Errors and Exceptions

Exception mechanism in Java

- Used for
  - Exceptions
  - Programming errors
  - Irrecoverable problems

- Exceptions and errors:
  - Are thrown
  - Can be caught
  - Organized in a hierarchy

Irrecoverable errors shouldn't be caught!

Exception hierarchy

- `Throwable`
  - `Error`
    - `VirtualMachineError`
    - …
  - `Exception`
    - `IOException`
    - …
    - `RuntimeException`
      - `ArithmeticException`
      - `NullPointerException`
      - `IndexOutOfBoundsException`
      - …

Irrecoverable errors, shouldn't be caught

Checked exceptions, must declare and handle or delegate explicitly.

Unchecked exceptions, programming errors, not declared.

Exceptions

- `Throwable`
  - ~~`Error`~~
  - `Exception`
    - `IOException`
    - …
    - ~~`RuntimeException`~~

- Part of the program
- Always declared
- Correct programs always handle them

Programming errors

- Throwable
  - ~~Error~~
  - Exception
    - ~~IOException~~
    - ~~…~~
    - RuntimeException
      - NullPointerException
      - …

- Not part of the program expected behavior:
  - Contract violations
  - Invariant violations
- Undeclared
- May be captured (situation dependent)

Irrecoverable errrors

- `Throwable`
  - `Error`
  - ~~`Exception`~~

- Should not be used, reserved for JVM problems

Throw (exception)

- When there is an exceptional situation throw a sub-class of `Exception`
- Example:

```
public … open(…) throws FileNotFoundException {
    …
    if (file == null)
        throw new FileNotFoundException();
    …
}
```

Throw (error)

- If there is a programming error…
  …throw a sub-class of `RuntimeException`
- Example:

```
public double sqrt(final double value) {
    if (value < 0.0)
        throw new
  IllegalArgumentException();
    …
}
```

Assert

- Programmer verification of a condition that must hold at a given point
- Usually only active when debugging

```
…
double x = absoluteValue(y);
assert 0.0 <= x;
…
```

Checks for an error.

Assert (with message)

- Flag -ea

```
…
double x = absoluteValue(y);
assert 0.0 <= x : “Error, negative
  abs value” ;
…
```

Programmer roles

- For a given module
  - Producer
  - Consumer

Concepts

User error	Not an error, must be expected. Should be dealt with using normal flow control primitives.
Exception	Part of the program logic. Usually associated to accessing exernal resources. Must be handeled.
Programming errors	Can be handled or not
JVM errors	Not handled

Representation and creation

- Errors and exceptions are *throwable*

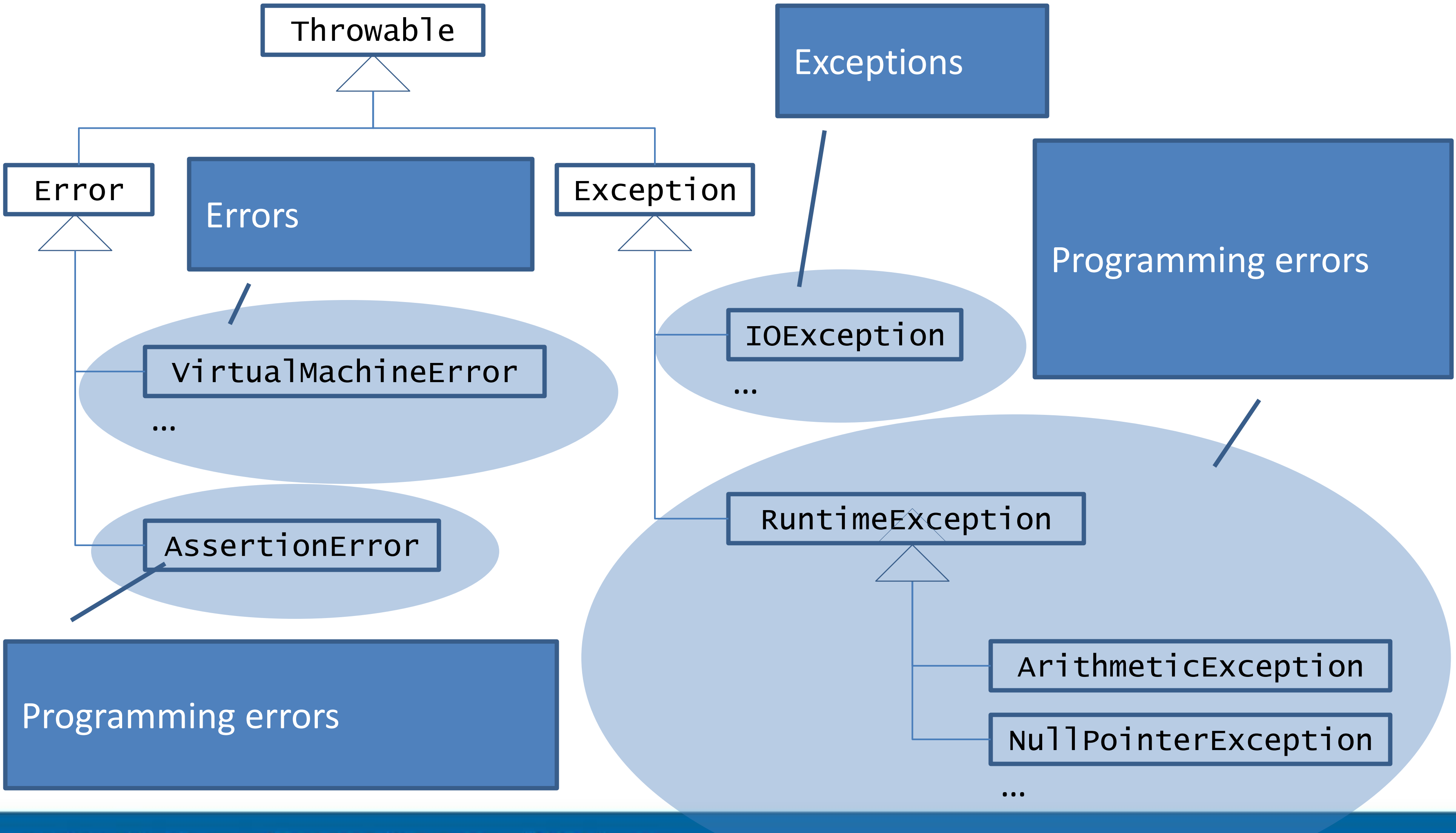
- Exceptions are created and thrown
  - Explicitly: throw
  - Implicitly: assert
  - Automatically: JVM
  - By methods we don’t control

Detection and treatment

- Exception can be caught
  - To deal with exception (totally or partially)
  - To clean up before ending the program
  - To finish the program adequately

Not a recovery, but an elegant ending

Hierarchy


Throwable
Exceptions
Error
Errors
Exception
Programming errors
IOException
…
VirtualMachineError
…
RuntimeException
AssertionError
ArithmeticException
NullPointerException
…
Programming errors

Caracteristics

Possible throw must be declared!

Ocorrência	Classe	Criação	Instrução	Recuperação[1]
Exception	`Exception`[2]	Explícita	`throw`	Sim
Programming error[3]	`RuntimeException`	Explícita	`throw`	Não
Programming error[4]	`AssertionError`	Implícita	`assert`	Não
JVM error	`Error`[5]	Automática	-	Não

[1] Non-critical applications.

[2] Except `RuntimeException` and derived.

[3] For violation of pre-conditions in contracts

[4] All other cases.

[5] Except `AssertionError`.

Explicit throw

Declaring possible throw:
- Mandatory for `Exception` (except `RuntimeException`).
- Not recommended for others (`Error` e `RuntimeException`).

```
public void someMethod(…) throws SomeException {
    …

    if (…)
        throw new SomeException("Informative message");

    …
}
```

Explicit throw

```
static public double squareRoot(final double value) {
    if (value < 0.0)
        throw new IllegalArgumentException(
            "Illegal value " + value
            + ", should be 0 ≤ value");
    …
}
```

Violation of pre-conditions

Delegation (throws)

```
public void someMethod(…) throws SomeException {
    …

    anObject.someOtherMethod(…);

    … // only if no exception is thrown.
}
```

Catching

```
public void someMethod(…) {
    try {
        …
        anObject.someOtherMethod(…);

        … // only if no exception is thrown.
    } catch (final SomeException exception) {
        … // fix the problem using information available.
    }

    … // continue execution.
}
```

Cleaning up with `finally`

```
public void someMethod(…) {
    try {
        …
        anObject.someOtherMethod(…);
        …
    } catch (final SomeException exception) {
        …
    } finally {
        … // clean house in any case.
    }

    …
}
```

`finally` block is executed regardless of success or failures of exception.

Cleaning up with `autocloseable`

```
public void someMethod(…) {
    try (... open autocloseable resources) {
        …

     } catch (final IOException exception) {
        …
    }
    …
}
```

`close` always executed regardless of success or failures of exception.

Rethrow

```
public void someMethod(…) throws SomeException {
    try {
        …
        anObject.someOtherMethod(…);

        …
    } catch (final SomeException exception) {
        … // fix part of the problem using information available.
        throw exception;
    }

    …
}
```

Additional information

```
public void someMethod(…) throws SomeOtherException {
    try {
        …
        anObject.someOtherMethod(…);

        …
    } catch (final SomeException exception) {
        … // clean house if necessary.
        throw new SomeOtherException(
            "Informative message", exception);
    }

    …
}
```

Exception causes new throw

Multiple catch

```
public void someMethod(…) {
    try {
        …
    } catch (final SomeException exception) {
        …
    } catch (final RuntimeException exception) {
        …
    } catch (final IOException exception) {
        …
    } catch (final Exception exception) {
        …
    }

    …
}
```

More specific expessions first

New Multiple catch

```
public void someMethod(…) {
    try {

        …

    } catch (SomeException | RuntimeException |
                IOException | Exception exception) {
        …
    }

    …
}
```

printStackTrace

```
public void someMethod(…) {
    …

    try {
        …
    } catch (final SomeException exception) {
        exception.printStackTrace();
        …
    }

    …
}
```

print invocation stack active when throw was executed

```
pt.iscte.dcti.poo.exceptions.SomeException
at pt.iscte.dcti.poo.SomeClass.someMethod(SomeClass.java:16)
at pt.iscte.dcti.poo.SomeOtherClass.someOtherMethod(SomeOtherClass.java:9)
at pt.iscte.dcti.poo.MainClass.main(MainClass.java:36)
```

User defined

```
public class SomeException extends Exception {

    public SomeException() {
    }

    public SomeException(final String message) {
        super(message);
    }

    public SomeException(final String message,
                         final Throwable cause) {
        super(message, cause);
    }

    public SomeException(final Throwable cause) {
        super(cause);
    }

}
```

Good practice

- Distinguish clearly errors from the others
- Deal with each exception using the most adequate mechanisms

Good practice

- Use `assert` only for programming errors

- Use `throw`
  - For programming errors
  - Exceptional cases

Good practice

- Do not recover until you know the reason for the failure
- Do not deactivate assertions unless they are too time-consuming
- Try to use existing exceptions

Good practice

- Comment possible throw of `RuntimeException` (and sub-classes) when thrown by incorrect user-interface usage

- Can catch and rethrow to improve readability or partially solve the problem

Good practice: security

- Code so that exception throw leaves object unchanged

- If not possible, try to leave all objects in a valid and coherent state

More …

- Exceptions:
  - http://java.sun.com/docs/books/tutorial/essential/exceptions/index.html
- Assert:
  - http://java.sun.com/j2se/1.4.2/docs/guide/lang/assert.html
- Y. Daniel Liang, *Introduction to Java Programming*, 7.ª edição, Prentice-Hall, 2010.

Summary

- Errors and Exceptions