Изключения

Лекция 9

AGENDA

- What an exception is
- Some exception terminology
- Why we use exceptions
- How to cause an exception
- How to deal with an exception
- About checked and unchecked exceptions
- Some example Java exceptions
- How to write your own exception

What is an exception?

- An exception or exceptional event is an event that occurs during the execution of a program that disrupts the normal flow of instructions
- The following will cause exceptions:
  - –Accessing an out-of-bounds array element
  - –Writing into a read-only file
  - –Trying to read beyond the end of a file
  - –Sending illegal arguments to a method
  - –Performing illegal arithmetic (e.g. divide by 0)
  - –Hardware failures

Exception Terminology

- When an exception occurs, we say it was thrown or raised
- When an exception is dealt with, we say it is handled or caught
- The block of code that deals with exceptions is known as an exception handler

Няколко понятия:

Действие	Понятие
Грешка по време на изпълнението на програмата	Exception
Генериране на изключение	Throwing
Прихващане на изключение в друга част от програмата	Catching
Програмния код за обработка на изключението	Catchblock
Последователността от ``call statement`` , която завършва с метода където е генерирано изключението	Stack trace

Why use exceptions?

- Compilation cannot find all errors
- To separate error handling code from regular code
  - Code clarity (debugging, teamwork, etc.)
  - Worry about handling error elsewhere
- To separate error detection, reporting, and handling
- To group and differentiate error types –Write error handlers that handle very specific exceptions

Няколко предефинирани изключения:

Exception

- *ClassNotFoundException*
- *IllegalAccessException*
- *InterrupredException*
- *NoSuchMethodException*
- *RuntimeException*
  - *ArithmeticException*
  - *ArrayStoreException*
  - *ClassCastException*
  - *NegativeArraysizeException*
  - *NullPointerException*
  - *SecurityException*
  - *IndexOutOfBoundsException*
  - *String IndexOutOfBoundsException*
  - *Array IndexOutOfBoundsException*
  - *IllegalArgumentException*
  - *NumberFormatException*
  - *IllegalThreadStateException*

Генериране на изключение(Throwing)
Нека обекта '*q*' да не е още инициализиран. Този факт може да се провери преди използването на обекта и обработката на ситуацията да се остави на друг контекст на програмата:

```
if( q = = null)
    throw new NullPointerException();
```

Възможно е генерирането на изключение посредством конструктор с един аргумент(низ от символи):

```
if(q == null)
    throw new NullPointerException("q = null");
```

Всички изключения имат по два конструктора - първият е подразбиращият се конструктор(без аргументи), а вторият е с един аргумент - низ от символи, който може да бъде анализиран в кода за обработка на изключението (exception handler).

Decoding Exception Messages

```
public class ArrayExceptionExample {
    public static void main(String args[]) {
        String[] names = {“Bilha", “Robert"};
        System.out.println(names[2]);
    }
}
```

- The println in the above code causes an exception to be thrown with the following exception message:

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 2 at
  ArrayExceptionExample.main(ArrayExceptionExampl
  e.java:4)
```

Exception Message Format

- Exception messages have the following format:

[exception class]: [additional description of exception] at [class].[method]([file]:[line number])

Exception message from array example:

java.lang.ArrayIndexOutOfBoundsException: 2 at ArrayExceptionExample.main(ArrayExceptionExampl e.java:4)

- What is the exception class?

  java.lang.ArrayIndexOutOfBoundsException

- Which array index is out of bounds?

  2

- What method throws the exception?

  ArrayExceptionExample.main

- What file contains the method?

  ArrayExceptionExample.java

- What line of the file throws the exception?

  4

Throwing Exceptions

- All methods use the throw statement to throw an exception
  - if (student.equals(null))
    throw newNullPointerException();
- The throw statement requires a single argument: a throwable object
- Throwable objects are instances of any subclass of the Throwable class
  - Include all types of errors and exceptions
  - Check the API for a full listing of throwable objects

Обработка на изключенията(Catching)

В програмата могат да се въведат "проследявани" блокове:

```
try {
      //опасен проследяван код, който може да предизвика изключение
}
catch(type1 id1) { // може да има нула или повече "catch" блокове
      //обработва изключения от тип "type1" в проследявания блок
}
catch(type2 id2) {
      //обработва изключения от тип "type2"
}…
finally { //може да има нула или повече  "finally" блокове
      //изпълнява се винаги, независимо дали има изключение или не
}
```

Handling Exceptions

- You can use a try-catch block to handle exceptions that are thrown

```
try {
    // code that might throw exception
}
catch ([Type of Exception] e) {
  // what to do if exception is thrown
}
```

Handling Multiple Exceptions

- You can handle multiple possible exceptions by multiple successive catch blocks

```
try {
    // code that might throw multiple
   // exceptions
}
catch (IOException e) {
   // handle IOException
}
catch (ClassNotFoundException e2)  {
// handle ClassNotFoundException
}
```

Finally Block

- You can also use the optional finally block at the end of the try-catch block
- The finally block provides a mechanism to clean up regardless of what happens within the try block
  - Can be used to close files or to release other system resources

Try-Catch-Finally Block

```
try {
    // code that might throw exception
}
catch ([Type of Exception] e) {
  // what to do if exception is thrown
}
finally {
   // statements here always get
  // executed, regardless of what
  // happens in the try block
}
```

Unchecked Exceptions

- Unchecked exceptions or runtime exceptions occur within the Java runtime system
- Examples of unchecked exceptions

  –arithmetic exceptions (dividing by zero)

  –pointer exceptions (trying to access an object's members through a null reference)

  –indexing exceptions (trying to access an array element with an index that is too large or too small)
- A method does not have to catch or specify that it throws unchecked exceptions, although it may

Checked Exceptions

- Checked exceptions or non runtime exceptions are exceptions that occur in code outside of the Java runtime system
- For example, exceptions that occur during I/O (covered next lecture) are nonruntime exceptions
- The compiler ensures that nonruntime exceptions are caught or are specified to be thrown (using the throws keyword)

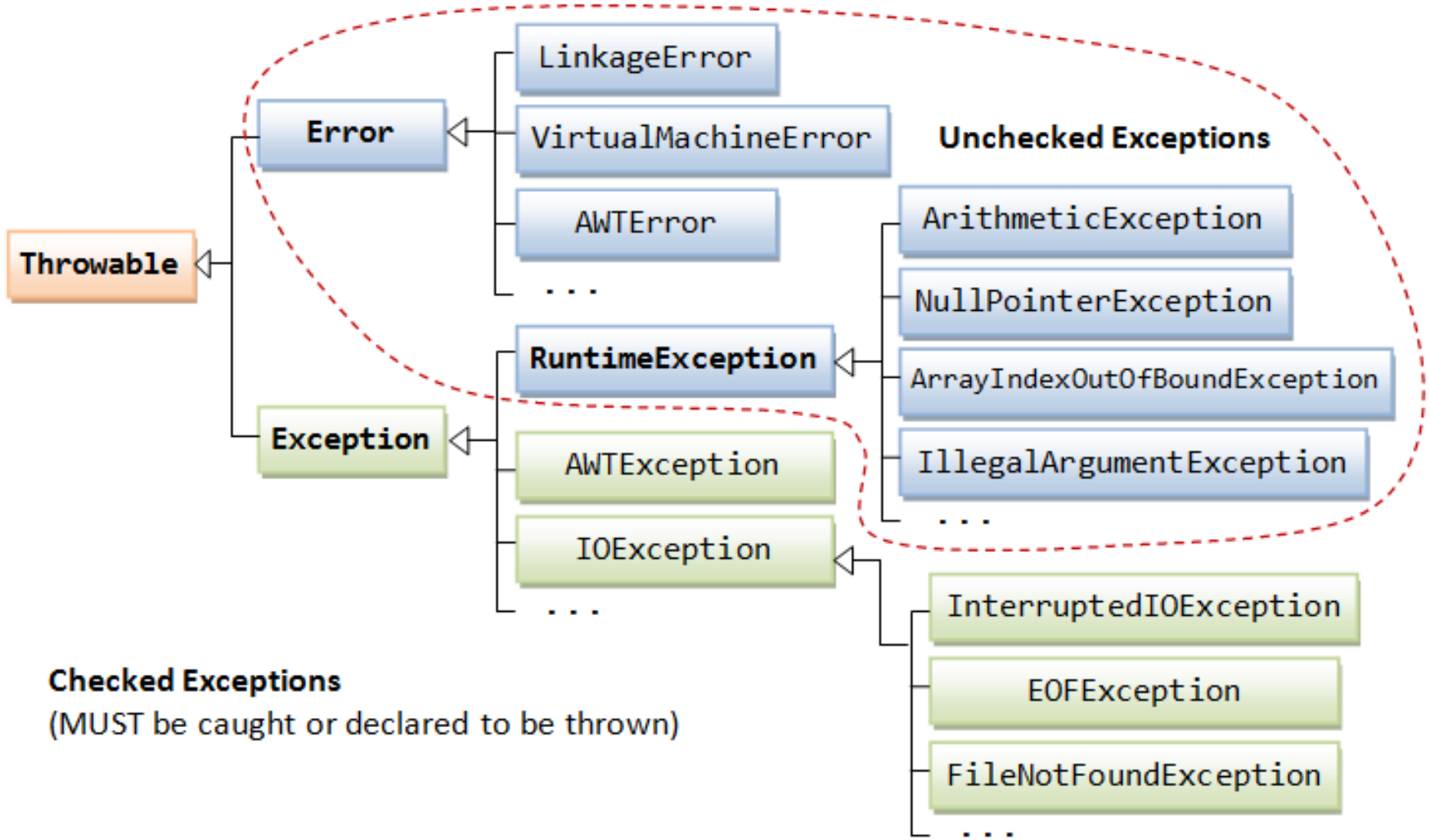
Handling Checked Exceptions

- Every method must catch checked exceptions OR specify that it may throw them (using the throws keyword)

```
void readFile(String filename) {
   try {
       FileReader reader = new
              FileReader("myfile.txt");
       // read from file . . .
    } catch (FileNotFoundException e) {
       System.out.println("file was not found");
    }
}
```

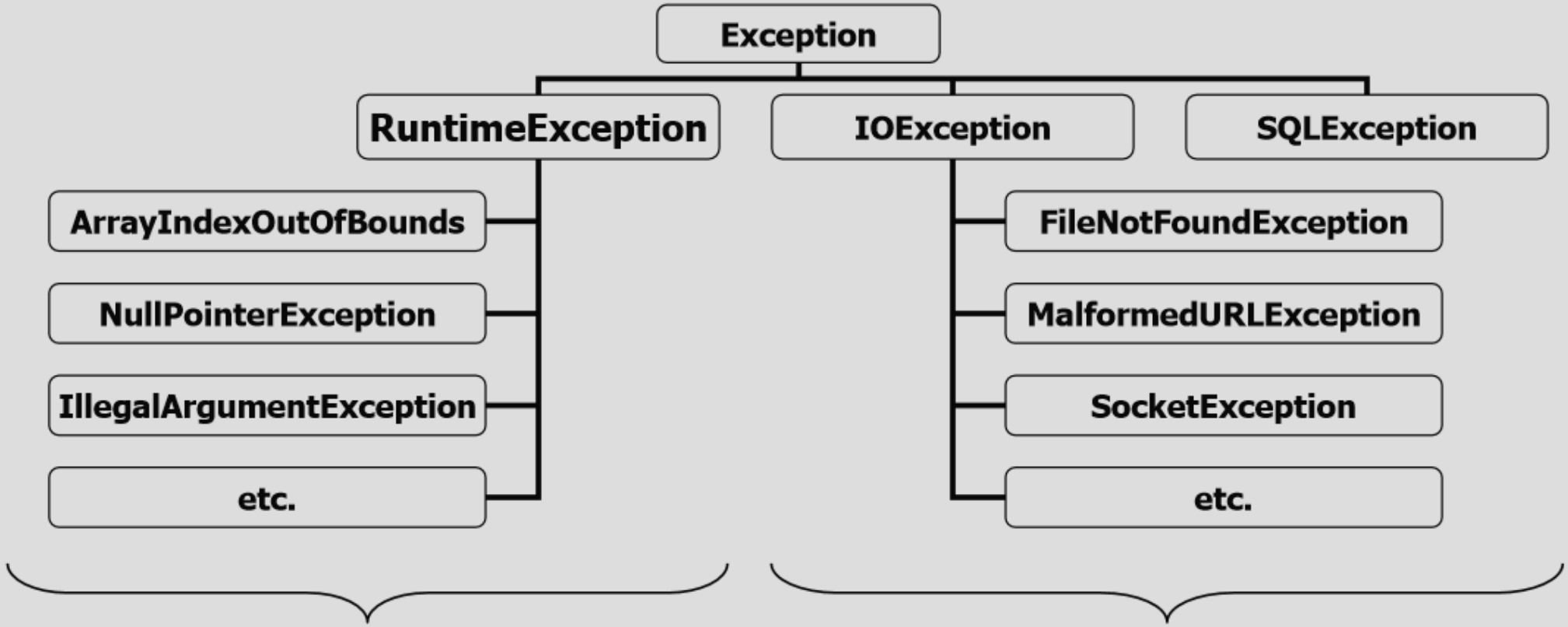
OR

```
void readFile(String filename) throws
   FileNotFoundException {
     FileReader reader = new FileReader("myfile.txt");
     // read from file . . .
}
```



Checked Exceptions
(MUST be caught or declared to be thrown)

Exception Class Hierarchy



Unchecked Exceptions

Checked Exceptions

- Look in the Java API for a full list of exceptions

Exceptions and Inheritance

- A method can throw less exceptions, but not more, than the method it is overriding

```
public class MyClass {
   public void doSomething()
               throws IOException, SQLException {
       // do something here
   }
}

public class MySubclass extends MyClass {
   public void doSomething() throws IOException {
       // do something here
   }
}
```

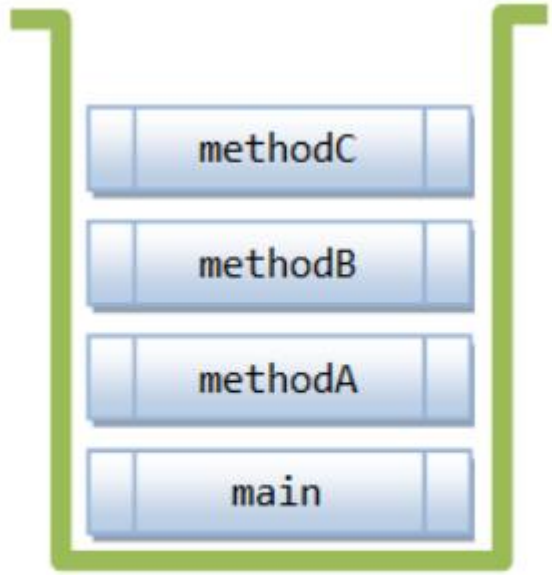
```java
public class MethodCallStackDemo {
   public static void main(String[] args) {
      System.out.println("Enter main()");
      methodA();
      System.out.println("Exit main()");
   }

   public static void methodA() {
      System.out.println("Enter methodA()");
      methodB();
      System.out.println("Exit methodA()");
   }

   public static void methodB() {
      System.out.println("Enter methodB()");
      methodC();
      System.out.println("Exit methodB()");
   }

   public static void methodC() {
      System.out.println("Enter methodC()");
      System.out.println("Exit methodC()");
   }
}
```

```
Enter main()
Enter methodA()
Enter methodB()
Enter methodC()
Exit methodC()
Exit methodB()
Exit methodA()
Exit main()
```



Method Call Stack
(Last-in-First-out Queue)

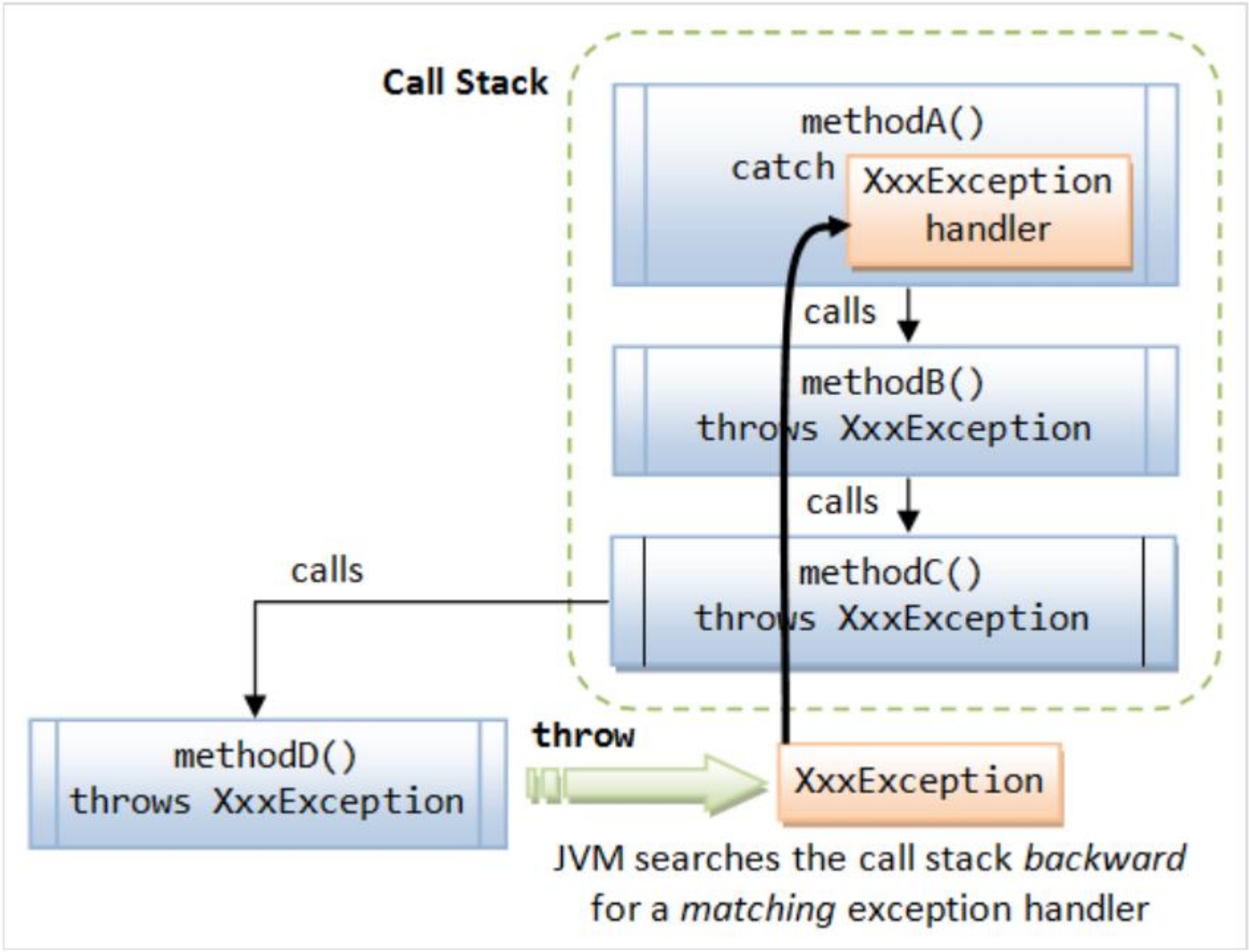
Нека да променим `methodC()`, така че операцията "divide-by-0" ,
ще извика `ArithmeticException`:

```
public static void methodC() {
   System.out.println("Enter methodC()");
   System.out.println(1 / 0);  // divide-by-0 triggers an ArithmeticException
   System.out.println("Exit methodC()");
}
```

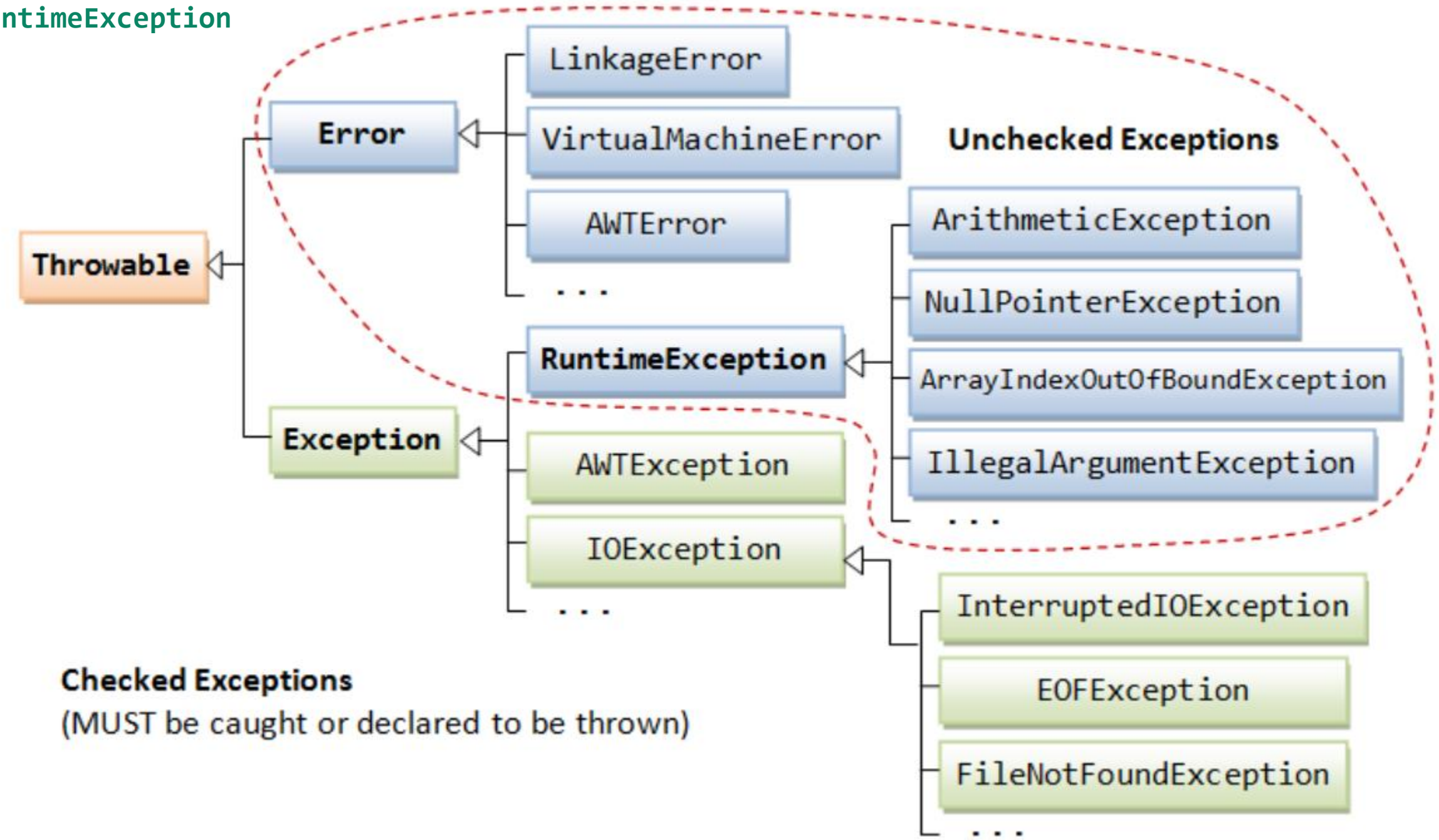
Резултатът от изпълнението:

```
Enter main()
Enter methodA()
Enter methodB()
Enter methodC()
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at MethodCallStackDemo.methodC(MethodCallStackDemo.java:22)
        at MethodCallStackDemo.methodB(MethodCallStackDemo.java:16)
        at MethodCallStackDemo.methodA(MethodCallStackDemo.java:10)
        at MethodCallStackDemo.main(MethodCallStackDemo.java:4)
```

Exception & Call Stack



Exception Classes - Throwable, Error, Exception & RuntimeException



Writing Your Own Exceptions

- There are at least 2 types of exception constructors:
  - Default constructor: No arguments

```
NullPointerException e = new
    NullPointerException();
```

  - Constructor that has a detailed message: Has a single `String` argument

```
IllegalArgumentExceptione e =
    new IllegalArgumentException("Number must
    be positive");
```

Checked or Unchecked?

- If a user can reasonably be expected to recover from an exception, make it a checked exception
- If a user cannot do anything to recover from the exception, make it an unchecked exception

Checked or Unchecked?

- If a user can reasonably be expected to recover from an exception, make it a checked exception
- If a user cannot do anything to recover from the exception, make it an unchecked exception

Overriding and Overloading Methods

An overriding method must have the same argument list and return-type (or subclass of its original from JDK 1.5). An overloading method must have different argument list, but it can have any return-type.

An overriding method cannot have more restricted access. For example, a method with `protected` access may be overridden to have protected or public access but not `private` or default access. This is because an overridden method is considered to be a replacement of its original, hence, it cannot be more restrictive.

An overriding method cannot declare exception types that were not declared in its original. However, it may declare exception types are the same as, or subclass of its original. It needs not declare all the exceptions as its original. It can throw fewer exceptions than the original, but not more.

An overloading method must be differentiated by its argument list. It cannot be differentiated by the return-type, the exceptions, and the modifier, which is illegal. It can have any return-type, access modifier, and exceptions, as long as it can be differentiated by the argument list.

`ArrayIndexOutOfBoundsException`: thrown by JVM when your code uses an array index, which is is outside the array's bounds. For example,

```
int[] anArray = new int[3];
System.out.println(anArray[3]);
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
```

`NullPointerException`: thrown by the JVM when your code attempts to use a null reference where an object reference is required. For example,

```
String[] strs = new String[3];
System.out.println(strs[0].length());
```

```
Exception in thread "main" java.lang.NullPointerException
```

`NumberFormatException`: Thrown programmatically (e.g., by `Integer.parseInt()`) when an attempt is made to convert a string to a numeric type, but the appropriate format. For example,

```
Integer.parseInt("abc");
```

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "abc"
```

`ClassCastException`: thrown by JVM when an attempt is made to cast an object reference fails. For example,

```
Object o = new Object();
Integer i = (Integer)o;
```

```
Exception in thread "main" java.lang.ClassCastException: java.lang.Object cannot be cast to java.lang.Integer
```

- **IllegalArgumentException**: thrown programmatically to indicate that a method has been passed an illegal or inappropriate argument. You could re-use this exception for your own methods.
- **IllegalStateException**: thrown programmatically when a method is invoked and the program is not in an appropriate state for that method to perform its task. This typically happens when a method is invoked out of sequence, or perhaps a method is only allowed to be invoked once and an attempt is made to invoke it again.
- **NoClassDefFoundError**: thrown by the JVM or class loader when the definition of a class cannot be found. Prior to JDK 1.7, you will see this exception call stack trace if you try to run a non-existent class. JDK 1.7 simplifies the error message to "Error: Could not find or load main class xxx".

Assertion (JDK 1.4)

```
1  public class AssertionSwitchTest {
2     public static void main(String[] args) {
3        char operator = '%';                    // assumed either '+', '-', '*', '/' only
4        int operand1 = 5, operand2 = 6, result = 0;
5        switch (operator) {
6           case '+': result = operand1 + operand2; break;
7           case '-': result = operand1 - operand2; break;
8           case '*': result = operand1 * operand2; break;
9           case '/': result = operand1 / operand2; break;
10          default: assert false : "Unknown operator: " + operator;  // not plausible here
11       }
12       System.out.println(operand1 + " " + operator + " " + operand2 + " = " + result);
13    }
14 }
```

Assertion can be used for verifying:

```
1  public class AssertionTest {
2     public static void main(String[] args) {
3        int number = -5;    // assumed number is not negative
4        // This assert also serve as documentation
5        assert (number >= 0) : "number is negative: " + number;
6        // do something
7        System.out.println("The number is " + number);
8     }
9  }
```

```
> java -ea AssertionSwitchTest     // enable assertion
```

```
Exception in thread "main" java.lang.AssertionError: -5
        at AssertionTest.main(AssertionTest.java:5)
```

Pre-conditions of public methods

```
// Constructor of Time class
public Time(int hour, int minute, int second) {
   if(hour < 0 || hour > 23 || minute < 0 || minute > 59 || second < 0 || second > 59) {
      throw new IllegalArgumentException();
   }
   this.hour = hour;
   this.minute = minute;
   this.second = second;
}
```

Lecture Summary

- Exceptions disrupt the normal flow of the instructions in the program
- Exceptions are handled using a try-catch or a try-catch-finally block
- A method throws an exception using the throw statement
- A method does not have to catch or specify that it throws unchecked exceptions, although it may

Lecture Summary

- Every method must catch checked exceptions or specify that it may throw them
- If you write your own exception, it must be a subclass of the Exception class and have at least the two standard constructors