Lecture 12

# Exceptions (part 1)

## Introduction

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

## 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

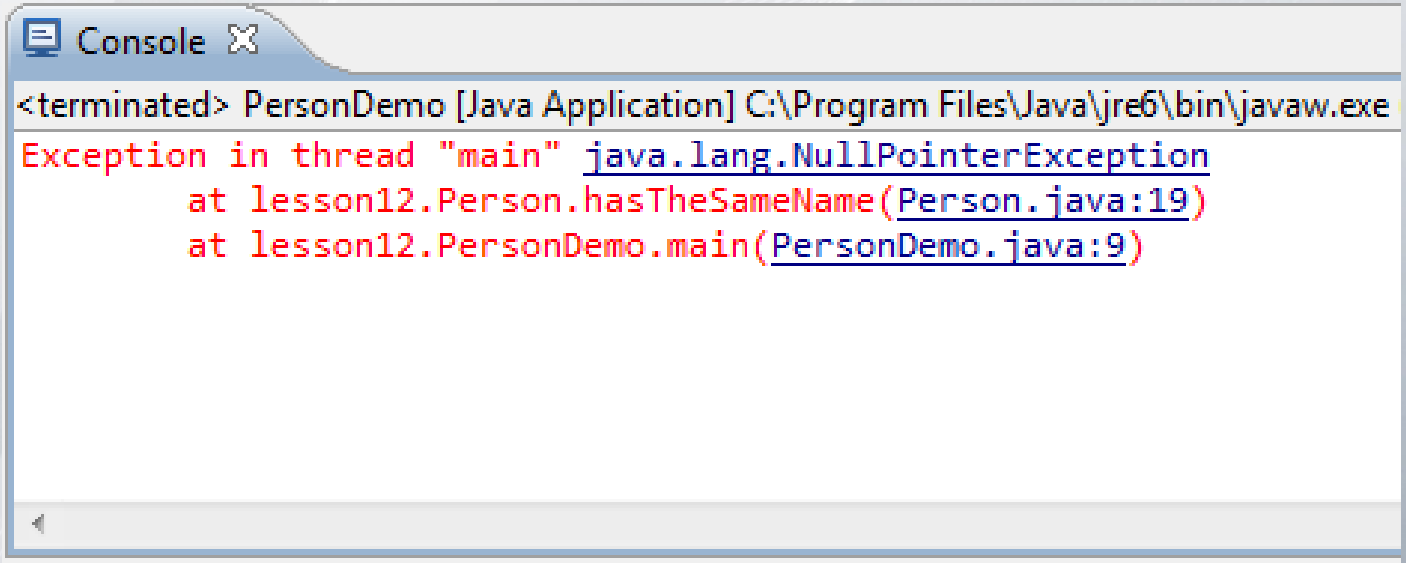
- Java use exceptions to deal with runtime errors
- An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.
- This way the logic for handle exceptions are separated than the business logic and the code is more clear and easy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

## Example for exception

```java
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

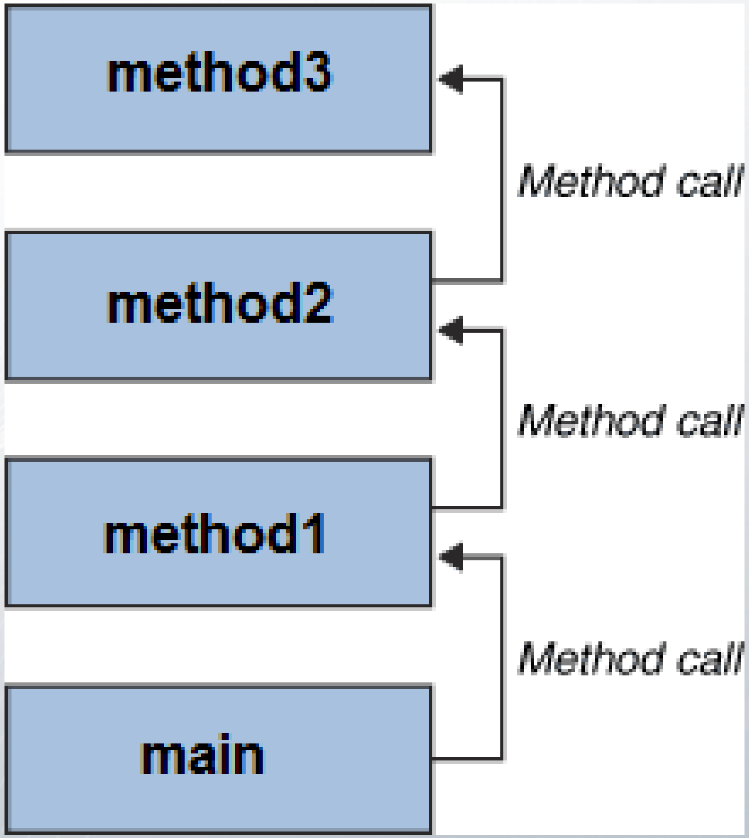
```
public class PersonDemo {

    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 call another method the new method is put on top of the call stack
- When the top method finish it's removed from the call stack
- Method in the lower level finish when all the methods on top of it are finished



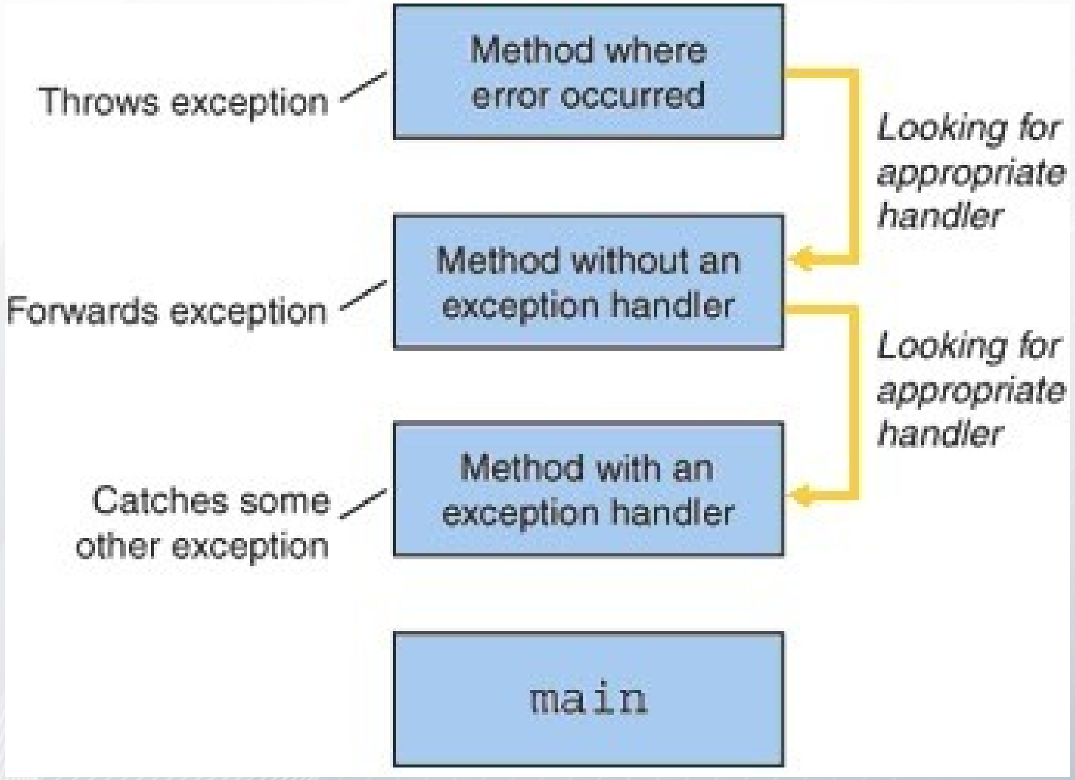
## How exception system works

- 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

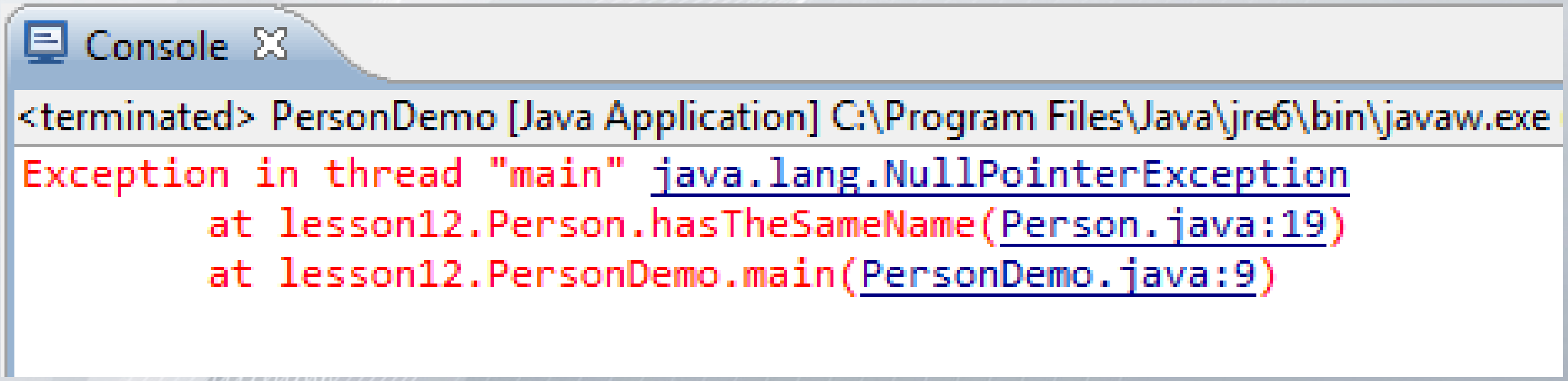
- The runtime system searches the call stack for a method that contains a block of code that can handle the exception.
- This block of code is called an exception handler
- The search begins with the method in which the error occurred and proceeds through the call stack in the reverse order in which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

## 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

## 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 is possible)*

## 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 argument
- The catch block contains code that is executed if and when the exception handler is invoked

# 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 this exceptions**

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

# Exercise

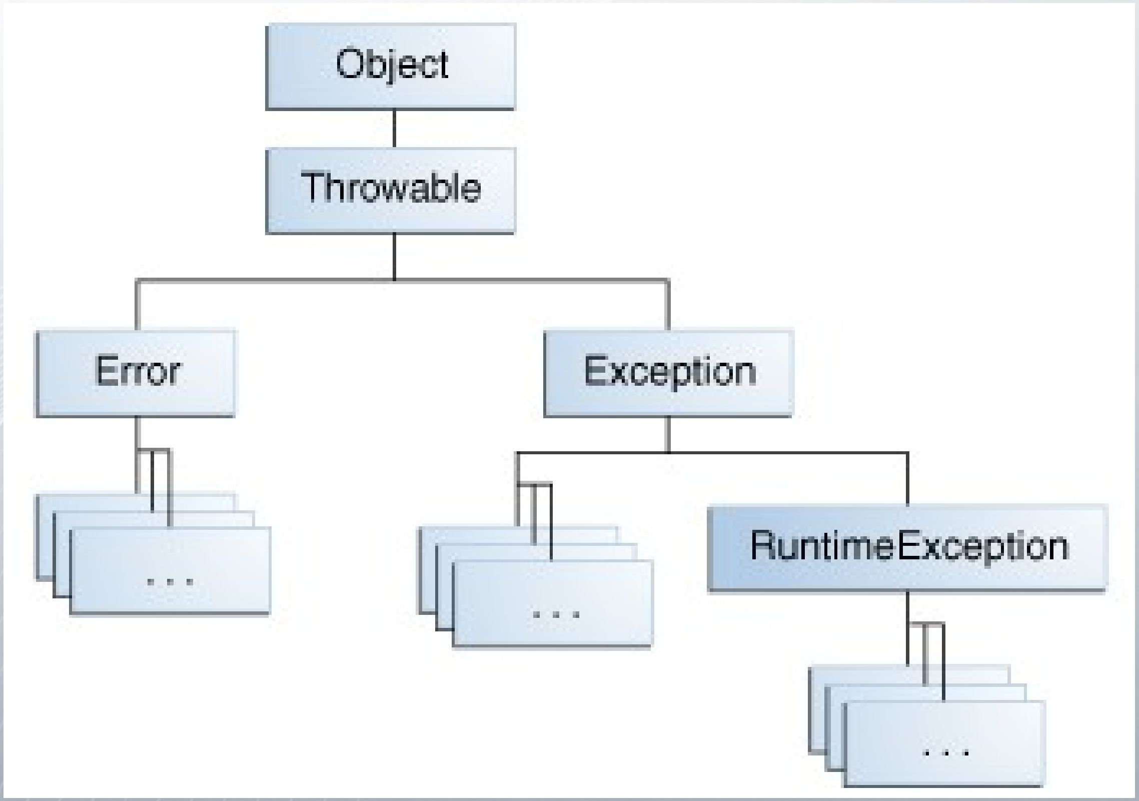
```java
public class ArrayInfoExample {

    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);
    }
}
```

## 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!

## Throwable class and its subclasses

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



## 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
- 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.

## Unchecked exeptions

- All classes which extends 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 extends 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'll talk more about this in the next lesson)
- We can invoke some methods to the exception object like printStackTrace(), getMessage()...(we'll talk more about this in the next lesson)