Exceptions

IT TALENTS
Training Camp

Introduction

- Sometimes during the execution of the program something goes wrong
- Our program should be designed to recover from these errors if it is possible
- Differences between compile time errors and runtime errors (exceptions)

IT TALENTS
Training Camp

Error codes

How the programmers used to deal with the runtime errors in the past?

Using error codes as return value

Error codes example

```
public class FileUtils {
    static int copyFile(String pathTofile1, String pathTofile2) {
        //open file
        if (<file1 do not exist>){
            return -1;
        }

        //create file
        if (<no access to create file2>){
            return -2;
        }

        while (<the end of file 1 is not reached>) {
            //read line from file 1
            if(<connection fails or something goes wrong>){
                return -3;
            }
            // write to file 2
            if(<no freespace to write into file 2>){
                return -4;
            }
        }
        return 0;
    }
}
```

Error codes example

```
public static void main(String[] args) {
        int result = FileUtils.copyFile("C:\\file1.txt", "D:\\test\\file2.txt");

        switch (result) {
        case -1:
            System.out.println("Error: File 1 do not exist.");
            break;
        case -2:
            System.out.println("Error: No permission to create the new file");
            break;
        case -3:
            System.out.println("Error: The connection to file 1 has been lost or
                something goes wrong.");
            break;
        case -4:
            System.out.println("Error: No free place to write in file 2");
            break;
        case 0:
            System.out.println("The file has been copied successfully.");
            break;
        }
    }
```

Exceptions in Java

- Java use exceptions to deal with runtime errors
- An exception is an event, which occurs during the execution of a program. It disrupts the normal flow of the program's instructions.
- This way the logic for handling exceptions is separated from the business logic and the code is clearer and easyer to read
- The exceptions are more convenient and powerful than the error codes

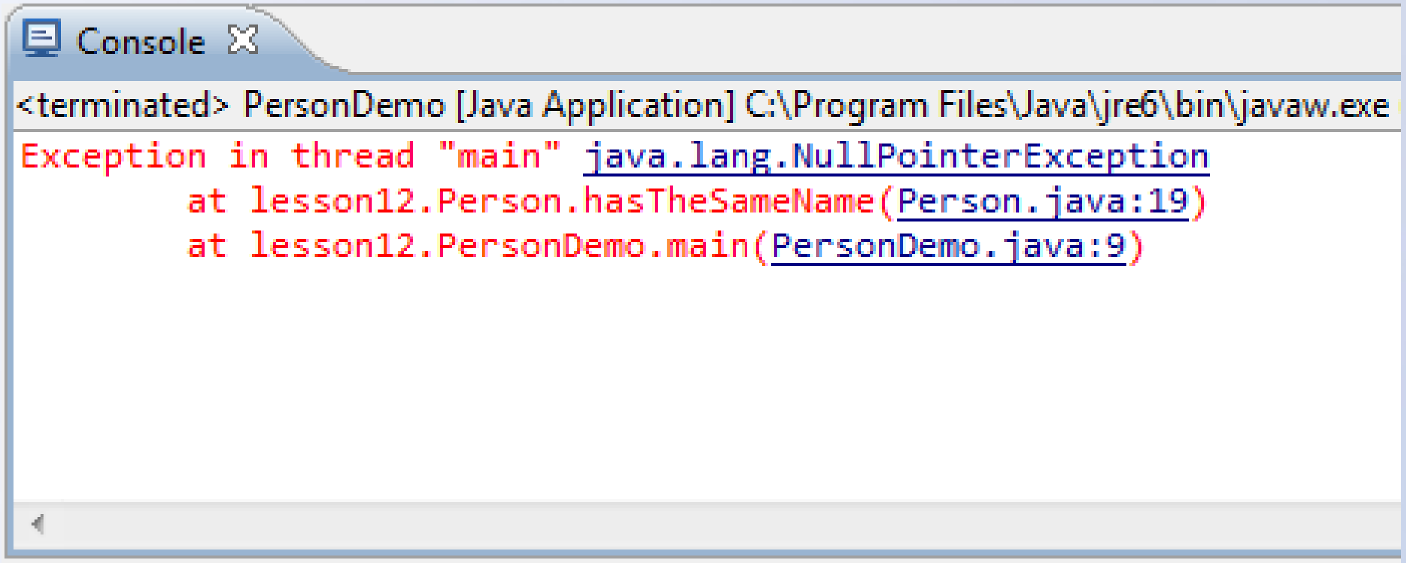
IT TALENTS
Training Camp

Example for exception

```
public class Person {

    private String name;

    public Person() {

    }

    public Person(String name) {
        this.name = name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public boolean hasTheSameName(Person p) {
        return name.equals(p.name);
    }
}
```

Example for exception

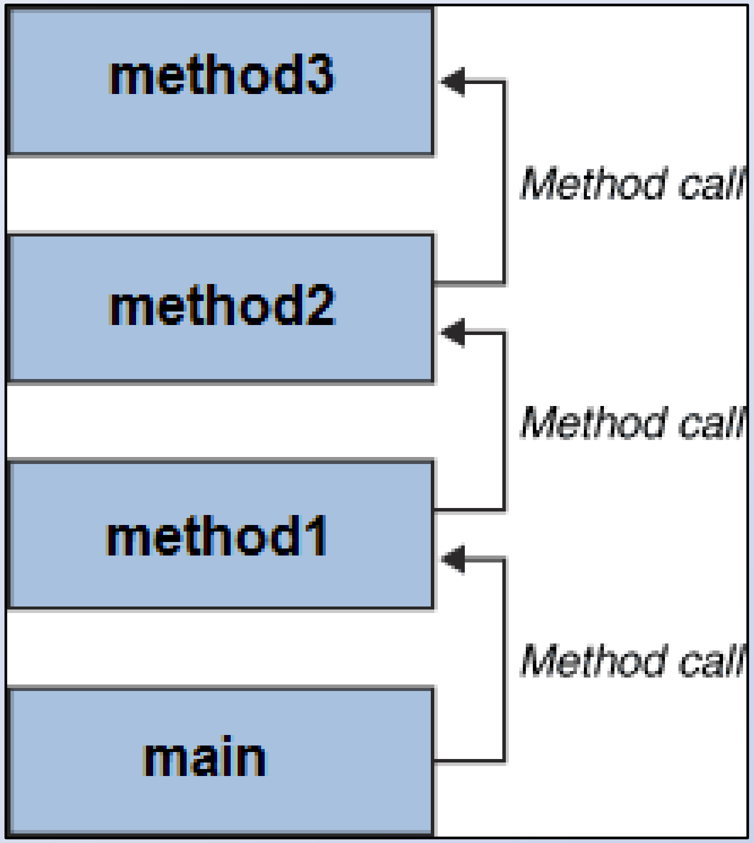
```
public class PersonDemo {

    public static void main(String[] args) {
        Person john = new Person();
        Person johnie = new Person("John");

        john.hasTheSameName(johnie);
    }

}
```



```
<terminated> PersonDemo [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe
Exception in thread "main" java.lang.NullPointerException
	at lesson12.Person.hasTheSameName(Person.java:19)
	at lesson12.PersonDemo.main(PersonDemo.java:9)
```

What is call stack

- The list of methods which was called is known as the call stack
- When a method calls another method the new method is put on top of the call stack
- When the top method finishes, it's removed from the call stack
- Method in the lower level finishes when all the methods on top of it are finished



IT TALENTS Training Camp

How exception system works

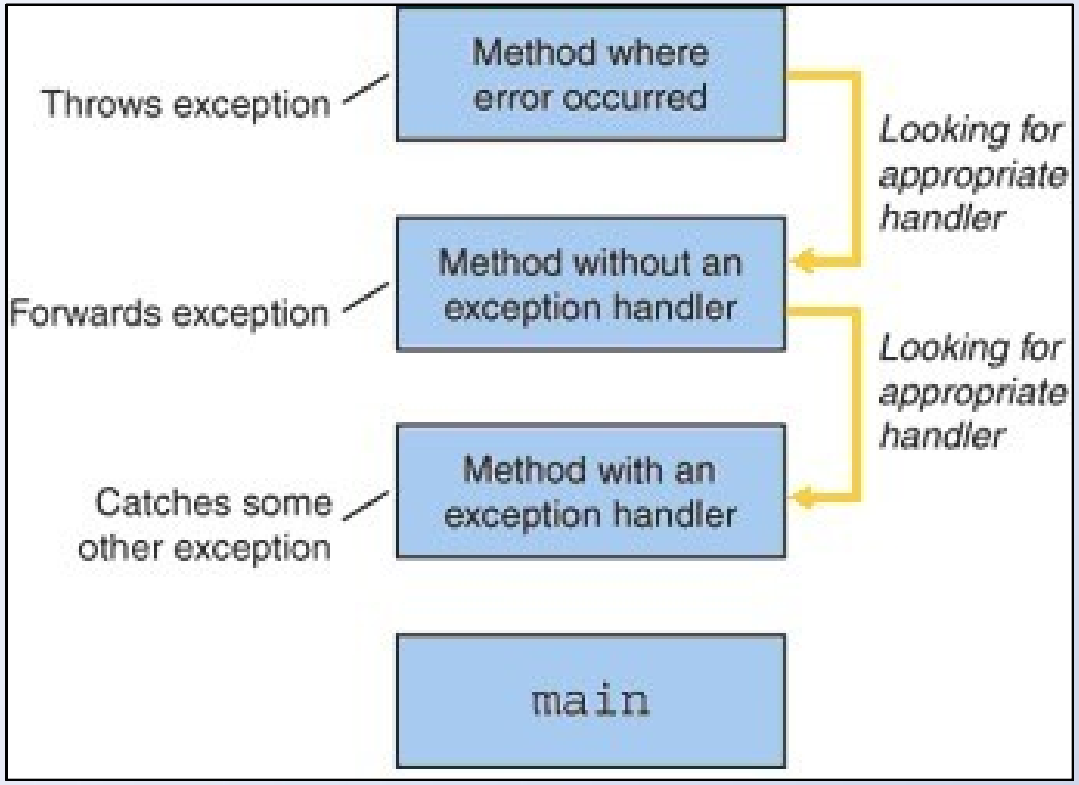
- When an error occurs within a method, the method creates an object and hands it off to the runtime system
- The object, called an exception object, contains information about the error
- Creating an exception object and handing it to the runtime system is called throwing an exception.
- After a method throws an exception, the runtime system attempts to find something to handle it

Searching for exception handler

- The runtime system searches the call stack for a method that contains a block of code that can handle the exception.
- This block of code is called an exception handler
- The search begins with the method in which the error occurred and proceeds through the call stack in the reverse order in which the methods were called

IT TALENTS
Training Camp

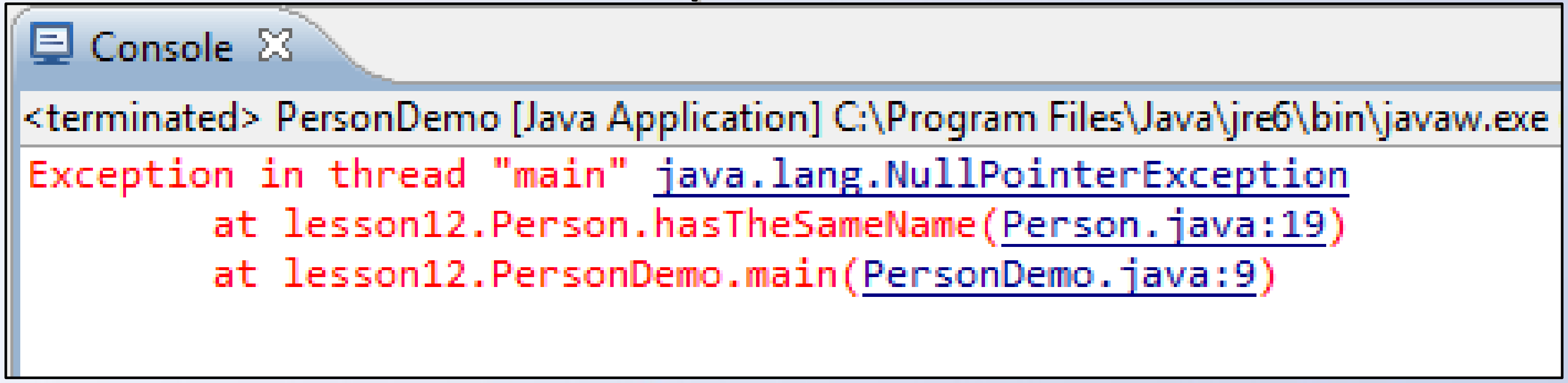
Searching for exception handler


Method where error occurred
Throws exception
Looking for appropriate handler
Method without an exception handler
Forwards exception
Looking for appropriate handler
Method with an exception handler
Catches some other exception
main

IT TALENTS
Training Camp

Stack trace

- It contains detailed information about the exception
- It consists of:
  - The full name of the exception
  - Message of the exception
  - Information about the call stack and the current line where the exception occurred



```
<terminated> PersonDemo [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe
Exception in thread "main" java.lang.NullPointerException
	at lesson12.Person.hasTheSameName(Person.java:19)
	at lesson12.PersonDemo.main(PersonDemo.java:9)
```

try-catch block

```
try {

    //dangerous code

} catch (<ExceptionType name>) {

    //exception handler code

}
```

Here is the code which
may throw exception

List with exception handlers
(can be more than one).
It contains the code which will
fix the program from the error (if possible)

IT TALENTS
Training Camp

try-catch block

- The try block contains one or more legal lines of code that could throw an exception
- You associate exception handlers with a try block by providing one or more catch blocks directly after the try block
- Each catch block is an exception handler and handles the type of exception indicated by its parameter
- The catch block contains code that is executed if and when the exception handler is invoked

IT TALENTS
Training Camp

The previous example with exceptions

```
public class FileUtils {

    /**
     * Copy the file located in pathTofile1 into new file in pathTofile2
     */
    static void copyFile(String pathTofile1, String pathTofile2) {
        try {
            //open file
            //create file
            while (<the end of file 1 is not reached>) {
                //read line from file 1
                // write to file 2
            }
        } catch (FileNotFoundException e) {
            System.out.println("Error:No file with this name has been found");
        } catch (IOException e) {
            System.out.println("Error: Copy not successful");
        }
    }
}
```

Exercise

- Write static method `void printArrayInfo(int[] array)` which prints the length of the array to the console. It also prints the third element to the console and after that, prints some text.
- Use exception handlers for NullPointerException and ArrayIndexOutOfBoundsException instead of if clauses

This is not the best way to avoid these exceptions

(if clauses should be used) but we do this only for demonstration purpose for this exercise

IT TALENTS Training Camp

Exercise

```java
public class ArrayInfoExample {

    public static void printArrayInfo(int[] array) {
        try {
            System.out.println("The array length is " + array.length);
            System.out.println("Some text.....");
            System.out.println("The third element is " + array[2]);
        } catch (NullPointerException e) {
            System.out.println("The array is null!");
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("The array must have at least 3 elements");
        }
    }

    public static void main(String[] args) {
        int[] array = null;
        printArrayInfo(array);
    }
}
```

IT TALENTS
Training Camp

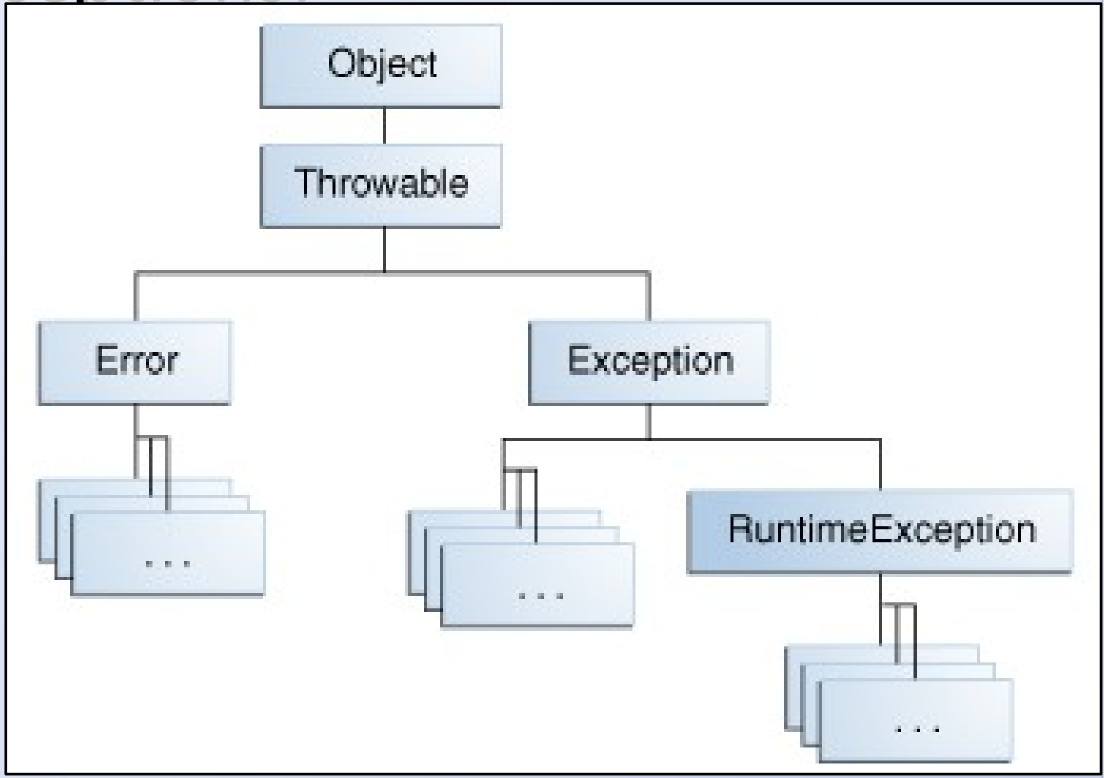
Exercise

- The result is: The array is null!
- As we see the text “Some text“ is not shown to the console
- This is because when exception occurred, all the code in the try block after this is not executed.
- As we see, we can have more than one exception handlers
- If there is no handler for the exception after try block, the exception is thrown to the previous method in the call stack. Try it!

IT TALENTS
Training Camp

Throwable class and its subclasses

- Throwable – parent of all exceptions
- There are 3 types of exceptions:
  - Errors
  - Unchecked exceptions
  - Checked exception



IT TALENTS
Training Camp

Throwable class and its subclasses

- Errors – Indicate hard failure in the Java virtual machine. Simple programs typically do not catch or throw Errors, because the application usually cannot anticipate or recover from them.
- Exception - An Exception indicates that a problem occurred, but it is not a serious system problem. Most programs you write will throw and catch Exceptions as opposed to Errors.

More about Throwable class

An instance of Throwable class contains

- Message
- Stacktrace
- Cause (instace of Throwable)

Unchecked exeptions

- All classes that extend RuntimeException are called ***unchecked***.
- They usually indicate programming bugs, such as logic errors or improper use of an API
- Example of unchecked exceptions:
    - ArithmeticException
    - NullPointerException
    - IndexOutOfBoundsException
    - IllegalArgumentException
    - ClassCastException

Checked exeptions

- These are exceptional conditions that a well-written application should anticipate and recover from
- Checked exceptions in Java extend the Exception class but do not extend RuntimeException class
- Checked exceptions are subject to the

“Catch or Specify Requirement“:

When in method's body some code may throws checked exception, the method must either handle this exception or specify that it may throws this exception

Checked exeptions

```
public class FileUtils {

    static void copyFile(String pathTofile1, String pathTofile2) throws IOException {
        //open file
        //create file
        while (<the end of file 1 is not reached>) {
            //read line from file 1
            // write to file 2
        }
    }
}
```

Specify that copyFile method
Throws IOException

```
public static void main(String[] args) {
    try {
        FileUtils.copyFile("C:\\file1.txt", "D:\\test\\file2.txt");
    } catch (FileNotFoundException e) {
        System.out.println("Error: No file with this name has been found");
    } catch (IOException e) {
        System.out.println("Error: Copy not successful");
    }
}
```

More about exceptions

- We can manually throw an exception from the code using keyword *throw*
- We can invoke some methods to the exception object like printStackTrace(), getMessage()...

Chained Exceptions

An application often responds to an exception by throwing another exception.

In effect, the first exception causes the second exception.

It can be very helpful to know when one exception causes another.

Chained Exceptions help the programmer do this.

Wrapped exceptions (cause)

Why are necessary?

Chained Exceptions example

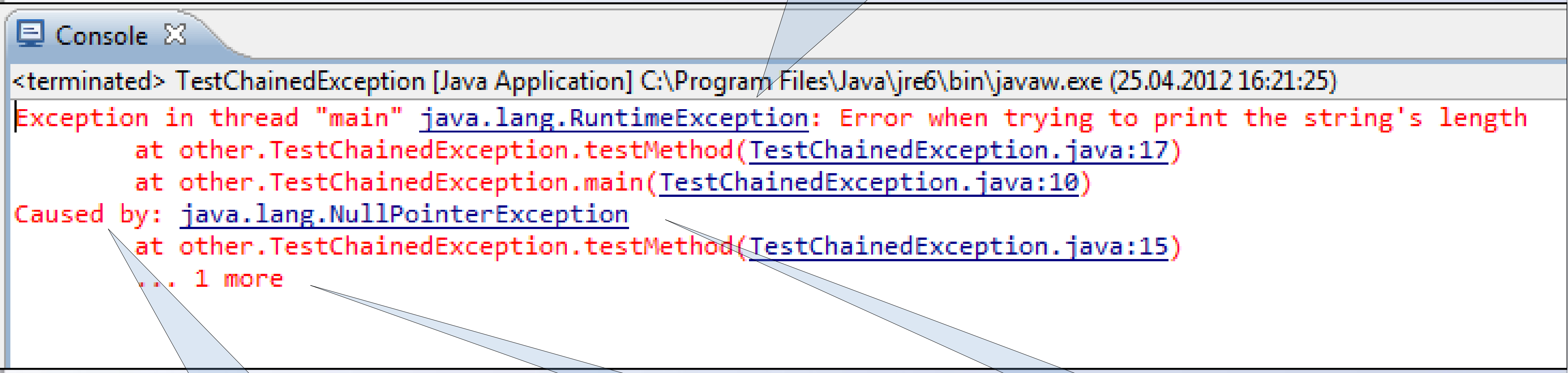
In this example, when an IOException is caught, a new SampleException exception is created with the original cause attached and the chain of exceptions is thrown up to the next higher level exception handler.

```
try {
    //...
} catch (IOException e) {
    throw new SampleException("Other IOException", e);
}
```

How to read stacktrace with chained exceptions

```
package other;

public class TestChainedException {
    public static void main(String[] args) {
        String s = null;
        testMethod(s);
    }

    public static void testMethod(String s) {
        try {
            System.out.println(s.length());
        } catch (NullPointerException npe) {
            throw new RuntimeException("Error when trying to print the
string's length", npe);
        }
    }
}
```

How to read stacktrace with chained exceptions



```
<terminated> TestChainedException [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (25.04.2012 16:21:25)
Exception in thread "main" java.lang.RuntimeException: Error when trying to print the string's length
	at other.TestChainedException.testMethod(TestChainedException.java:17)
	at other.TestChainedException.main(TestChainedException.java:10)
Caused by: java.lang.NullPointerException
	at other.TestChainedException.testMethod(TestChainedException.java:15)
	... 1 more
```

Console
Exception which broke the program
Caused by:
Hide unnecessary information
Wrapped exception (cause)

IT TALENTS Training Camp

How exceptions should be shown to the end user

- The end user is not a programmer
- So, it's not a good practice to show technical details (stacktrace) to the end user
- Instead, nice message should be shown
- If we want, we can add technical information but it should be shown only if the user want to see it

IT TALENTS
Training Camp

Global handler

- It's good practice to write a global handler in some upper layer which handle all uncatched exceptions (unchecked exceptions)
- This way we'll prevent user from unhandled exceptions and showing unpleasant messages

Question

What happens with this code?

```
try {
    //..
} catch (Exception e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

Answer

Compilaton error!

Unreachable catch block for IOException. It is already handled by the catch block for Exception

Because IOException extends class Exception,

so the second catch block will never execute.

Tips

We can handle multiple exceptions using catch block with parent class.

This is useful when we want to handle more than one exceptions in the same way

(we use a (too) general exception handler)

Re-throwing exception

- Sometimes we want to handle the exception just for a moment, use it for something (write in the log) and then re-throw it, because we can't handle it at all.
- Keyword *throw* is used(we saw it in the previous slides)

```
try {
    //...
} catch (IOException e) {
    logger.log(Level.WARNING, "Error in testMethod: " + e.getMessage());
    throw e;
}
```

IT TALENTS
Training Camp

Defining own exceptions

- Just extend the class Exception
- Do not create a subclass of RuntimeException or throw a RuntimeException
- If we need, we can add some fields to these which are inherited by Exception
- It's good practice each module to throw only its own exceptions
- For readable code, it's a good practice to append the string Exception to the names of all classes that inherit from the Exception class.

Defining own exceptions

```java
public class ITTalentsException extends Exception{

    public ITTalentsException() {
        super();
    }

    public ITTalentsException(String message, Throwable cause) {
        super(message, cause);
    }

    public ITTalentsException(String message) {
        super(message);
    }

    public ITTalentsException(Throwable cause) {
        super(cause);
    }

}
```

Finally block

- The finally block always executes when the try block exits.
- This ensures that the finally block is executed even if an unexpected exception occurs
- It allows the programmer to avoid having cleanup code accidentally bypassed by a return, continue, or break.
- Putting cleanup code in a finally block is always a good practice, even when no exceptions are anticipated.

IT TALENTS
Training Camp

Finally block

The finally block is a key tool for preventing resource leaks. When closing a file or otherwise recovering resources, place the code in a finally block to ensure that resource is always recovered.

```
try {
    // some code which open PrintWriter out
} catch (Exception e) {
    //.. handle exception
} finally {
    if (out != null) {
        System.out.println("Closing PrintWriter");
        out.close();
    } else {
        System.out.println("PrintWriter not open");
    }
}
```

Catch block is not mandatory. finally can exist without catch, but not without try.

Let's discuss some good practices when using exceptions

IT TALENTS
Training Camp