Erros e Exceções

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

Exceções

- Lançamento interrompe execução normal e "salta" para a cláusula catch correspondente
- Apanhar e tratar uma exceção pode ser feito em vários pontos do código
- Há excepções cujo tratamento ou delegação é obrigatório

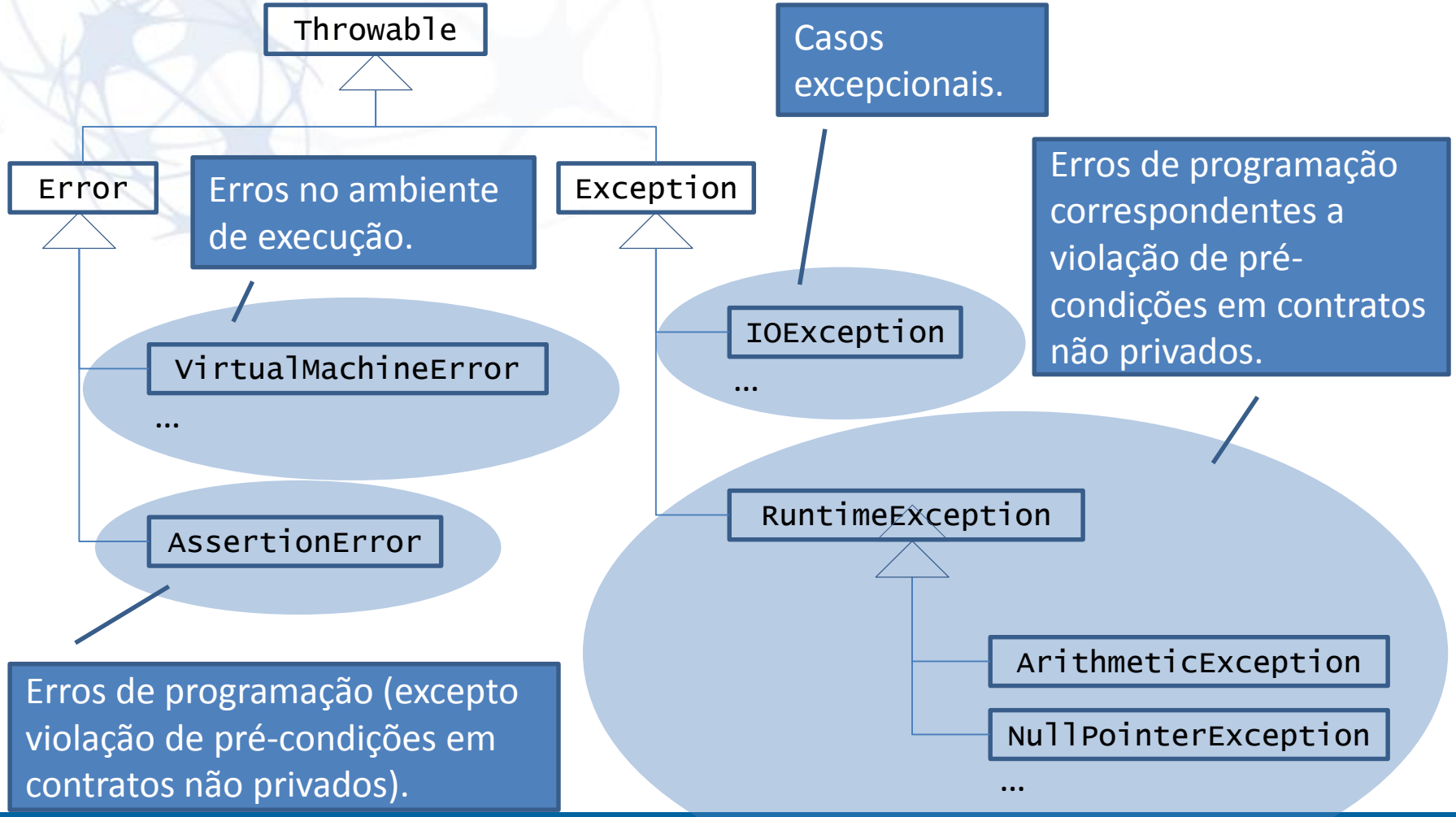
Detecção e tratamento

- Lidar integralmente com o caso
- Lidar parcialmente com o caso
- "Arrumar a casa" e sair
  - Avisar o utilizador
  - Terminar o programa de forma adequada, fechando os recursos

Mecanismo de excepções do Java

- Usado para
  - Exceções
  - Erros de programação
  - Erros irrecuperáveis
- Excepções e erros:
  - São *lançados* (a ver mais tarde)
  - Podem ser *apanhados*
  - Organizados numa *hierarquia*

Hierarquia de lançáveis em Java



Casos excecionais (Exception)

- Parte da lógica do programa
- Possíveis lançamentos declaram-se sempre
- Programas correctos lidam sempre com elas
- Têm que ser tratadas ou delegadas explícitamente

Erros de programação (RunTimeException)

- Não fazem parte da lógica do programa
- Não se declaram
- Em certos casos podem ser capturados e tratados

Erros irrecuperáveis

- Não devem ser usados
- Não devem ser apanhados

Lançamento

- Lançar subclasse de `Exception`
- Exemplo:

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

Lançamento: erro de programação

- Lançar subclasse de `RuntimeException`
- Exemplo:

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

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.

Asserção

- Verificação do programador
- No Java: `assert`
- Opção -ea da máquina virtual

…

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

…

Lançamento explícito (erro de programação)

```
static public double squareRoot(final double value)
    {
        if (value < 0.0)
            throw new IllegalArgumentException(
                "Illegal value " + value
                + ", should be 0 ≤ value");
        //só passa aqui se não for lançada excepção
        …
}
```

Lançamento explícito

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");
    …
}
```

Delegação ou declaração de passagem (throws)

```
public void someOtherMethod(…) throws
  SomeException {
    …
    anObject.someMethod(…);
    //só passa aqui se não for lançada
  excepção
}
```

Apanhar exceção

```
public void yetAnotherMethod(…) {
    try {
        …
        anotherObject.someOtherMethod(…);
        …
    } catch (final SomeException exception) {
        … // corrigir problema se possível
    }
    … // continuar a execução
}
```

Corrigir e sair

```
public void yetAnotherMethod(…) {
    try {
        …
        anotherObject.someOtherMethod(…);
        …
    } catch (final SomeException exception) {
        … // corrigir problema se possível
      return;
    }
    … // não continua a execução
}
```

Arrumar a casa com `finally`

```
public void yetAnotherMethod(…) {
    try {
        …
        anotherObject.someOtherMethod(…);
        …
    } catch (final SomeException exception) {
        …
    } finally {
        myFile.close();
        … // arrumar a casa
    }
}
```

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

Resposta parcial

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

        …
    } catch (final SomeException exception) {
       // impossível corrigir totalmente, relançar
       throw exception;
    }
}
```

Informação adicional

```
public void someMethod(…) throws SomeOtherException {
    try {
        anObject.someOtherMethod(…);
    } catch (final SomeException exception) {
        // ou lançar outra excepção
        throw new SomeOtherException(
             "Informative message", exception);
    }
    …
}
```

Captura múltipla

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

Operação `printStackTrace`

```
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)
```

Definir uma exceção

```
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

- Distinguir erros dos casos excecionais
- Lidar com cada caso usando os mecanismos adequados
- Nunca deixar dados incoerentes (de preferência tudo ou nada)
- Nunca deixar o programa terminar sem avisar o utilizador e fechar recursos

Boas práticas

- Perceber causa ao recuperar
- Usar asserções adequadamente
- Use exceções da biblioteca sempre que possível

Especificação e verificação do contrato

- Pré-condições
- Pós-condições
- Documentação
- Pré-condições verificadas usando
  - Métodos não privados: *excepções*
  - Métodos privados: *asserções*
- Pós-condições verificadas usando *asserções*

Vantagens da verificação do contrato

- Erros detectados mais cedo
- Localização mais precisa de erros
- Menos erros no programa final
- Maior confiança na qualidade do código

Manual do consumidor e contrato

```
/**
   Calculates and returns the greatest absolute value
   of two positive integers
   @param   m the first integer
   @param   n second integer
   @return  the maximum of the two parameters
   @pre     m >= 0 && n >= 0
   @post    (max == m || max == n) && (max >= n && max >= m)
*/
public static int max(final int m, final int n) { … }
```

Estas etiquetas não são suportadas nativamente pelo javadoc.

Verificação do contrato: pré-condições

```
public static int max(final int m, final int n) {
    if (m < 0 || n < 0)
        throw new IllegalArgumentException(
            "Illegal arguments: " + m + ", " + n);

    … // implementação    …

    … // verificação da pós-condição

    return …;
}
```

Verificação da pré-condição do contrato.

(Condição) invariante de instância

- Verdadeira se e só se a instância for válida e coerente
- Tem de ser verdadeira:
  - Após construção (através da interface pública)
  - Após qualquer alteração (através da interface pública)

Invariante

```
public class Student {

public Student(String id, String name) {

     this.id = id;

    this.name = name;

    if (!invariant())

          throw new IllegalStateException();

}

private boolean invariant() {

    // id and name not null and not empty

    return id != null && name != null && id.trim().length().size() != 0
    && name.trim().length().size() != 0;

}
```

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