Programming in Java – Exceptions

Paolo Vercesi
Technical Program Manager

© 2019 ESTECO SpA

Exception basics

Checked and unchecked exceptions

© 2019 ESTECO SpA

Exception basics

© 2019 ESTECO SpA

How to report error conditions

When implementing a method there are three traditional approaches to report error conditions

Error codes

The method returns an error code.

E. g. 0 if everything is ok, -1 if an error happens, etc.

Error flags

The method set/reset an error flag in its class to report an error condition.

Exception

The method throws an Exception to interrupt its execution and to inform the caller that an exceptional condition arose.

© 2019 ESTECO SpA

Error codes

```java
public class FixedSizeDisplay {

    public static final int OK = 0;
    public static final int TEXT_LENGTH_TOO_BIG = 1;

    private static final int SIZE = 10;

    public int display(String text) {
        if (text.length() > SIZE) {
            System.out.println(text.substring(0, 10));
            return TEXT_LENGTH_TOO_BIG;
        } else {
            System.out.println(text);
            return OK;
        }
    }
}
```

The method cannot return any value, it must return an error code

The caller must always check the returned error code

© 2019 ESTECO SpA

Error flags

```
public class FixedSizeDisplay {

    private static final int SIZE = 10;

    private boolean error;

    public void display(String text) {
        if (text.length() > SIZE) {
            System.out.println(text.substring(0, 10));
            error = true;
        } else {
            System.out.println(text);
            error = false;
        }
    }

    public boolean checkError() {
        return error;
    }
}
```

The method can return any value

The caller must always check the error status, usually by using a method of the same class

The PrintStream class (the class of System.out) uses this approach.

© 2019 ESTECO SpA

Throwing exceptions

```
public class FixedSizeDisplay {

    private static final int SIZE = 10;

    public void display(String text) throws Exception {
        if (text.length() > SIZE) {
            System.out.println(text.substring(0, 10));
            throw new Exception("Text length: " +
                text.length() + " exceeds display size");
        }
        System.out.println(text);
    }
}
```

The keyword throw is used to throw exceptions

The caller of display(String) shouldn't check all the invocations, but it needs to deal with exceptional cases only.

When we throw an exception, the method execution is interrupted at the point where the exception is thrown, and the exception is propagated to the caller hierarchy until it is caught by an appropriate try-catch block.

© 2019 ESTECO SpA

Example

```java
public class ThrowException {

    public static void main(String[] args) {
        String s = "Hello";

        System.out.println(s.charAt(10));
    }
}
```

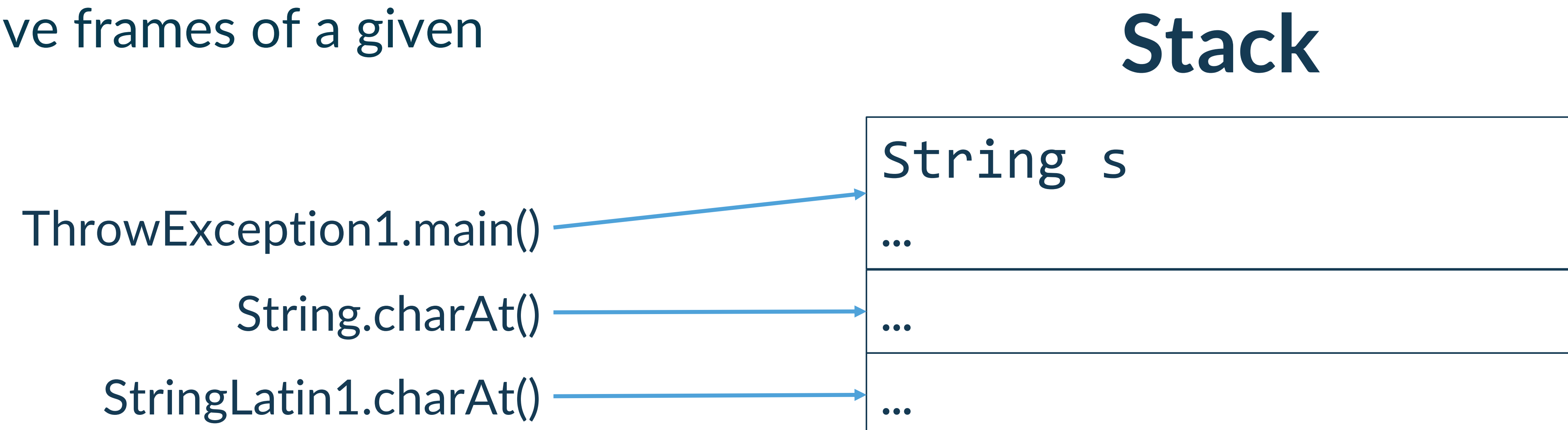
An exception is an Object that describes an exceptional condition. It brings with itself a stack trace and usually an explanatory message.

```
$ java.exe it.units.sdm.exceptions.ThrowException
Exception in thread "main" java.lang.StringIndexOutOfBoundsException: String index out of range: 10
	at java.base/java.lang.StringLatin1.charAt(StringLatin1.java:48)
	at java.base/java.lang.String.charAt(String.java:1512)
	at it.units.sdm.exceptions.ThrowException1.main(ThrowException1.java:8)
```

© 2019 ESTECO SpA

Stack trace

A stack trace is a report of the active frames of a given thread.



The exception stack trace reports all the method in the stack by indicating the fully-qualified class name and the line number of the last executed instruction for each method.

```
$ java.exe it.units.sdm.exceptions.ThrowException
Exception in thread "main" java.lang.StringIndexOutOfBoundsException: String index out of range: 10
        at java.base/java.lang.StringLatin1.charAt(StringLatin1.java:48)
        at java.base/java.lang.String.charAt(String.java:1512)
        at it.units.sdm.exceptions.ThrowException1.main(ThrowException1.java:8)
```

The same information can be obtained for debugging by invoking the static method Thread.dumpStack()

© 2019 ESTECO SpA

Catching exceptions

```java
public class CatchException {

    public static void main(String[] args) {
        String s = "Hello";

        try {
            System.out.println(s.charAt(10));
        } catch (StringIndexOutOfBoundsException ex) {
            System.out.println("An error happened: " + ex.getMessage());
        }
    }
}
```

```
$ java it.units.sdm.exceptions.ThrowException1
An error happened: String index out of range: 10
```

Exceptions can be caught by try-catch blocks

© 2019 ESTECO SpA

Checked and unchecked exceptions

© 2019 ESTECO SpA

The Exceptions hierarchy

`Throwable`

All exceptions are subclasses of Throwable

`Exception`

`Error`

An Error indicates a serious problems that a reasonable application should not try to catch. Most such errors are abnormal conditions.

`RuntimeException`

`CheckedException`

RuntimeException and its subclasses are unchecked exceptions. Unchecked exceptions do not need to be declared in a method or constructor's throws clause.

The checked exception classes are all exception classes other than the unchecked exception classes. Checked exceptions need to be declared in method or constructor’s throws clause.

© 2019 ESTECO SpA

Exception types

Throwable	Never catch Throwables.
Error	Errors happening at the JVM level.
Exception	The base class for checked and unchecked exceptions. By Catching Exception you catch all the exceptions that can be solved by the application logic.
RuntimeException (unchecked exceptions)	Might indicate a bug in the application. Usually are not caught because the problem is at the code level. E.g., NullPointerException, ArrayIndexOutOfBoundsException, StringIndexOutOfBoundsException, etc.
Checked exception	Indicate exceptions that can be caused by wrong data and they can be addressed by the application logic. You should catch and manage them. E.g., when you catch a FileNotFoundException you can ask the user to indicate another file.

© 2019 ESTECO SpA

Creating your own exceptions

```
public class FixedSizeDisplay {

    private static final int SIZE = 10;

    public void display(String text) throws TextTooLongException {
        if (text.length() > SIZE) {
            var newText = text.substring(0, 10);
            System.out.println(newText);
            throw new TextTooLongException(text.length());
        }
        System.out.println(text);
    }

    public static class TextTooLongException extends Exception {

        public TextTooLongException(int size) {
            super("Text length: " + size + " exceeds display size");
        }
    }
}
```

© 2019 ESTECO SpA

Exceptions in interfaces 1/2

it.units.sdm.Display

```
public interface Display {

    void display(String text) throws TextTooLongException;

}
```

it.units.sdm.Calculator

```
public class Calculator {

    final Display display;
    //...

    Calculator(Display display) {
        this.display = display;
    }

     void onePressed() throws TextTooLongException {
        string += "1";
        display.display(string);
    }
}
```

When we invoke display(String) we must either declare to throw the TextTooLongException or catch it

© 2019 ESTECO SpA

Exceptions in interfaces 2/2

```
public interface Display {
  void display(String text) throws TextTooLongException;
}

class MyDisplay implements Display {
    @Override
    public void display(String text) throws MyTextTooLongException {
        //..
    }
}

class ConsoleDisplay implements Display {
    @Override
    public void display(String text) {
        //..
    }
}

class PopupDisplay implements Display {
    @Override
    public void display(String text) throws Exception {
        //..
    }
}
```

The implementation can throw a subclass of the declared exception(s)

The implementation can not throw any exception

© 2019 ESTECO SpA

Handling multiple exceptions 1/2

```
public static void main(String[] args) {
    try {
        myMethod();
    } catch (UserException1 ex) {
        // do something
    } catch (UserException2 ex) {
        //do something
    }
}

static void myMethod() throws UserException1, UserException2 {
    if (System.currentTimeMillis() % 5 == 0) {
        throw new UserException1();
    } else {
        throw new UserException2();
    }
}
```

All exceptions must be caught or declared in the method declaration

A method can declare more exceptions

© 2019 ESTECO SpA

Handling multiple exceptions 2/2

```
public static void main(String[] args) {
    try {
        myMethod();
    } catch (UserException1 | UserException2 ex) {
        // do something
    }
}

static void myMethod() throws UserException1, UserException2 {
    if (System.currentTimeMillis() % 5 == 0) {
        throw new UserException1();
    } else {
        throw new UserException2();
    }
}
```

Exceptions can be caught together. If we perform the same recover operation.

Why don't we use
`catch (Exception ex)`
to catch both the exceptions?

© 2019 ESTECO SpA

try-catch-finally 1/3

```java
public class Storage {

    public void store(String text) throws TimeoutException {
        //do something
    }

    public void close() {
        //close
    }

    public static void main(String[] args) throws TimeoutException {
        var storage = new Storage();
        for (String arg : args) {
            storage.store(arg);
        }
        storage.close();
    }
}
```

The specification of the Storage class says that a storage object must be closed, to be sure that all data has been store.

Are we satisfying the specification?

© 2019 ESTECO SpA

try-catch-finally 2/3

```
public class Storage {

    public void store(String text) throws TimeoutException {
        //do something
    }

    public void close() {
        //do something
    }

    public static void main(String[] args) throws TimeoutException {
        var storage = new Storage();
        try {
            for (String arg : args) {
                storage.store(arg);
            }
        } finally {
            storage.close();
        }
    }
}
```

The finally block *always* executes when the try block exits. This ensures that the finally block is executed even if an unexpected exception occurs.

© 2019 ESTECO SpA

try-catch-finally 3/3

```java
public static void main(String[] args) throws TimeoutException {
    var storage = new Storage();
    try {
        for (String arg : args) {
            int i = 0;
            while (true) {
                try {
                    storage.store(arg);
                    break;
                } catch (TimeoutException ex) {
                    if (++i == 5) {
                        throw ex;
                    }
                }
            }
        }
    } finally {
        storage.close();
    }
}
```

try-catch-finally blocks can be nested.

Exceptions can be rethrown

Is this code readable?

© 2019 ESTECO SpA

Chaining exceptions

```
public interface Display {
    void display(String text) throws DisplayException;
}

class PopupDisplay implements Display {

    @Override
    public void display(String text) throws DisplayException {
        try {
            double fontWidth = 100.0 / text.length();
        } catch (ArithmeticException ex) {
            throw new DisplayException(ex);
        }
    }
}
```

```
Exception in thread "main" it.units.sdm.DisplayException: java.lang.ArithmeticException: / by zero
        at it.units.sdm.PopupDisplay.display(FixedSizeDisplay.java:63)
        at it.units.sdm.MainDisplay.main(FixedSizeDisplay.java:7)
Caused by: java.lang.ArithmeticException: / by zero
        at it.units.sdm.PopupDisplay.display(FixedSizeDisplay.java:60)
        ... 1 more
```

© 2019 ESTECO SpA

Assignment

```java
public interface Collection {

    boolean isEmpty();

    int getSize();

    boolean contains(String string);

    String[] getValues();
}

public interface Stack extends Collection {

    void push(String string);

    String pop();

    String top();
}
```

```java
public interface List extends Collection {

    void add(String string);

    String get(int index);

    void insertAt(int index, String string);

    void remove(int index);

    int indexOf(String string);

}
```

Implement the Stack and List interfaces.

Use exceptions to report wrong usages
e. g. illegal indexes, empty stack, etc.

© 2019 ESTECO SpA

More on exceptions, from the creator of Java

Failure and Exceptions A Conversation with James Gosling

https://www.artima.com/articles/failure-and-exceptions

© 2019 ESTECO SpA

Thank you!

esteco.com

Read the ESTECO Copyright Policy © 2019 ESTECO SpA