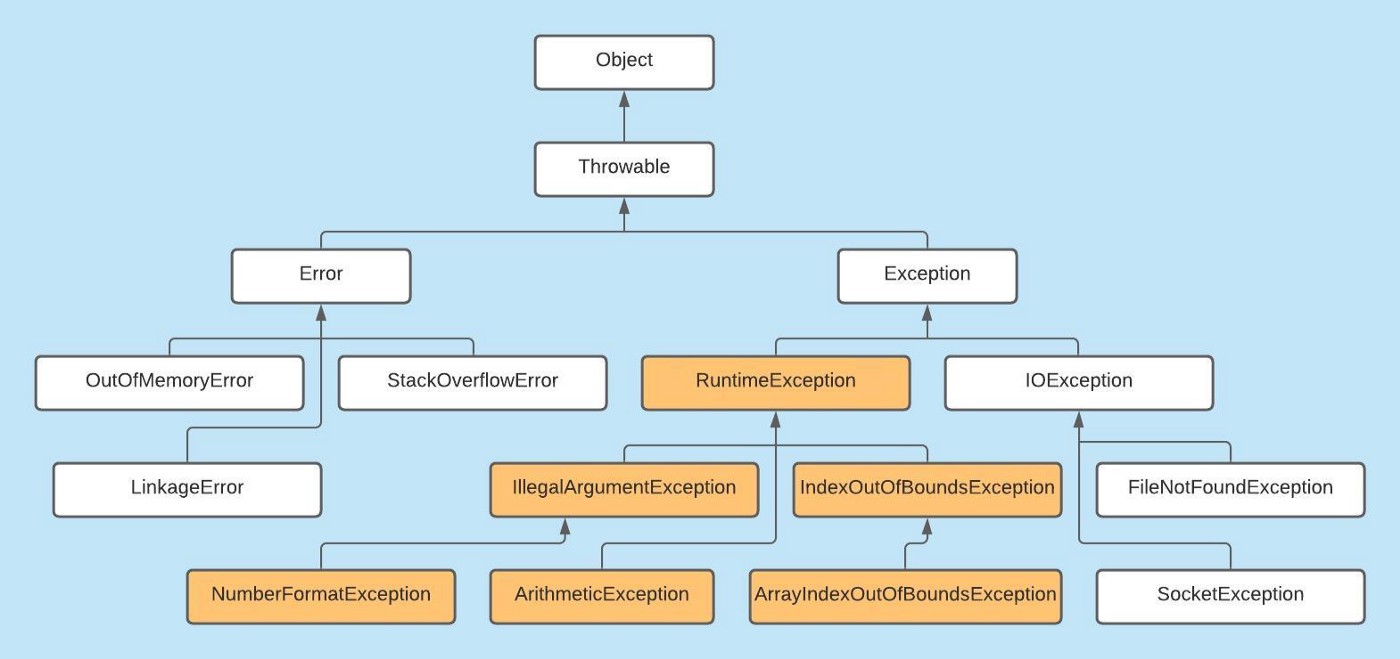
The output from the above program

When the exception is thrown, you will see an exception stack in the console. The exception stack is a set of classes that involved in a specific issue. We know that one class might use another class and that class another one and so on so the exception stack will show you this chain of classes. It goes from top to bottom. The first line of the stack is the most detailed one and it will have the main message and reason of exception. We always read exception stack from top to bottom.

**Exception types**

Exceptions in java represented by a set of classes. There are two types of exceptions — runtime(unchecked) exception and checked exception. We also have an Error in java. Errors are not exceptions and we never should handle and throw errors inside our program. By behavior, errors are similar to runtime exceptions.

**Runtime Exceptions**



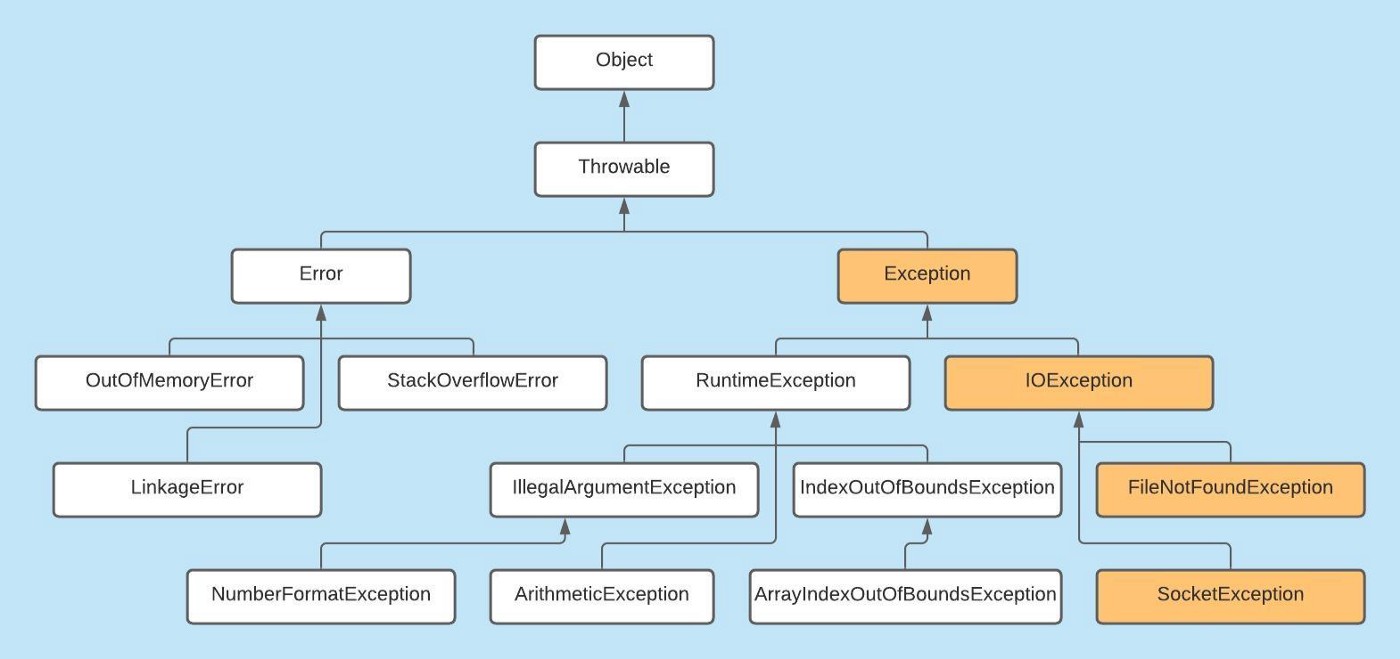
Runtime Exceptions in Java

The above picture shows an exception framework diagram(not all exception classes included). Notice Throwable class has two direct child classes Error and Exception. We shouldn’t throw or handle Throwable(it’s possible). Every class which extends Error is basically an error and every class which extends Exception is an exception.

Now, Runtime exceptions highlighted in the picture.

1. Runtime exceptions are exceptions that extend RuntimeException class
2. They are optional to handle. We will discuss how to handle exceptions below.

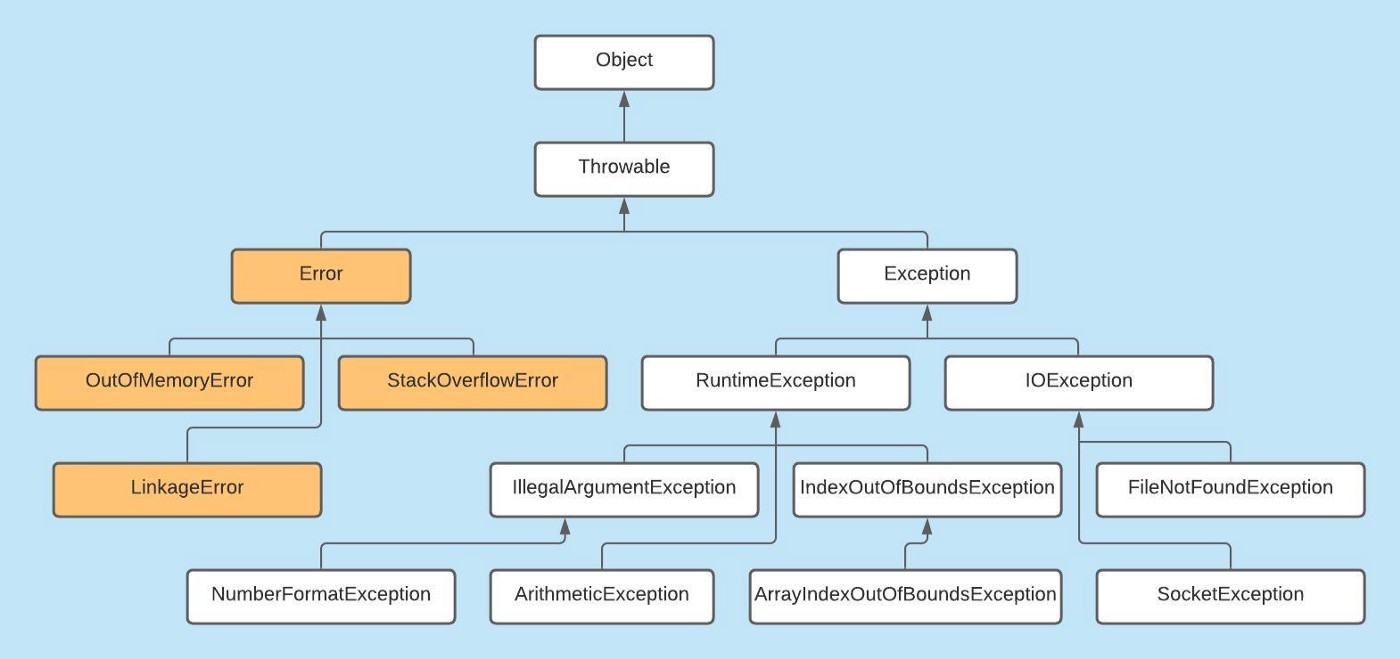
**Checked Exceptions**



Checked Exceptions in Java

1. The Exception class and its subclasses, except RuntimeException, are checked exceptions.
2. Java requires the code to handle checked exceptions otherwise it will not simply compile.

**Error**

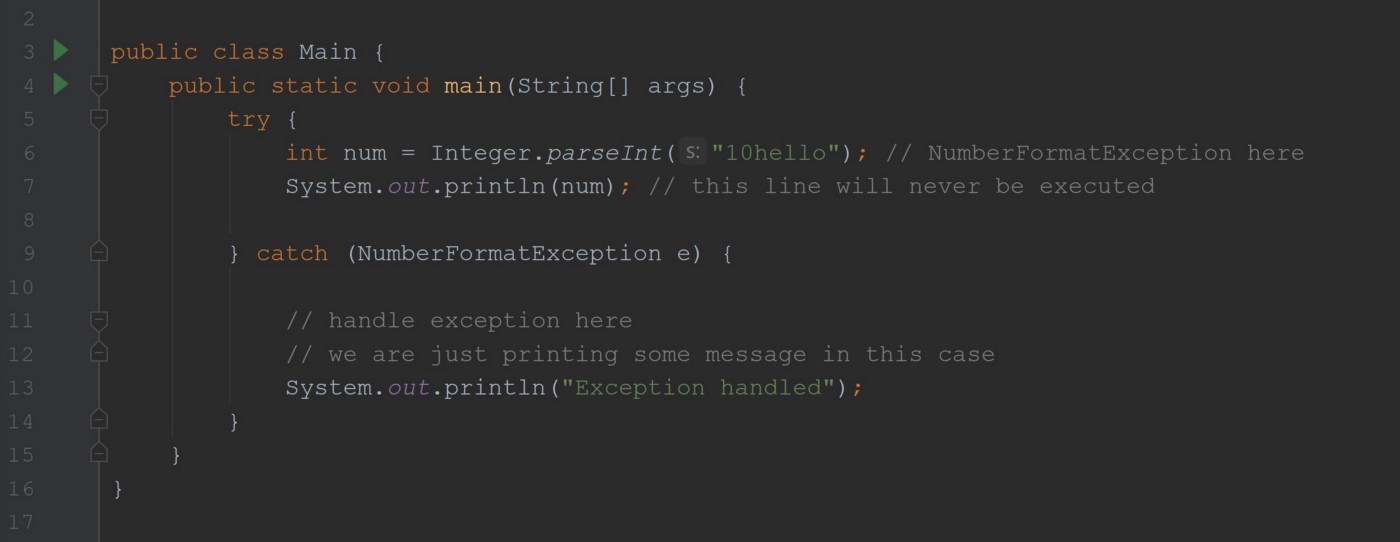


Error in Java

1. Errors are thrown by JVM and should not handle them. Errors are considered when serious system failures(as memory overflow) occur. So there is no point to handle it at the application level.
2. We never throw an Error.
3. Errors are rare.

**How to handle an exception?**

To handle exceptions we can use a try-catch statement.



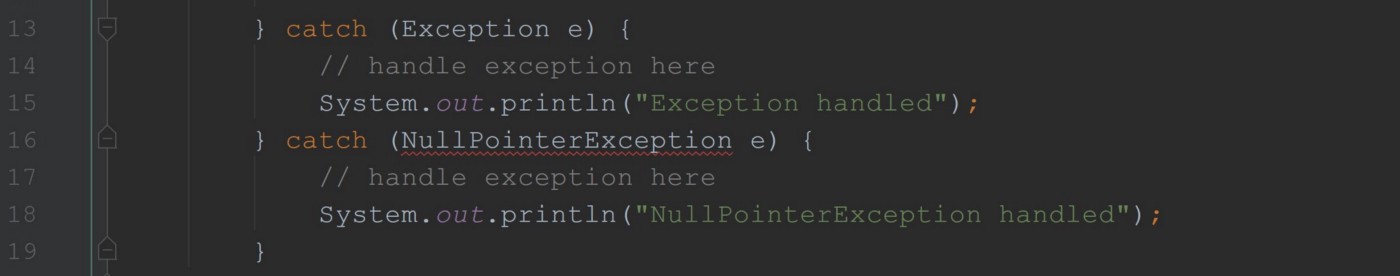
in Main.java file

Code that potentially throws an exception we can put in the try statement’s body. If the exception is thrown it will be handled by the catch statement and its body will get executed. In the catch statement, we can specify what kind of exception to handle. In the above example, we are handling a *NumberFormatException.*So when the exception will occur and if we are catching this exception, the body of the catch statement will be executed. That’s how we handle it. You might just print something as we did in our example or you can do anything you want, it’s really up to your requirement and program specifications.



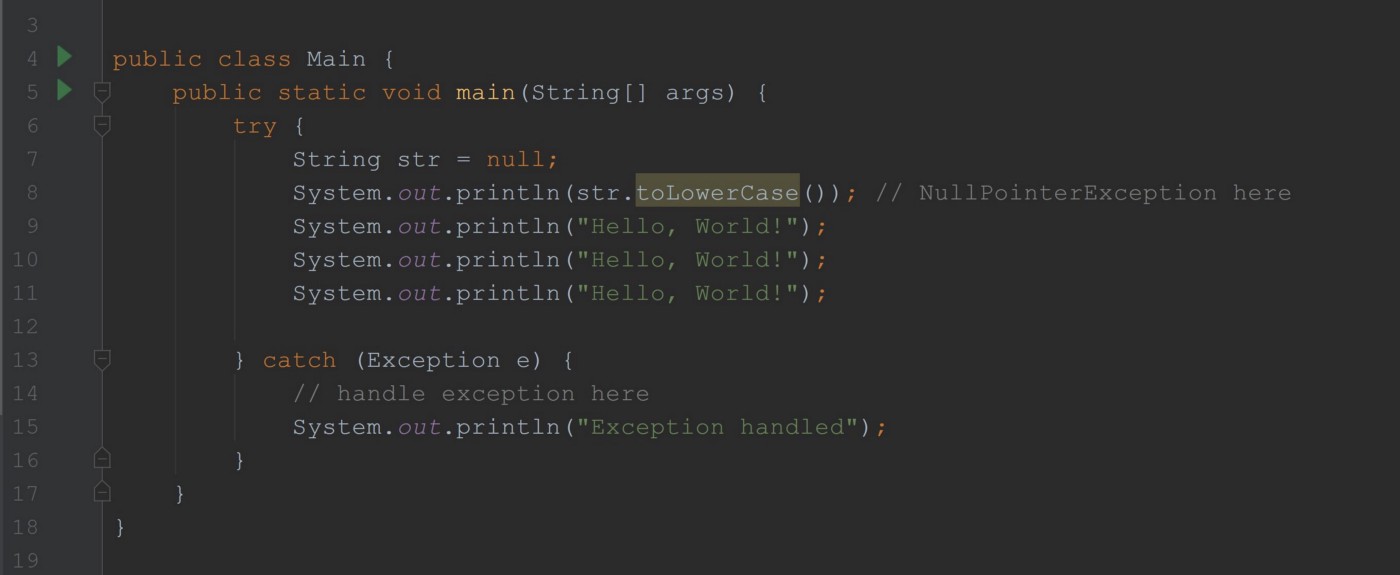
in Main.java file

We can have multiple catch statements in our try-catch statement. This allows us to handle different exceptions differently. There is one rule — a smaller(or we can say subclass exception) exception always should be on the top. In our example, NullPointerException is a subclass(not direct) of Exception so we put it first.



it does not compile

If we will put a bigger(parent) exception first, it will not compile because parent exception can catch all its child exceptions.



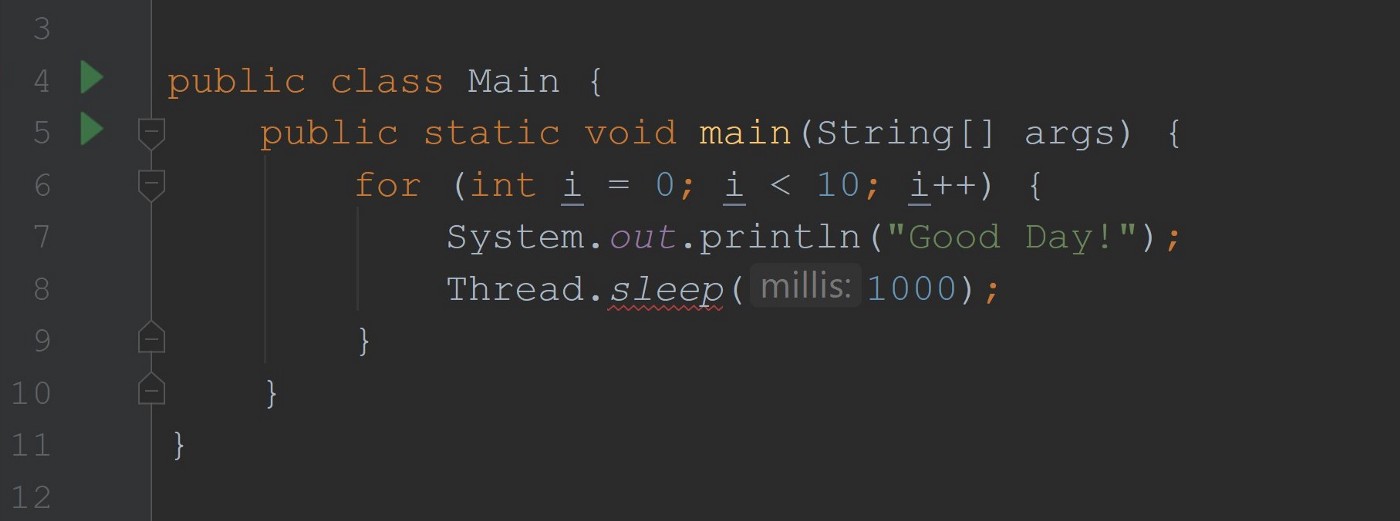
in Main.java file

The above example works fine. Even though the program will throw *NullPointerException,*the *Exception* will catch it. As we already discussed above *Exception*can catch it because it’s the parent class of *NullPointerException. The output* of this program will be

Exception hanled

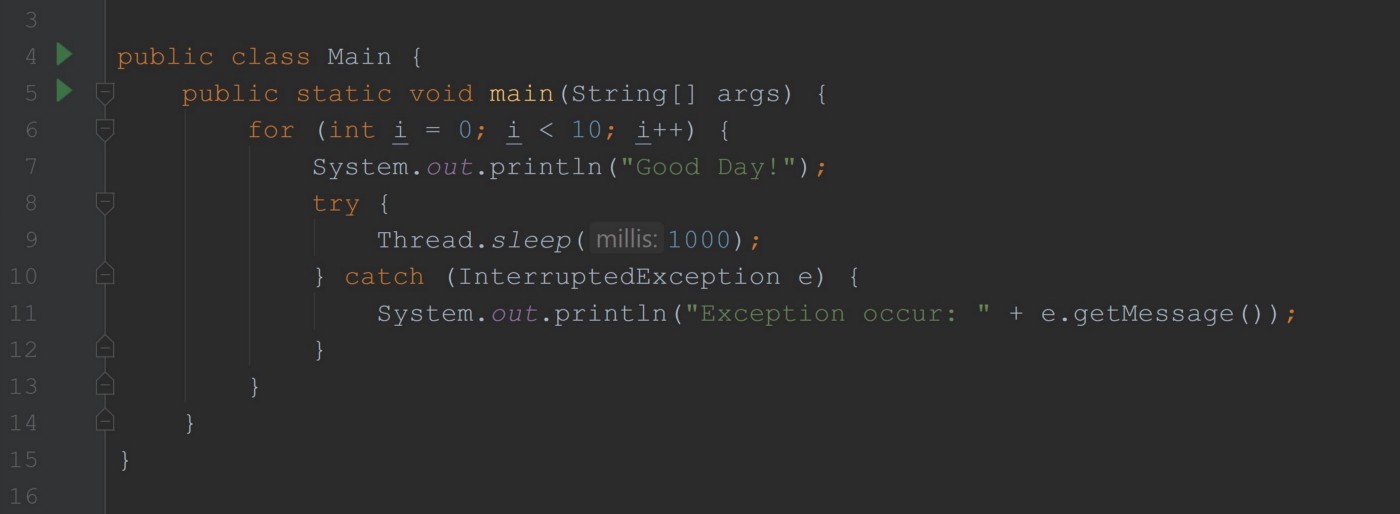
Notice, we don’t have Hello, World! three times in the output. From the line where the exception is thrown, it will directly go to the catch statement body if we are handling the proper exception type. Otherwise, it will throw it to the console.

Now, let’s discuss how to declare an exception and we will talk about checked exceptions.



Compilation Error - Unhandled exception: java.lang.InterruptedException

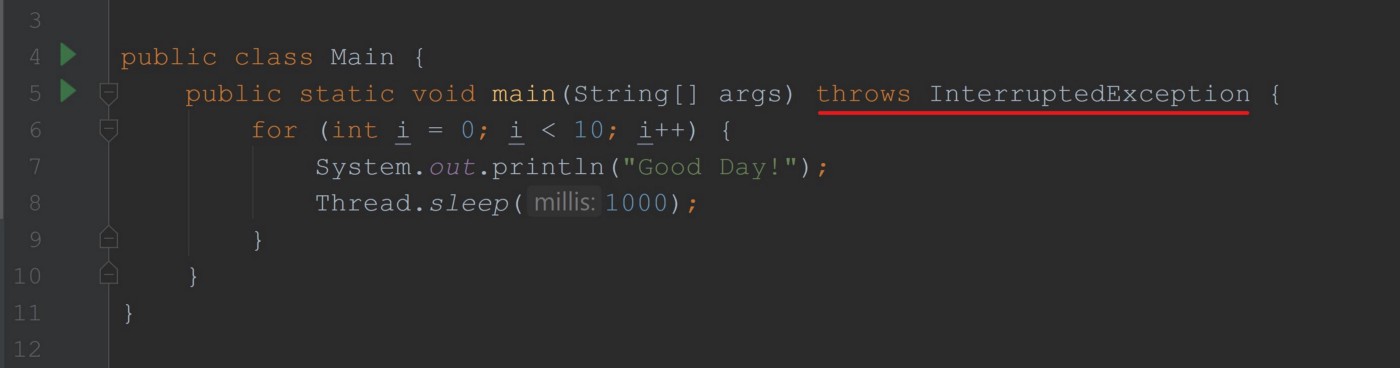
Our above example supposed to print Good Day! 10 times and wait 1 second after each print. To wait, we are using Thread.sleep(1000) . The reason it’s not compiling because our sleep method throws checked exception. And we must handle or declare checked exceptions.



in Main.java file

Now, it’s working fine and it will print Good Day! 10 times and wait one second after each. We handled it with a try-catch statement. There is nothing wrong with this code, and no exception will occur. And then why do we need to handle it? Because that’s how it works in Java. Most of the time the methods which throws checked exceptions that might throw a real exception so it makes client code to handle it even though it might not throw any exception. In our example Thread.sleep(1000);method pausing the thread and threads are managed by OS. I’m thinking that this method cannot give a %100 guarantee on pausing the thread so it throws checked the exception and force the client code to handle it.

We can use an instance of exception e(in our case) to get information about this exception.

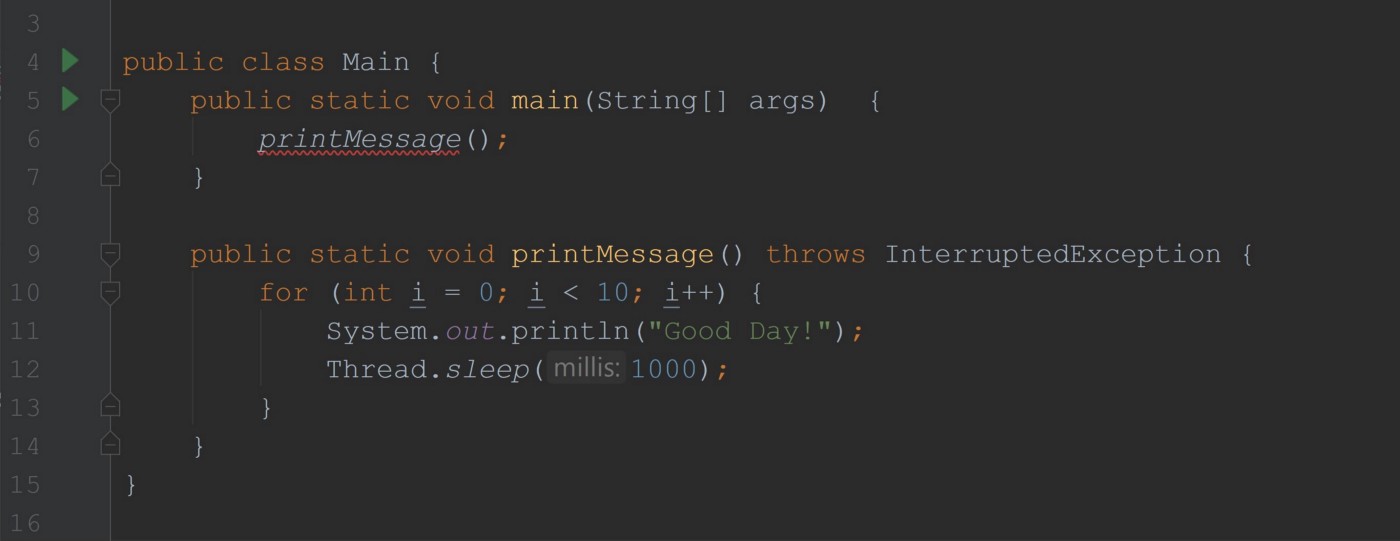


in Main.java file

The above example also works just fine without compilation errors. Notice we removed try-catch and added throws InterruptedException in the main method declaration. This is the way we declare exceptions. Exceptions can be declared only at the method level in the method declaration. Declaring an exception means we are telling that our method potentially throws an exception. If you declared a checked exception in your method, the code that will be using your method should handle it or declare it as well.

public static native void sleep(long millis) throws InterruptedException;

That’s how the declaration of Thread.sleep method looks like. You see it declares InterruptedException in its method declaration. When we were using we had to handle or declare it.



Compiler Error in line 6 — Unhandled exception: java.lang.InterruptedException

We created a separate method to print our messages with Thread.sleep. We are declaring throws InterruptedException in our method and everything is good there. But look when we are using our method it is giving an unhandled exception compiler error. So line 12 in our method throws a checked exception and we must handle it. We are handling it by declaring it in our method. What we actually saying is this — “I know my code throws a checked exception, but I don’t want to handle it, and a code that will use this method must handle it”. That’s why line 6 is giving compiler error.

You can declare a parent exception and it will handle all child exceptions. Also, you can declare multiple exceptions.

**How to throw an exception?**

We can throw our own exception.



in Person.java file

public void setAge(int age) {  
 if (age < 0) {  
 throw new IllegalArgumentException("Age cannot be negative");  
 }  
  
 this.age = age;  
}

Here, we are throwing our own exception. We use throw keyword and create an instance of an exception we need to throw. In our case, we are throwing the exception if age provided as negative.

**Custom exceptions**

We can create our own exception type by extending existing exceptions. If we want to create a checked exception we can extend Exception class and if we want to create a runtime exception we can extend RuntimeException class.