# Erros e Excepções

# Mecanismo de excepções do Java

- Usado para
  - Excepções propriamente ditas (casos excepcionais)
  - Erros de programação
  - Erros irrecuperáveis (a ver mais tarde)

- Excepções e erros:
  - São *lançados* (a ver mais tarde)
  - Podem ser *apanhados*
  - Organizados numa *hierarquia* (como uma taxonomia)

Mas erros irrecuperáveis não devem ser apanhados!

# Hierarquia de erros e excepções em Java

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

Erros irrecuperáveis, que não é razoável apanhar e processar.

Excepções propriamente ditas (casos excepcionais) cujo possível lançamento é *obrigatório declarar e capturar ou delegar*.

Erros de programação, cujo possível lançamento *não se declara*.

# Casos excepcionais (Exception)

- Parte da lógica do programa
- Possíveis lançamentos declaram-se sempre
- Programas correctos lidam sempre com elas

# Erros de programação (RunTimeException)

- Não fazem parte da lógica do programa:
  - Violações de contratos
  - Violações de invariantes
- Possíveis lançamentos não se declaram
- Em certos casos podem ser capturados e tratados

# Erros irrecuperáveis

- Em geral não devem ser usados, são lançados quando há problemas graves / irrecuperáveis de funcionamento da máquina virtual ou bibliotecas

# Lançamento: caso excepcional

- Situação excecional: lançar subclasse de `Exception`
- Exemplo:

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

# Lançamento: erro de programação

- Erro de programação: lançar subclasse de `RuntimeException`
- Exemplo:

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

# Asserção

- Proposição que programador pretende que seja sempre verdadeira num dado ponto do programa
- Suportada no Java pela instrução `assert`
- Verificação activa usualmente durante desenvolvimento (opção -ea da máquina virtual)

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

Permite detectar imediatamente um erro no cálculo do valor absoluto.

# Papéis do programador

- Relativamente a um módulo o programador pode ser
  - Produtor – Se o desenvolveu total ou parcialmente
  - Consumidor – Se o usa de alguma forma

# Conceitos

| | |
|---|---|
| **Erro do utilizador humano** | Não é um erro! É uma situação normal e expectável. Deve lidar-se com situações deste tipo usando as ferramentas usuais de controlo de fluxo do Java. |
| **Caso excepcional** | Os casos excepcionais fazem parte da lógica do programa, embora não possam ser considerados parte da seu fluxo de operação *normal*. Muitas vezes estão associados a problemas em recursos externos a que o programa precisa de recorrer. |
| **Erro de programação** | Os erros de programação são erros cometido pelos programadores e por isso não podem ser considerados parte da lógica do programa. |
| **Erros no ambiente de execução** | São erros que ocorrem no ambiente de execução do programa, não correspondendo nem a casos excepcionais, nem a erros de programação; não são parte da lógica do programa. |

# Representação e criação

- Em Java, casos excepcionais, erros de programação e erros no ambiente de execução representam-se através de *lançáveis* (designam-se em geral por *excepções*)

- As excepções são criadas e lançadas
  - Explicitamente: throw
  - Implicitamente: assert
  - Automaticamente: JVM, por exemplo
  - Por métodos sobre os quais não temos controlo

# Detecção e tratamento

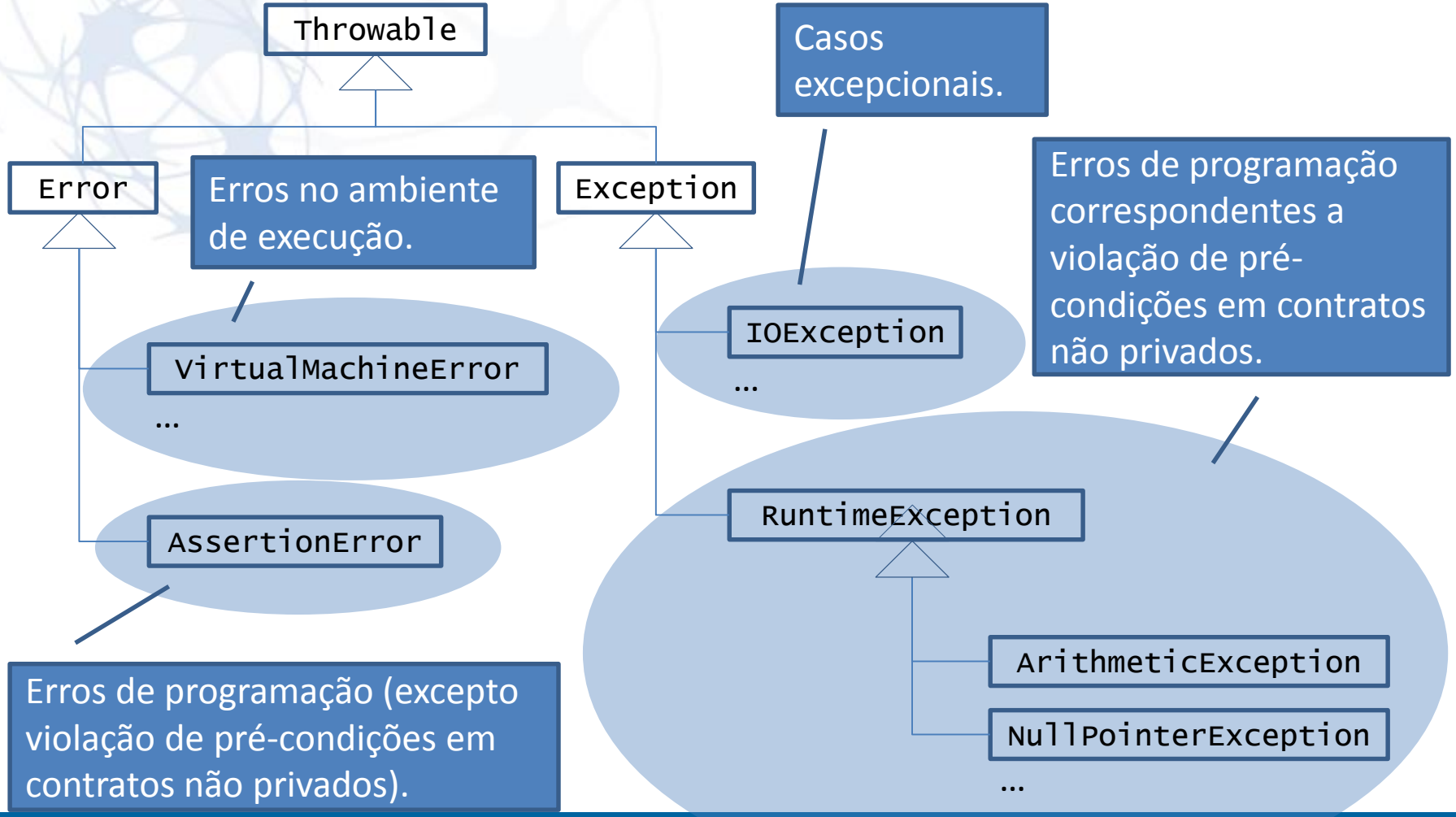
- As excepções em Java podem ser capturadas
  - Para lidar integralmente com um caso excepcional
  - Para lidar parcialmente com um caso excepcional
  - Apenas para "arrumar a casa" , quando não se sabe lidar com um caso excepcional
  - Para terminar o programa de forma adequada em caso de erro de programação ou de erro no ambiente de execução

Isto não é uma recuperação, mas sim uma finalização elegante

# Excepções Java

- Usados para representar
  - Casos excepcionais
  - Erros de programação
  - Erros no contexto de execução de um programa
- São objectos de classes pertencentes à hierarquia com base na classe `Throwable`
- São lançáveis através da instrução `throw`
- Lançamento interrompe o fluxo normal de um programa

# Hierarquia de lançáveis em Java



# Hierarquia de lançáveis em Java

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

Erros no ambiente de execução. Criação automática. Normalmente irrecuperáveis.

Erros de programação (excepto violação de pré-condições em contratos não privados). Criação implícita quando asserção falha. Normalmente irrecuperáveis.

Casos excepcionais. Criação explícita. Lançamento com `throw`. Normalmente recuperáveis.

Erros de programação correspondentes a violações pré-condições de contratos não privados. Criação explícita. Lançamento com `throw`. Normalmente irrecuperáveis.

# Características

Possibilidade de lançamento tem de ser declarada!

| Ocorrência | Classe | Criação | Instrução | Recuperação[1] |
|---|---|---|---|---|
| Caso excepcional | `Exception`[2] | Explícita | `throw` | Sim |
| Erro de programação[3] | `RuntimeException` | Explícita | `throw` | Não |
| Erro de programação[4] | `AssertionError` | Implícita | `assert` | Não |
| Erros no ambiente de execução | `Error`[5] | Automática | - | Não |

[1] Para aplicações não críticas.
[2] Excepto `RuntimeException` e derivadas.
[3] Para violações de pré-condições em contratos não privados.
[4] Para todos os restantes casos.
[5] Excepto `AssertionError`.

# Algumas `RuntimeException`

| Excepção | Utilização |
|---|---|
| `IllegalArgumentException` | Argumento de método não é válido. |
| `IllegalStateException` | Estado de objecto é inválido ou não permite a operação. |
| `NullPointerException` | Tentativa de acesso a instância através de referência nula. |
| `IndexOutOfBoundsException` | Índice de matriz ou colecção fora dos limites válidos. |
| `ConcurrentModificationException` | Tentativa de alteração de objecto em momento em que tal não é permitido (e.g., alteração de lista concorrente com iteração dessa lista). |
| `UnsuportedOperationException` | Classe do objecto não suporta a operação em questão. |

# Lançamento explícito para caso excepcional

Declaração de possibilidade de lançamento:
- Obrigatória para `Exception` (excepto `RuntimeException`).
- Não recomendada para restantes casos (`Error` e `RuntimeException`).

```
public void someMethod(…) throws SomeException {
    …
    if (…)
        throw new SomeException("Informative message");
    …
}
```

# Lançamento explícito para erro de programação

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

Violação de pré-condição em contrato não privado.

# Delegação ou declaração de passagem (throws)

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

    anObject.someOtherMethod(…);

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

# Lidando com caso excepcional

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

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

    … // continue execution.
}
```

# Arrumando a casa com `finally`

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

    …
}
```

Bloco `finally` executado mesmo no caso de excepções não capturadas, retornos, novos lançamentos, asserções falhadas, etc.

# Auto-close (try-with-resources)

```
public void someMethod(…) {
    try (Scanner s = new Scanner(new File(“test.txt”);
         PrintWriter w = new PrintWriter(new File(“other.txt”)))){
        …
    } catch (final FileNotFoundException exception) {
        …
    }
}
```

Ambos os ficheiros são fechados automaticamente, mesmo em caso de excepção. Fecho ocorre por ordem inversa à de criação

# Lidando parcialmente com caso excepcional

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

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

    …
}
```

Relançamento da excepção capturada.

# Informação adicional sobre a ocorrência

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

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

    …
}
```

Apenas se se tratar de um caso excepcional.

Excepção capturada é a causa do novo lançamento.

# Captura múltipla

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

    …
}
```

Excepções mais específicas têm de estar primeiro.

# Operação `printStackTrace`

```
public void someMethod(…) {
    …

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

    …
}
```

Imprime uma representação da pilha de invocações no momento em que a excepção foi lançada.

```
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)
```

# Excepções definidas pelo programador

```
public class SomeException extends Exception {
    public Exception() {
    }
    public Exception(final String message) {
        super(message);
    }
    public Exception(final String message,
                     final Throwable cause) {
        super(message, cause);
    }
    public Exception(final Throwable cause) {
        super(cause);
    }
}
```

# Boas práticas

- Distinga claramente a natureza dos erros e casos excepcionais

- Lide com cada ocorrência usando os mecanismos adequados (e convencionais) à sua natureza

# Boas práticas

- Use `assert` apenas para erros de programação
  - Pré-condições de métodos privados
  - Pós-condições
  - Invariantes de instância
  - Asserções propriamente ditas

- Use `throw`
  - Para erros de programação
    - Pré-condições de métodos não privados
  - Casos excepcionais

# Boas práticas

- Não tente recuperar de uma ocorrência sem antes perceber a sua causa
- Não desactive as asserções durante a operação normal do *software senão quando tiverem um impacte negativo significativo sobre o seu desempenho*
- Tente recorrer a excepções já existentes na biblioteca padrão do Java

# Boas práticas

- Documente possibilidade de lançamento de de `RuntimeException` (e classes derivadas) quando esse lançamento puder ocorrer devido erros na utilização da interface da operação

- Lembre-se que pode capturar uma excepção e lançar um outro que torne mais claro qual o problema (não deve abusar desta estratégia)

# Boas práticas: segurança

- Escreva o seu código de modo a que o lançamento de uma excepção num método deixe o programa no mesmo estado em que estava antes da operação correspondente ser invocada

- Se isso não for possível, escreva o código de modo a, pelo menos, deixar todos os objectos num estado válido

# Mais informação

- Excepções:
  - http://java.sun.com/docs/books/tutorial/essential/exceptions/index.html
- Asserções:
  - 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.

# Sumário

- Erros e Excepções