Stack trace

It shows the call stack in the application up to the point where the stack trace message was generated.

It appears as a message in your IDE when the application throws an error.

the call stack is a LIFO data structure providing information about the execution order of methods. It is composed of **stack frames**. Each stack frame represents a single method.

Stack trace in details

```
import java.util.Scanner;

public class NumberFormatExceptionDemo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String input = scanner.nextLine();

        int number = Integer.parseInt(input); // an exception is possible here!
        System.out.println(number + 1);
    }
}
```

If we enter a word instead of a number, for instance, "Java," the application throws an error and shows this stack trace message:

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "Java"
        at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:67)
        at java.base/java.lang.Integer.parseInt(Integer.java:668)
        at java.base/java.lang.Integer.parseInt(Integer.java:786)
        at NumberFormatExceptionDemo.main(NumberFormatExceptionDemo.java:8)
```

1. The thread in which the exception was thrown. If you remember, when the application starts, it creates a main thread.
2. The class responsible for the type of error. In our case, it is a NumberFormatException class from java.lang package.
3. A message indicating why the exception was thrown.

Now let's move on and explore the remaining four lines. The very first line from the bottom indicates line 8, which is found in the main method of NumberFormatExceptionDemo class. This is the line of the program whose

execution led to the exception. The next invoked method was parseInt(String s) from the Integer class accepting one String argument. Inside this method, another overloaded parseInt(String s, int radix) method from the Integer class was invoked on line 786.

https://ucarecdn.com/02075864-0be9-4232-95cf-1ae6ec5563b1/

Inside the second parseInt(String s, int radix) method, on line 668 the application throws an exception invoking
the NumberFormatException.forInputString(String s, int radix) method.

https://ucarecdn.com/83588978-04b9-41b7-9865-d7febd333616/

Finally, in the fourth line from the bottom, we can see the invocation of the forInputString(String s, int radix) static method from the NumberFormatException class. Below, on line 64, you can see the message from the stack trace example above. That is how the message from the very first line was generated.

https://ucarecdn.com/cba1532b-918d-420a-819f-6137fc6cf7e1/-/stretch/off/-/resize/2200x/-/format/webp/

In this topic, we are using Java 17, so the line numbers of base Java classes can vary depending on the Java version.

Now let's make some changes to our application. We are going to move part of the code to the method so that it will also be called when executing the application.

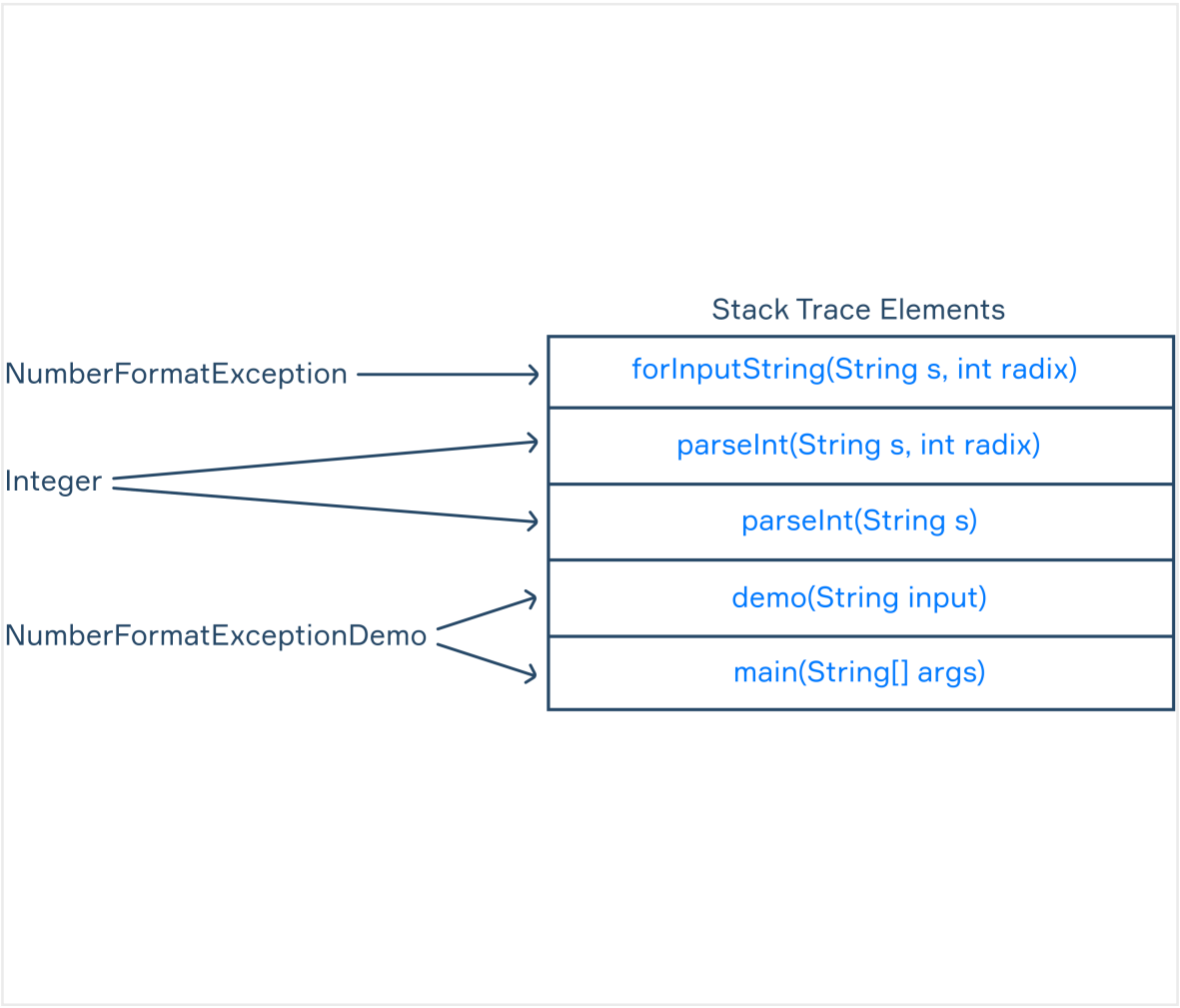
```
import java.util.Scanner;

public class NumberFormatExceptionDemo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String input = scanner.nextLine();

        demo(input);
    }

    public static void demo(String input) {
        int number = Integer.parseInt(input); // an exception is possible here!
        System.out.println(number + 1);
    }
}
```

This time we have one more line in our stack trace, representing the execution of the demo(String input) method.

```
Exception in thread "main" java.lang.NumberFormatException: For input string:
"Java"
        at java.base/
java.lang.NumberFormatException.forInputString(NumberFormatException.java:
67)
        at java.base/java.lang.Integer.parseInt(Integer.java:668)
        at java.base/java.lang.Integer.parseInt(Integer.java:786)
        at
NumberFormatExceptionDemo.demo(NumberFormatExceptionDemo.java:12)
        at
NumberFormatExceptionDemo.main(NumberFormatExceptionDemo.java:8)
```

Basically, this stack trace message shows the call stack from the main method to the place where the exception was thrown.

The following diagram represents the call stack of the example above. Since the call stack is a LIFO data structure, the main(String[] args) method that was called when the application was launched is at the bottom, and it will be the last printed element of the stack trace.



Getting a Stack Trace on demand

We have shown an example of getting a stack trace after your application throws an error. What if you need to get a stack trace at any specific point? It can be obtained without throwing an error by calling the Thread.currentThread().getStackTrace() method. This way it returns a StackTraceElement array, and you can print the stack trace using a loop.

```
for (StackTraceElement element : Thread.currentThread().getStackTrace()) {
    System.out.println(element);
}
```

There are also other ways of getting a stack trace, such as calling the new Throwable().getStackTrace() or new Throwable().printStackTrace() methods.

StackTraceElement class: an overview

According to the official Java Documentation, the StackTraceElement class is described as an element in a stack trace representing a single stack frame. That is, each element returned by Thread.currentThread().getStackTrace() is a stack frame where the element printed at the top represents the execution point where the stack trace was generated.

```
import java.util.Scanner;

public class NumberFormatExceptionDemo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String input = scanner.nextLine();

        demo(input);
    }

    public static void demo(String input) {
        for (StackTraceElement element : Thread.currentThread().getStackTrace())
{
            System.out.println(element);
        }

        int number = Integer.parseInt(input); // an exception is possible here!
        System.out.println(number + 1);
    }
}
```

If we input a number and the application does not throw an exception, the stack trace message will print the following three lines.

java.base/java.lang.Thread.getStackTrace(Thread.java:1610)
NumberFormatExceptionDemo.demo(NumberFormatExceptionDemo.java:13)
NumberFormatExceptionDemo.main(NumberFormatExceptionDemo.java:8)

The most useful feature of the StackTraceElement class is that it provides methods to simplify these lines and get only the necessary information. If you print System.out.println(element.getClassName()) inside the mentioned loop, you will get the stack trace message in this form:
java.lang.Thread
NumberFormatExceptionDemo
NumberFormatExceptionDemo
Other methods of the class, such as getMethodName() or getLineNumber(), work in a similar way.