COM2104: Advanced Programming

LECTURE 2: USER INPUT AND HANDLING EXCEPTIONS

Objectives

- Receive User Input
- Handling Exceptions
- Define special Exceptions
- Documentation (Javadoc)

RECEIVE USER INPUT

THROWING/HANDLING

Java User Input (Scanner)

Before using Scanner class, using import java.util.Scanner.

To use the Scanner class, create an object of the class:

```
Scanner myObj = new Scanner(System.in);
```

You could change the name of the object during the implementation.

https://www.w3schools.com/java/java_user_input.asp

Input types

Method	Description
nextBoolean()	Reads a Boolean value from the user
nextByte()	Reads a byte value from the user
nextDouble()	Reads a double value from the user
nextFloat()	Reads a float value from the user
nextInt()	Reads a int value from the user
nextLine()	Reads a String value from the user
nextLong()	Reads a long value from the user
nextShort()	Reads a short value from the user

Examples about using user input

```java
import java.util.Scanner;  // Import the Scanner class

class Main {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);  // Create a Scanner object
    System.out.println("Enter username");

    String userName = myObj.nextLine();  // Read user input
    System.out.println("Username is: " + userName);  // Output user input
  }
}
```

Output:

```
Enter username
Wen
Username is: Wen
```

Examples about using user input: Cont.

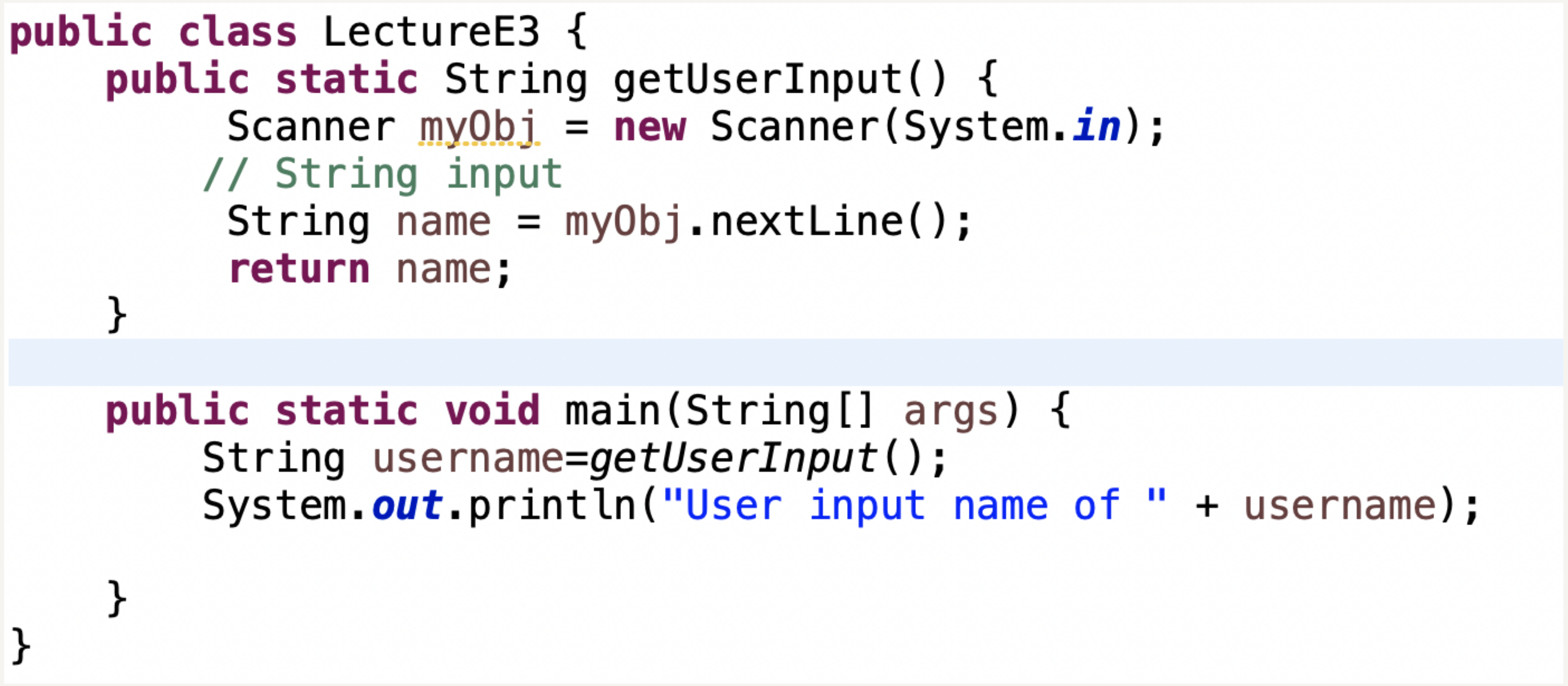
```
import java.util.Scanner;


class Main {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);
    System.out.println("Enter name, age and salary:");
    // String input
    String name = myObj.nextLine();
    // Numerical input
    int age = myObj.nextInt();
    double salary = myObj.nextDouble();
    // Output input by user
    System.out.println("Name: " + name);
```

Output:

```
Enter name, age and salary:
Wen
29
10000
Name: Wen
Age: 29
Salary: 10000
```

Pack the user input in a method

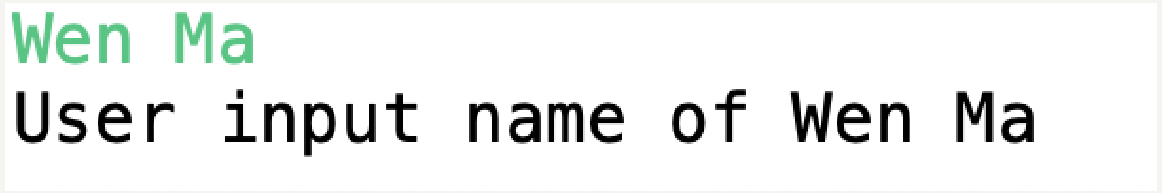


```java
public class LectureE3 {
    public static String getUserInput() {
         Scanner myObj = new Scanner(System.in);
        // String input
         String name = myObj.nextLine();
         return name;
    }

    public static void main(String[] args) {
        String username=getUserInput();
        System.out.println("User input name of " + username);

    }
}
```

Output:



```
Wen Ma
User input name of Wen Ma
```

EXCEPTIONS

What is an exception?

When executing Java code, errors may occur because:

Programmer made errors

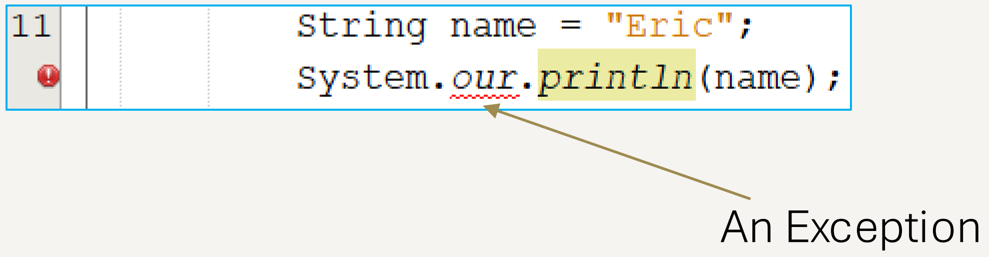
Errors due to wrong input

Other unforeseeable things

- When an error occurs, Java will normally stop and generate an error message. The technical term for this is: Java will throw an **exception** (throw an error).

Handling Exception

- Programmers should handle these exceptions.

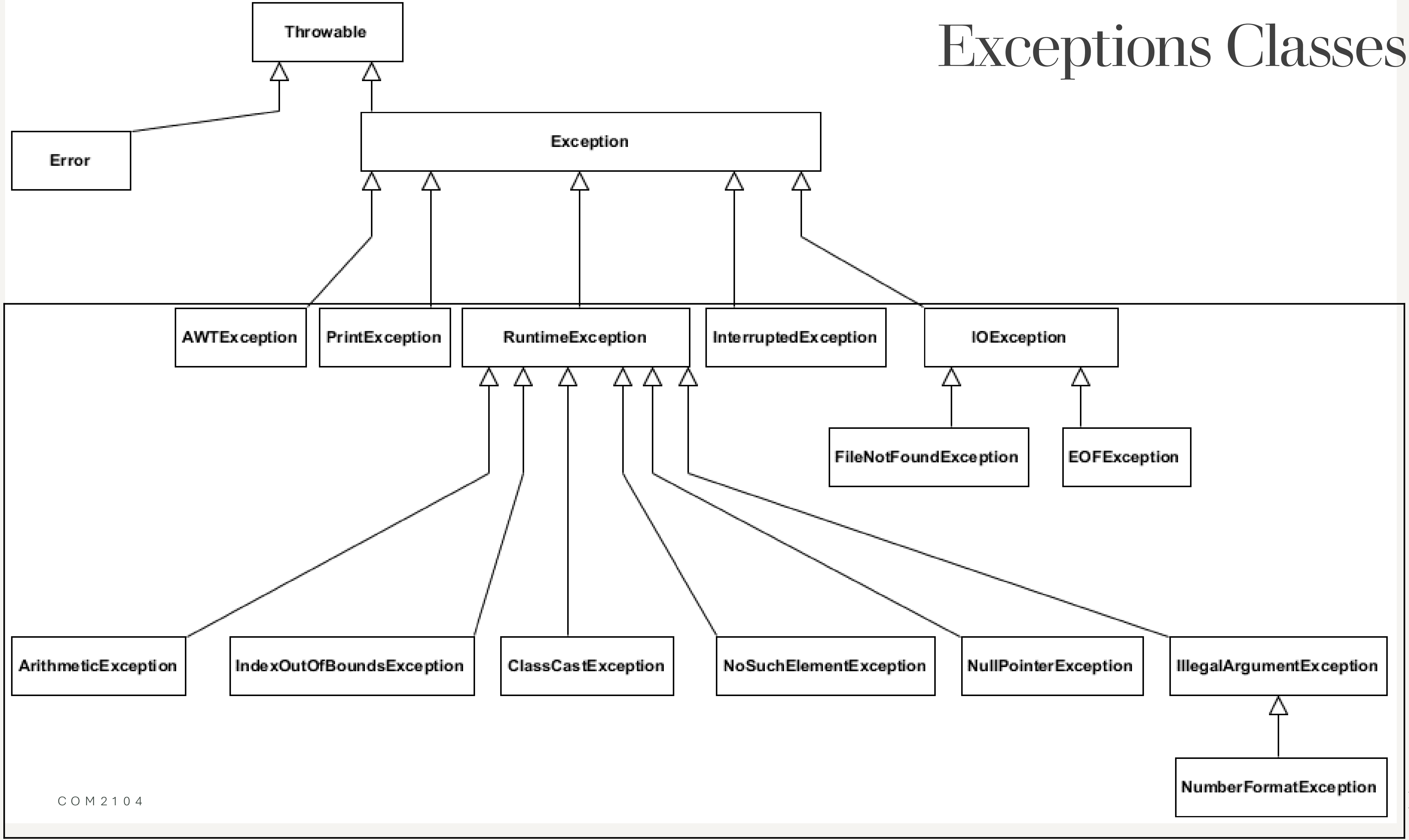


```
String name = "Eric";
System.our.println(name);
```

Exception Handling

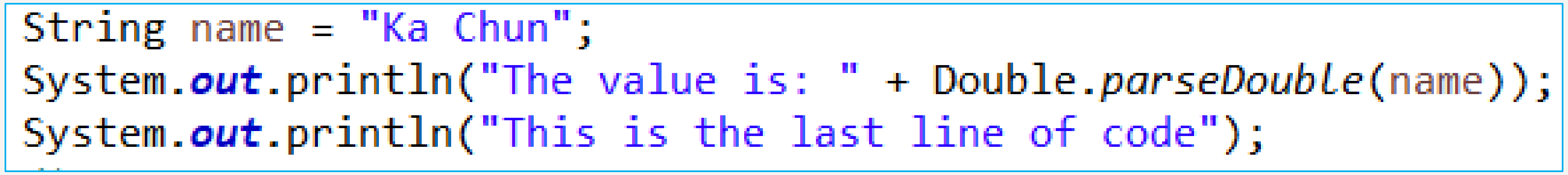
- An exception is an object that is generated as the result of an error or an unexpected event.
- In Java, there are many exception classes corresponding to many exception types.
- Java allows you to create exception handlers.
- The process of intercepting and responding to exceptions is called *exception handling*.

Exceptions Classes

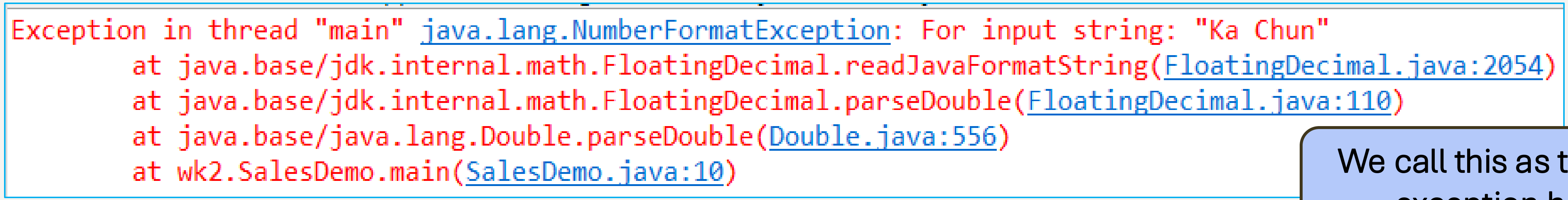


Consequence if we don't handle the exception by our own

The *default exception handler* deals with unhandled exceptions. It prints an error message and crashes the program.

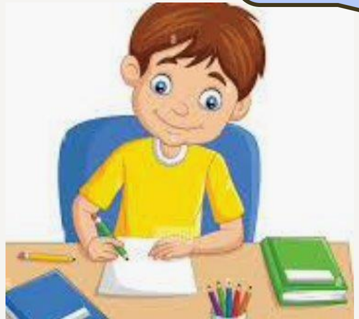


```
String name = "Ka Chun";
System.out.println("The value is: " + Double.parseDouble(name));
System.out.println("This is the last line of code");
```



```
Exception in thread "main" java.lang.NumberFormatException: For input string: "Ka Chun"
	at java.base/jdk.internal.math.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:2054)
	at java.base/jdk.internal.math.FloatingDecimal.parseDouble(FloatingDecimal.java:110)
	at java.base/java.lang.Double.parseDouble(Double.java:556)
	at wk2.SalesDemo.main(SalesDemo.java:10)
```

We call this as the default exception handler



Non-default Exception handler

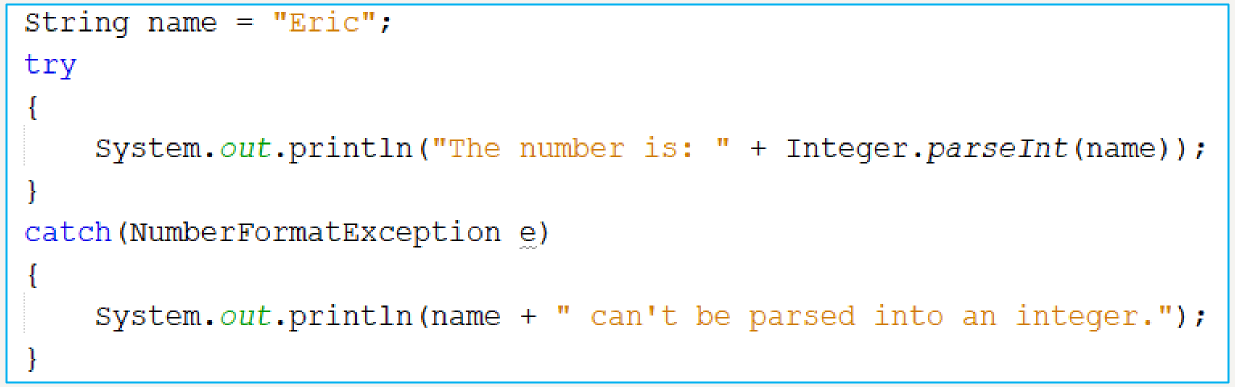
- An non-default exception handler is a section of written code that gracefully responds to exceptions
  - Inform the users which kinds of exception occurred.

Syntax for Non-default Exception handler

- We could use try-catch structure to handle exceptions.
  - Inside the {} of after try, we write the code we want to **execute**.
  - Inside the {} of after catch, we hint the user about the **type of exception** we found.

```
try
{
    try block statements;
                ...
}
catch (ExceptionType ParameterName)
{
    statements;
                ...
}
```

More explanation about try-catch structure



```
String name = "Eric";
try
{
    System.out.println("The number is: " + Integer.parseInt(name));
}
catch(NumberFormatException e)
{
    System.out.println(name + " can't be parsed into an integer.");
}
```

- A *try block* is:
  - one or more statements that are executed, and
  - can potentially throw an exception.
- The program will not **halt** if the try block throws an exception.
- After the try block, a catch clause appears.

Catch Clause

- A **catch clause** begins with the key word **catch**:

```
catch (ExceptionType ParameterName)
```

  - *ExceptionType* is the name of an exception class and
  - *ParameterName* is a variable name which will represent the exception object.

- The code that immediately follows the catch clause is known as a ***catch block*** (the curly braces are required).

- The code in the catch block is **executed if the try block throws an exception**.

Examples about Handling Exceptions

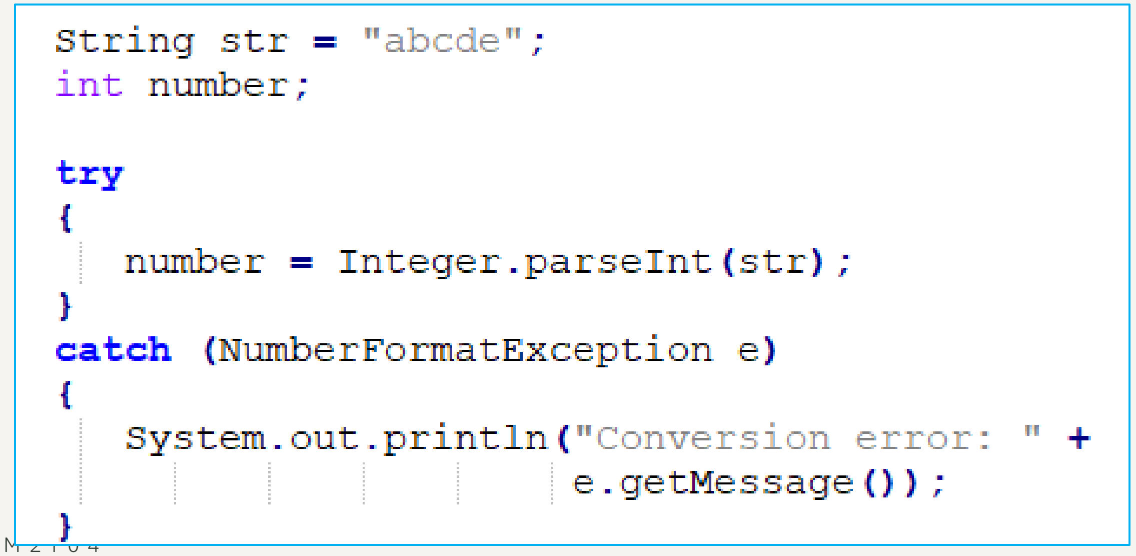
- This code is designed to handle a FileNotFoundException if it is thrown.

```
try
{
   File file = new File ("MyFile.txt");
   Scanner inputFile = new Scanner(file);
}
catch (FileNotFoundException e)
{
   System.out.println("File not found.");
}
```

For the catch clause, the exception type is FileNotFoundException.

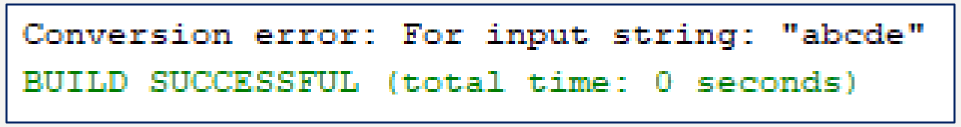
Examples about Handling Exceptions: Cont.

- You should determine which type of exceptions may potentially **occur** during executing **try block**.
- After catch block, your program will continuously be executed.
  - Each exception object has a method named **.getMessage()** that can be used to retrieve the default error message for the exception.



```
String str = "abcde";
int number;

try
{
    number = Integer.parseInt(str);
}
catch (NumberFormatException e)
{
    System.out.println("Conversion error: " +
                            e.getMessage());
}
```



```
Conversion error: For input string: "abcde"
BUILD SUCCESSFUL (total time: 0 seconds)
```

Examples about Handling Exceptions: Cont.

```
public class Main {
  public static void main(String[ ] args) {
    try {
      int[] myNumbers = {1, 2, 3};
      System.out.println(myNumbers[10]);
    } catch (Exception e) {
      System.out.println("Something went wrong.");
    }
```

Output

```
Something went wrong.
```

If we don't know the exact type of the exception, we could use Exception to represent it.

https://www.w3schools.com/java/java_try_catch.asp

The finally Clause

- We could add a finally clause after the try-catch structure.

```
try
{
    (try block statements...)
}
catch (ExceptionType ParameterName)
{
    (catch block statements...)
}
finally
{
    (finally block statements...)
}
```

Finally block

The finally Clause

- The ***finally block*** *contains* one or more statements.
- The statements in the finally block will **be executed** whether an exception occurs or not.

Handling Multiple Exceptions

- The code in the try block may be capable of **throwing more than one type of exception**.
- A catch clause needs to be **written for each type of exception** that could potentially be thrown.
- The JVM* will run the first matched catch clause.
- The catch clauses must be listed **from most specific to most general**.

JVM: Java virtual machine.

Structure for handling multiple exceptions

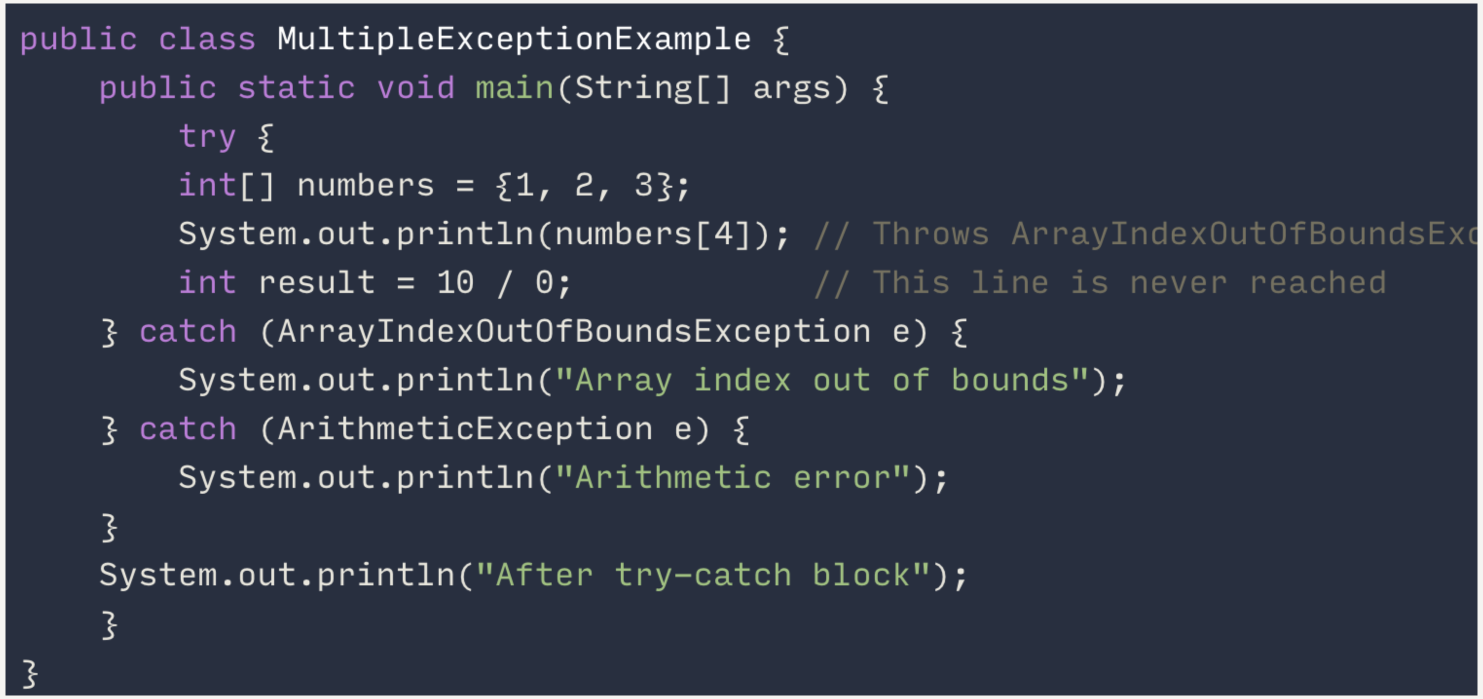
- A try statement may have over one catch clause. The structure is shown in the below.
- We could also add one **finally clause** at the end. But it is optional.

```
try
{
    try block statements;
                   ...
}
catch (ExceptionType1 ParameterName1)
{
    statements;
                   ...
}
catch (ExceptionType2 ParameterName2)
{
    statements;
...
}
```

One example

```
try
{
  number = Integer.parseInt(str);str="Hello"
}
catch (NumberFormatException e) Specific
{
  System.out.println(str + " is not a
number.");
}
catch (IllegalArgumentException e) //OK General
{
  System.out.println("Bad number format.");
}
```

Another Example



```
public class MultipleExceptionExample {
    public static void main(String[] args) {
        try {
        int[] numbers = {1, 2, 3};
        System.out.println(numbers[4]); // Throws ArrayIndexOutOfBoundsExc
        int result = 10 / 0;            // This line is never reached
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("Array index out of bounds");
    } catch (ArithmeticException e) {
        System.out.println("Arithmetic error");
    }
    System.out.println("After try-catch block");
    }
}
```

This is what happens when you run that:

- The `numbers[4]` line throws an ArrayIndexOutOfBoundsException.
- Java immediately jumps to the first matching catch block.
- The ArrayIndexOutOfBoundsException is caught and handled.
- The rest of the try block (i.e. `int result = 10 / 0;`) is skipped.
- Execution continues after all the catch blocks with "After try-catch block".

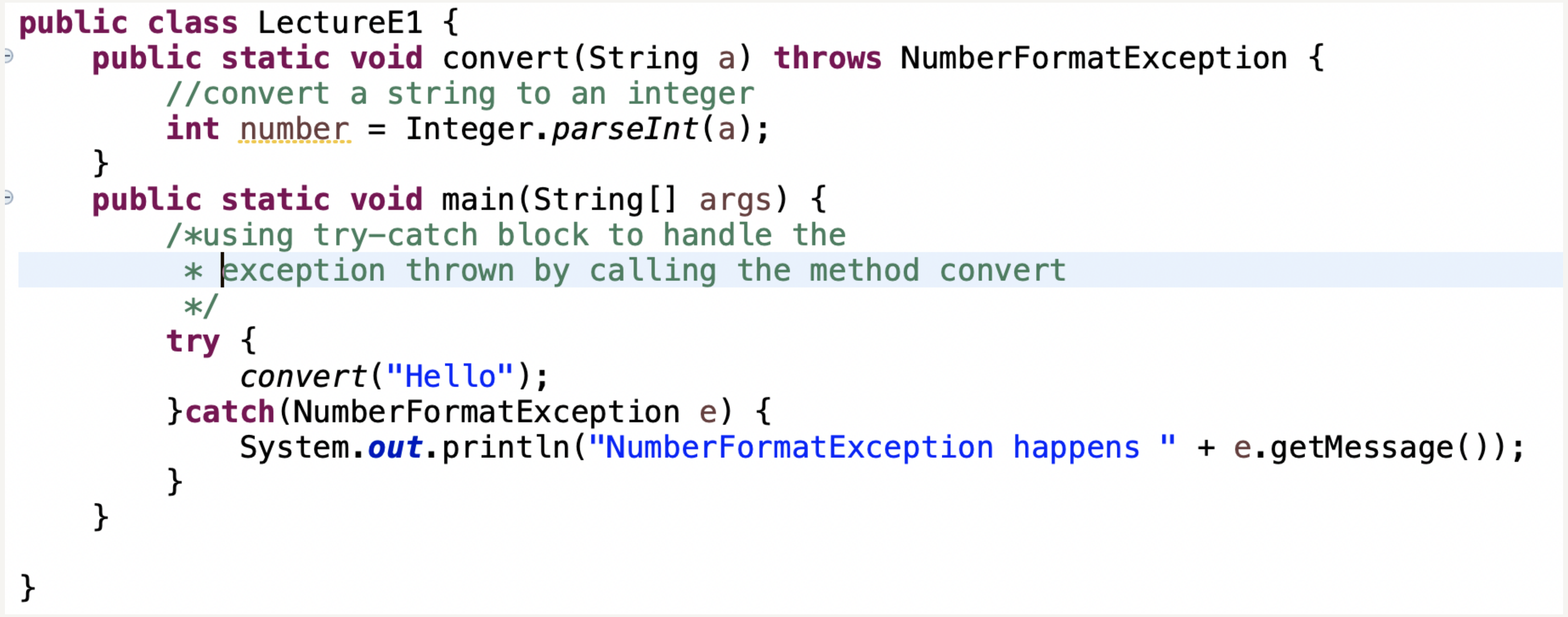
Capture the exceptions from a method

Using throws keyword on the head of a method.

```
type method (arguments) throws Exception1, Exception2, … {   }
```

We could put statements in the try block in each above example to a method. Add using throws keyword to throw the potential exceptions.

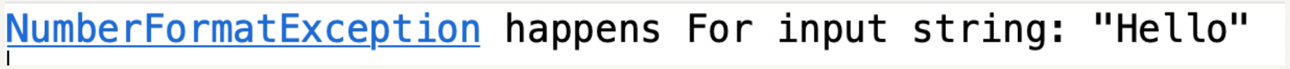
One example



```
public class LectureE1 {
    public static void convert(String a) throws NumberFormatException {
        //convert a string to an integer
        int number = Integer.parseInt(a);
    }
    public static void main(String[] args) {
        /*using try-catch block to handle the
         * exception thrown by calling the method convert
         */
        try {
            convert("Hello");
        }catch(NumberFormatException e) {
            System.out.println("NumberFormatException happens " + e.getMessage());
        }
    }

}
```

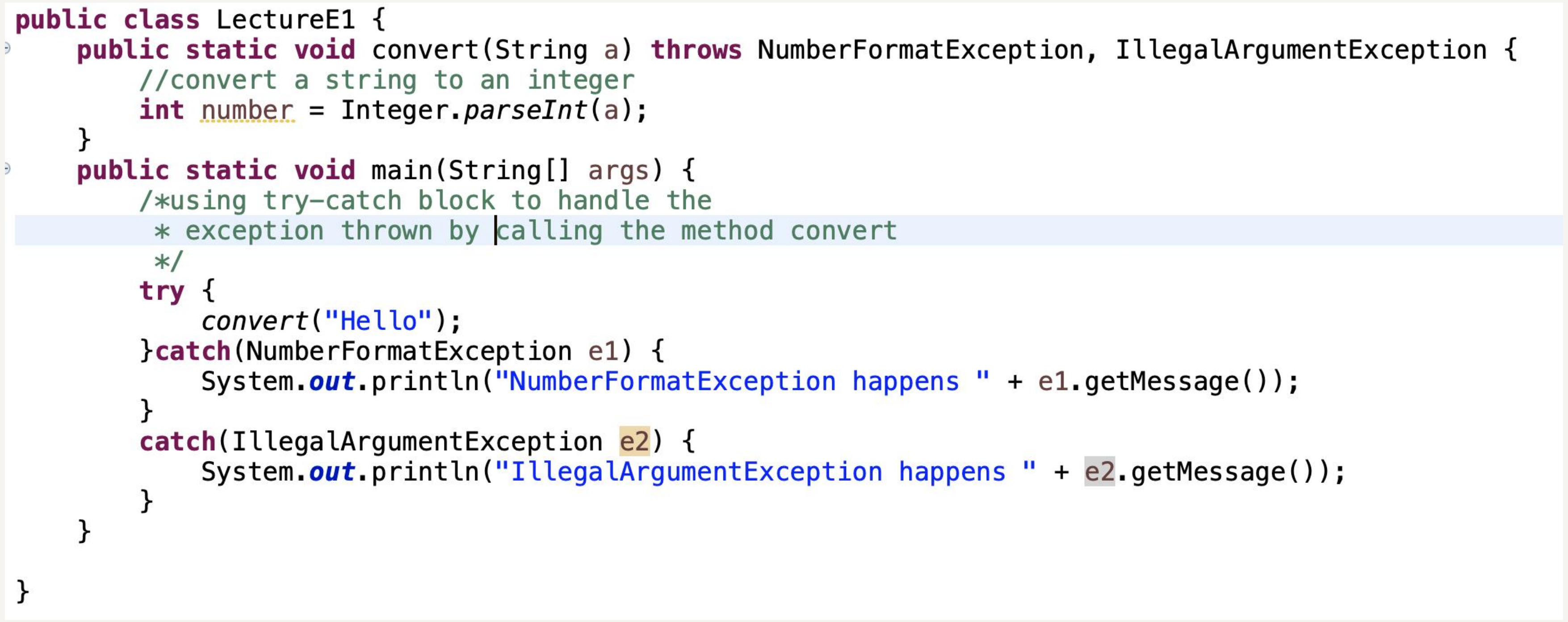
Output



```
NumberFormatException happens For input string: "Hello"
```

Convert(String a) method throws the exception, we use a main function to implement try catch to handle the exception.

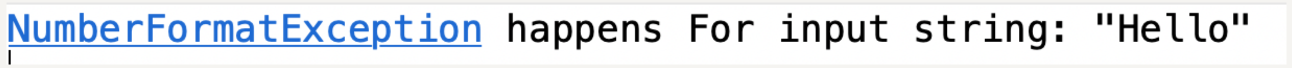
The extended version for the above example



```
public class LectureE1 {
    public static void convert(String a) throws NumberFormatException, IllegalArgumentException {
        //convert a string to an integer
        int number = Integer.parseInt(a);
    }
    public static void main(String[] args) {
        /*using try-catch block to handle the
         * exception thrown by calling the method convert
         */
        try {
            convert("Hello");
        }catch(NumberFormatException e1) {
            System.out.println("NumberFormatException happens " + e1.getMessage());
        }
        catch(IllegalArgumentException e2) {
            System.out.println("IllegalArgumentException happens " + e2.getMessage());
        }
    }

}
```

Output



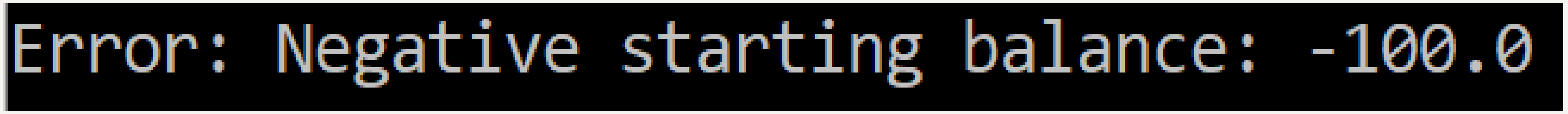
```
NumberFormatException happens For input string: "Hello"
```

DEFINE SPECIAL EXCEPTIONS

Define special exceptions

Considering the following scenarios about a bank account:

- A negative starting balance is passed to the constructor.
- A negative interest rate is passed to the constructor.
- A negative number is passed to the deposit method.
- A negative number is passed to the withdraw method.
- The amount passed to the withdraw method exceeds the account's balance.

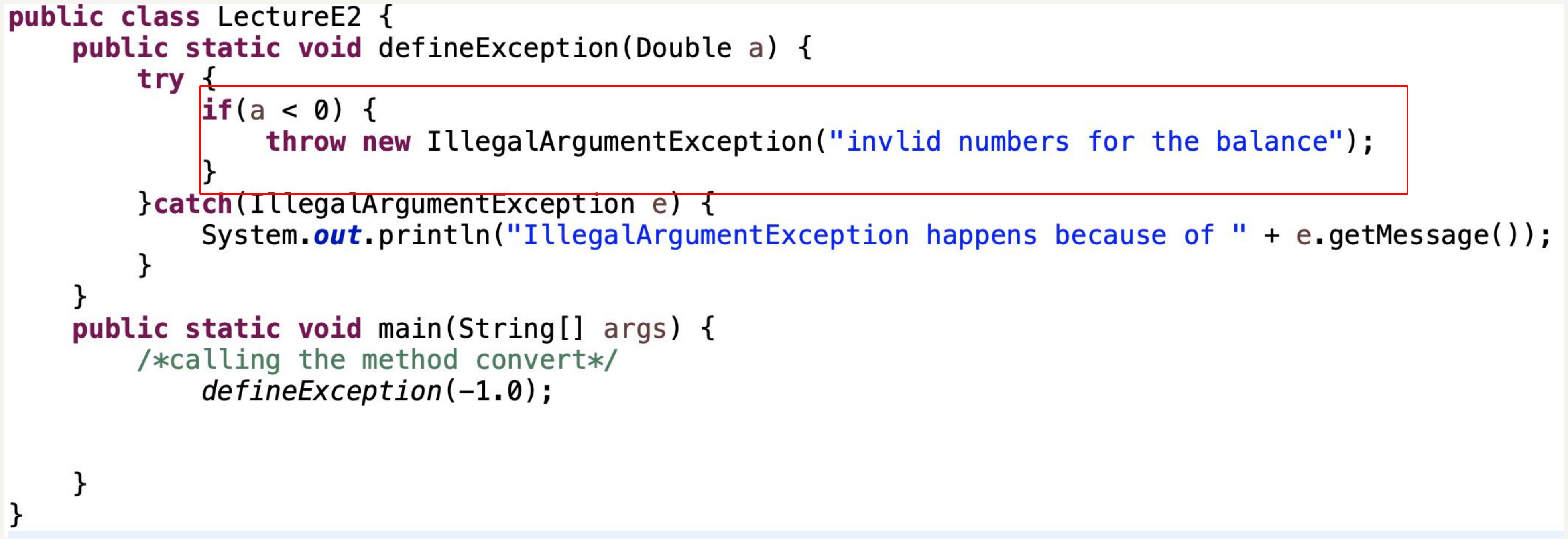


```
Error: Negative starting balance: -100.0
```

Using if statement and throw keyword

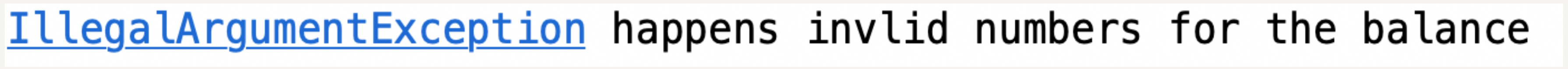
```
try{
        if (value is out of range){
            throw new IllegalArgumentException(some messages);
          }
}catch(IllegalArgumentException e){
        System.out.println("IllegalArgumentException " + e.getMessage());
}
```

One example



```java
public class LectureE2 {
    public static void defineException(Double a) {
        try {
            if(a < 0) {
                throw new IllegalArgumentException("invlid numbers for the balance");
            }
        }catch(IllegalArgumentException e) {
            System.out.println("IllegalArgumentException happens because of " + e.getMessage());
        }
    }
    public static void main(String[] args) {
        /*calling the method convert*/
            defineException(-1.0);


    }
}
```

Output: 

```
IllegalArgumentException happens invlid numbers for the balance
```

JAVADOC (SUPPLEMENTARY)

DOCUMENTATIONS

Javadoc

- Javadoc is a convenient, standard way to document your Java code, allow others to read your code easily.
- Javadoc is a tool for creating HTML documentation from comments

Some notes about Javadoc

- The document comments are embedded inside /**… */
- The first paragraph is a description of the method documented
- Tags help parse your comments:
  - The parameters of the method (@param)
  - What the method returns (@return)
  - Any exceptions the method may throw (@throws)

Javadoc tags

Tag	Parameter	Description
@author	author_name	Describes an author
@param	description	provide information about method parameter or the input it takes
@see	reference	generate a link to other element of the document
@version	version-name	provide version of the class, interface or enum.
@return	description	provide the return value
@exception @throws	description	Describes an exception that may be thrown from this method
@since	Creation year	Indicate the creation year for the code

Javadoc Tag Example

```java
/**
* The HelloWorld program implements an application that
* simply displays "Hello World!" to the standard output.
*
* @author  Zara Ali
* @version 1.0
* @since   2014-03-31
*/
public class HelloWorld {

   public static void main(String[] args) {
      // Prints Hello, World! on standard output.
      System.out.println("Hello World!");
   }
}
```

```java
public class AddNum {
   /**
   * This method is used to add two integers. This is
   * a the simplest form of a class method, just to
   * show the usage of various javadoc Tags.
   * @param numA This is the first paramter to addNum method
   * @param numB  This is the second parameter to addNum method
   * @return int This returns sum of numA and numB.
   */
   public int addNum(int numA, int numB) {
      return numA + numB;
   }

   /**
   * This is the main method which makes use of addNum method.
   * @param args Unused.
   * @return Nothing.
   * @exception IOException On input error.
   * @see IOException
   */
```

https://www.tutorialspoint.com/java/java_documentation.htm

Generate Javadoc

- Netbeans:
  - select Run > Generate Javadoc from the menu bar
  - or, right-click the project in the Projects window and choose Generate Javadoc.
  - The IDE will generate the Javadoc and open it in a separate browser window.
- Eclipse:
  - select Project > Generate Javadoc from the menu bar
  - Choose new destination or copy the default destination
- Command Window (CMD):
  - **`javadoc FileName.java`**

End