Java Object Oriented Programming

Exceptions

Hiba ALQASIR

2021-2022

Exceptions

What is an **Exception**?

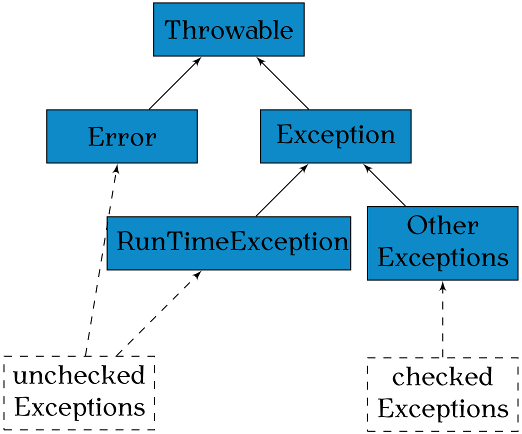
- An object thrown at runtime

When an **Exception** is thrown?

- An unexpected event occurs and disrupts the normal flow of the program.

Types of Exception

- Checked Exception: checked at **compile-time**.
  E.g.: IOException.
- Unchecked Exception: checked at **run-time**.
  E.g.: NullPointerException.
- Error: irrecoverable
  E.g. : OutOfMemoryError.



Exception Handling

- Mechanism for dealing with checked exceptions.
- Maintains the normal flow of the program.

Exception Handling

What we can do?

1. Catch the exception using **try ... catch** block
2. Propagate the exception using **throws** keyword

Java try block

- Java **try** block is used to enclose the code that might throw an exception.
- It must be followed by either catch or finally block.

```java
try{
    // code that may throw exception ** protected code **
}catch(Exception_class_name exception_object_name){
    // catch block
}
```

```java
try{
    // code that may throw exception ** protected code **
}finally{
    // finally block ** always executes **
}
```

Java catch block

- Java **catch** block is used to handle the Exception.
- It must be used after the try block only.
- We can use multiple **catch** block with a single try.

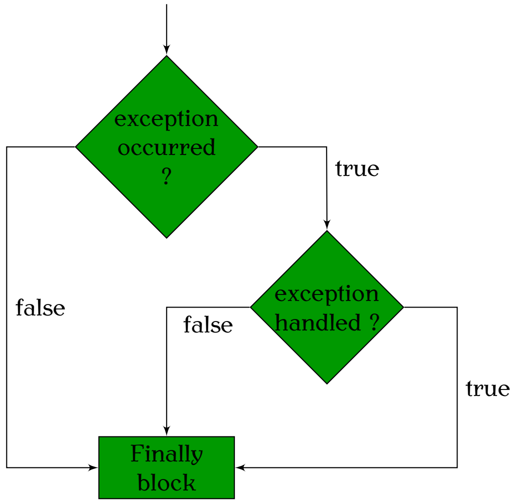
```java
try{
    // code that may throw exception ** protected code **
}catch(Exception_class1_name exception_object1_name){
    // catch block1
}catch(Exception_class2_name exception_object2_name){
    // catch block2
}
```

Java finally block

- Java **finally** block is used to execute important code.
- It is always executed whether exception is handled or not.
- It follows try or catch block.

```
try{
    // code that may throw exception
}finally{
    // finally block ** always executes **
}
```

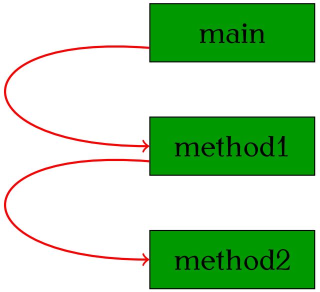
Java finally block



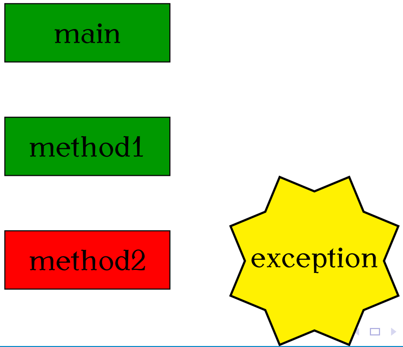
Exception propagation

1. Something went wrong!
2. An object of class Exception is created inside the method.
3. Does the method provide a treatment of this exception?
   a. Yes: the exception is processed and the program resumes after processing the exception.
   b. No: the exception is returned to the method that called the current method, we return to point 3

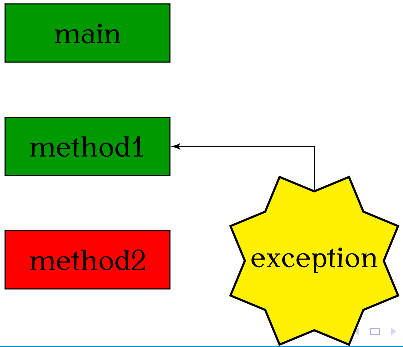
Exception propagation



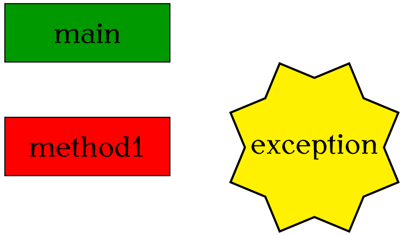
Exception propagation



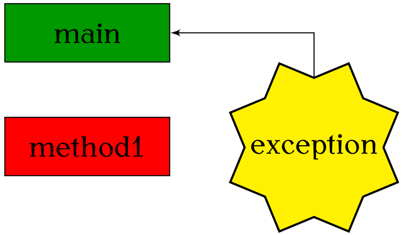
Exception propagation



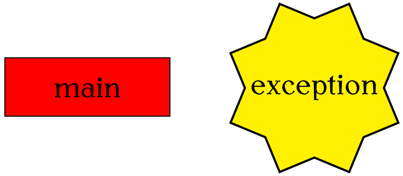
Exception propagation



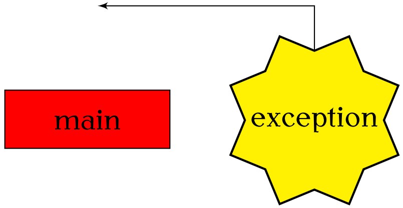
Exception propagation



Exception propagation



Exception propagation



Java **throws** keyword

- If a method does not handle a checked exception, the method must declare it using the **throws** keyword.

Syntax

```
void method_name() throws Exception_class_name {
      // Method implementation
}
```

Example

```
void printDay(int[] days, int id) throws ArrayIndexOutOfBoundsException{
    try{
        System.out.println(days[id]);
    }finally{
        System.out.println("finally!");
    }
}
```

Java **throw** keyword

- The Java **throw** keyword is used to explicitly throw an exception.
- We can use it to throw either checked or uncheked exception.

Syntax

```java
void method_name() throws Exception_class_name {
      // Method implementation
      throw new Exception_class_name();
}
```

Example

```java
void adultsOnly(int age) throwsArithmeticException{
    if(age>18) {
        System.out.println("This person is > 18");
    } else {
        throw new ArithmeticException("This person is < 18");
    }
}
```

throw vs. throws

throw	**throws**
To throw an exception explicitly	To declare an exception
The checked exceptions cannot be propagated with throw only	The checked exceptions can be propagated with throws
Followed by an instance	Followed by class.
Used within the method	Used with the method signature
You cannot throw multiple exceptions	You can declare multiple exceptions

Exception Classes

Class	**Description**
ClassCastException	Thrown to indicate that the code has attempted to cast an object to a subclass of which it is not an instance.
FileNotFoundException	Signals that an attempt to open the file denoted by a specified pathname has failed.
IndexOutOfBoundsException	Thrown to indicate that an index of some sort is out of range.
IOException	Signals that an I/O exception of some sort has occurred.
NullPointerException	Thrown when an application attempts to use null in a case where an object is required.

Java Custom Exception

You can create your own exceptions in Java.

- It must be a child of java.lang.Throwable class.
- A checked exception extends the Exception class.
- A runtime exception extends the RuntimeException class.

```java
class MyException extends Exception {
    // ...
}
```

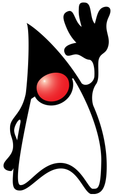
Exercise #1

What exception types can be caught by the following handler?

```java
try {
    // try block
} catch (Exception e) {
    // catch block
}
```

What is wrong with using this type of exception handler?

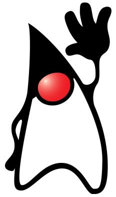
```java
try {
    // try block
} catch (Exception e) {
    // catch block1
} catch (ArithmeticException a) {
    // catch block2
}
```



Exercise #2

Modify the following *cat* method so that it will compile.

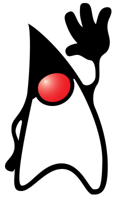
```java
public static void cat(File file) {
    RandomAccessFile input = null;
    String line = null;

    try {
        input = new RandomAccessFile(file, "r");
        while ((line = input.readLine()) != null) {
            System.out.println(line);
        }
        return;
    } finally {
        if (input != null) {
            input.close();
        }
    }
}
```



Exercise #3

What is the output of the following method, if the parameter $a = 9$?

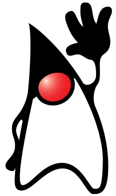
```java
public void add(int a) {
    a = a + 10;
    try {
        a = a + 10;
        try {
            a = a * 10;
            throw new Exception();
        } catch (Exception e) {
            a = a - 10;
        }
    } catch (Exception e) {
        a = a - 10;
    }
    System.out.println("a = " + a);
}
```



Exercise #4

What is the output of the following Java program?

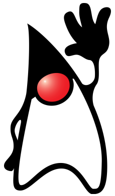
```java
public class Main {
    void method1() {
        try {
            System.out.println("method1() called");
            method2();
            System.out.println("Was there un exception?");
        } catch (Exception e) {
            System.out.println("exception is caught");
        } finally {
            System.out.println("finally block is called");
        }
    }
    void method2() throws Exception {
        System.out.println("method2() called");
        throw new Exception();
    }
    public static void main(String args[]) {
        new Main().method1();
    }
}
```



Exercise #4

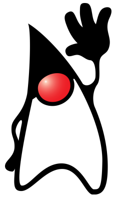
And now?

```java
public class Main {
    void method1() {
        try {
            System.out.println("method1() called");
            method2();
            System.out.println("Was there un exception?");
        } catch (Exception e) {
            System.out.println("exception is caught");
        } finally {
            System.out.println("finally block is called");
        }
    }
    void method2(){
        System.out.println("method2() called");
    }
    public static void main(String args[]) {
        new Main().method1();
    }
}
```



Exercise #5

Write a method to read float values from a file, print each value and add them to the end of an array. First value in the file is the number of elements. You must detect any errors that might occur.



Exercise #6

1. Create your own exception class. Write a constructor for this class that takes a String argument and stores it. Write a method that prints out the stored String.
2. Create a class with two methods. In the first method, throw an exception of the new type that you defined in .1. In the second, call the first one, catch its exception and, in the catch clause, throw a RuntimeException.
3. Test your code in the main method.