Exceptions

Content

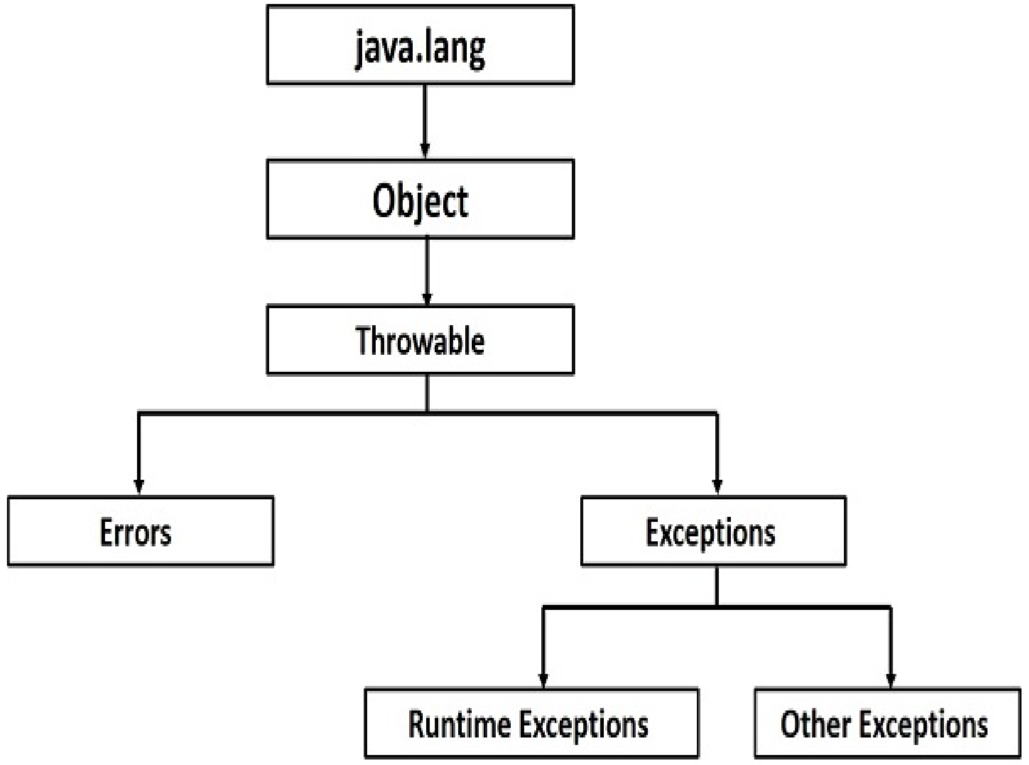
- Exceptions Types
- Exceptions Keywords
- Exceptions Handling
- Customization Exceptions

Types

Exceptions

Hierarchy

Types



Exceptions

Types

Types

RuntimeException (unchecked)

- any exception that extends RuntimeException
- counted as bugs and must be fixed to complete app
- unchecked by the compiler – developer responsibility

Compilation Exceptions (checked)

- any exception that doesn't extend RuntimeException
- user defined exceptions
- are NOT bugs !! And therefore checked by the compiler

Errors : Serious problems that user /programmer is not responsible about it and they shouldn't handle it.

Exceptions

Types

Runtime Exceptions

- ArithmeticException
- NullPointerException
- NegativeArraySizeException
- ArrayIndexOutOfBoundsException
- SecurityException
- NumberFormatException
- ClassCastException

Exceptions

Compile Exceptions

Types

- IOException [EOFException, FileNotFoundException...]
- SQLException
- DOMException, SAXException
- ClassNotFoundException
- RemoteException
- AWTException

Exceptions

Types

Errors

An Error is a subclass of Throwable that indicates serious problems that a reasonable application should not try to catch

VirtualMachineError

- Assertation Error
- IncompatibleClassChangeError
- ExceptionInitializerError
- OutOfMemoryError
- StackOverFlowError
- Internal Error

Exceptions

Keywords

Keywords

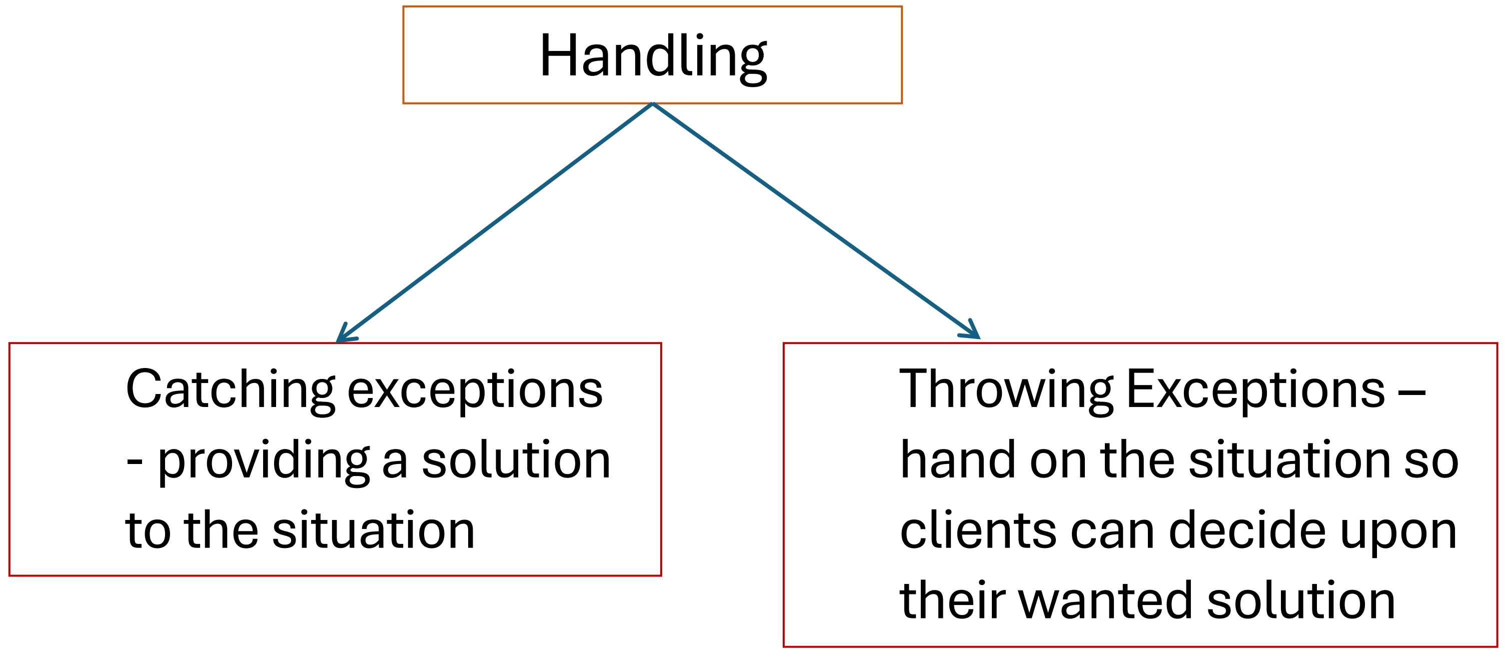
Keyword	Description
try	Used to specify a block where we should place exception code. **must be followed by either catch or finally**.
catch	Used to handle the exception. It must be preceded by try block which means **we can't use catch block alone**.
finally	Used to execute the important code of the program. It is executed whether an exception is handled or not.
throw	used to throw an exception.
throws	used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

Exception

Handling Exceptions

- Compilation Exception must be handled (caught or thrown)
- Runtime Exception may be handled (caught or thrown)

Handling


Handling
Catching exceptions - providing a solution to the situation
Throwing Exceptions – hand on the situation so clients can decide upon their wanted solution

Exceptions

Methods Exceptions

Methods

- **public String getMessage()** – Provides information about the exception that has occurred through a message, which is initialized in the *Throwable constructor*.
- **public Throwable getCause()** – Provides root cause of the exception as represented by a *Throwable object*.
- **public void printStackTrace()** – Used to display the output of *toString()* along with the stack trace to *System.err* (error output stream).
- **public StackTraceElement [] getStackTrace()** – Returns an array with each element present on the stack trace. The index 0 element will symbolize the top of the call stack, and the last element of array will identify the bottom of the call stack.

Exceptions

Catching Exception

Catching Exceptions

- Should be written as close to the origin throwing point as possible.
- Catching java.lang.Exception will catch all types of exceptions.
- Use java.lang.Exception methods to get information:

```
Try{
      .............
    }catch(Exception e){

   System.out.println(e.getMessage());

   e.printStackTrace(System.out);
}
```

Exceptions

Catching Exceptions

Catching Exception

```
public class Test {

public static void main(String args[]) {
try {
        int a[] = new int[2];
        System.out.println("Access element three:" +a[3]);
}catch (ArrayIndexOutOfBoundsException e) {
                System.out.println("Exception thrown :"+ e);
                } System.out.println("Out of the block");
        }

}
```

Exceptions

Catching Exceptions

Catching Exception

```
public void check(String fileName, String value) {

try {
     FileInputStream in = new FileInputStream(fileName);
    if (Integer.parseInt(value) >= 100) {return;}
} catch (I Exception e) {// handle I/O problem…
              return;
} catch (NumberFormatException e) {
              // handle runtime exception…
} finally {
System.out.println("This is printed in any case...");}
       System.out.println("Done!");}
```

Exceptions

Throwing Exceptions

Throwing Exception

- Any method can delegate exceptions to the caller.
- A method must declare any thrown checked Exception as part of its signature.
- Throwing Runtime Exceptions (unchecked) is allowed but not always necessary.
- **throws** – used in a method or constructor signatures to declare all their thrown exceptions.
- **throw** – is used inside a method body when throwing a created exception object.

Exceptions

Throwing Exceptions

Throwing Exception

```
public class Checking {

public static int check(String s) throws NumberFormatException
        {
        return Integer.parseInt(s);
        }


public static void main(String[] args) {
        int num = check(args[0]);
        System.out.println(num + 1);
        }
}
```

Throwing Exceptions

Throwing Exception

```
public class Checking {
public static int check(String s) throws NumberFormatException {
        int x = Integer.parseInt(s);
        if (x > 100)
        throw new NumberFormatException("Number is too big");
        return x;
}

public static void main(String[] args) {
        int num = check(args[0]);
        System.out.println(num + 1);
        }
}
```

Exceptions

Customized Exceptions

Customization

```
public class NumberOutOfLimitsException extends Exception {

// added field
private int num = 0;

public NumberOutOfLimitsException(String msg, int num) {
        super(msg);
        this.num = num;
        }

// added method
public int getNum() {
        return num;
        }
}
```

Customized Exceptions

Exceptions

Customization

```
public class NumChecker {
     public void check(int num) throws
NumberOutOfLimitsException {
          if (num < 0 || num > 100)
  throw new NumberOutOfLimitsException("Wrong value", num);
     }
}

public  class TestChecker {
     public static void main(String[] args) {
          NumChecker nc = new NumChecker();
    try {
         nc.check(Integer.parseInt(args[0]));
         System.out.println(args[0] + " is OK");
        } catch (NumberOutOfLimitsException e) {
        System.out.println(e.getMessage() + " " +
        e.getNum());
               }
         }
        }
```

Exceptions

Thank You !!