Exceções e ficheiros

Exceções

O lançamento de *exceções* pode ser utilizado como um mecanismo para interromper a execução normal de um **método**, caso o objeto tenha sido utilizado de forma incorreta:

- Invocação de uma operação com argumentos inválidos

- Sequência de invocações inválida

As exceções elas próprias **são objetos** (com atributos, operações, e construtores)

Tipos de Exceção

(para validar argumentos)

IllegalArgumentException
adequada quando um argumento inválido é utilizado na invocação de uma operação

NullPointerException
adequada quando é passada uma referência null não permitida como argumento

IllegalStateException
adequada quando é invocada uma operação não permitida no estado atual do objeto

...

IllegalArgumentException

(validação de argumentos)

```
static int[] naturals(int n) {
   if(n < 0)
     throw new IllegalArgumentException("Must be positive:" + n);

   int[] nat = new int[n];

   //...
}
```

throw funciona como um "return com erro", sendo que o método termina quando é lançada uma exceção

NullPointerException

(validação de argumentos)

```
static int[] copy(int[] v) {
   if(v == null)
     throw new NullPointerException("Argument cannot be null");

   int[] copy = new int[v.length];

   //…
}
```

IllegalStateException

(validação de estado)

```
class IntSet {
   int[] elements = new int[0];

   int max() {
    if(elements.length == 0)
     throw new IllegalStateException("set is empty");

    //...
   }
}
```

Tipos de Exceção

(resultantes de erros de programação)

ArithmeticException
lançada quando é feita uma divisão por zero

NullPointerException
lançada quando é feito um acesso numa referência null

NumberFormatException
adequada quando é invocada uma operação não permitida no estado atual do objeto

...

Tratamento de exceções

```
String s = …
try {
   int n = Integer.parseInt(s); // String -> int
   System.out.println(s + " is a number");
}
catch(NumberFormatException e) {
   System.out.println(s + " is not a number");
}
```

a execução do bloco do *try* é interrompida e prossegue no bloco do *catch* assim que ocorre a exceção

Tipos de Exceção

(checked)

FileNotFoundException

lançada quando um ficheiro não é encontrado ou
não é permitido escrever na sua localização

...

as exceções *checked* obrigam o programa a
tratar da ocorrência com try-catch

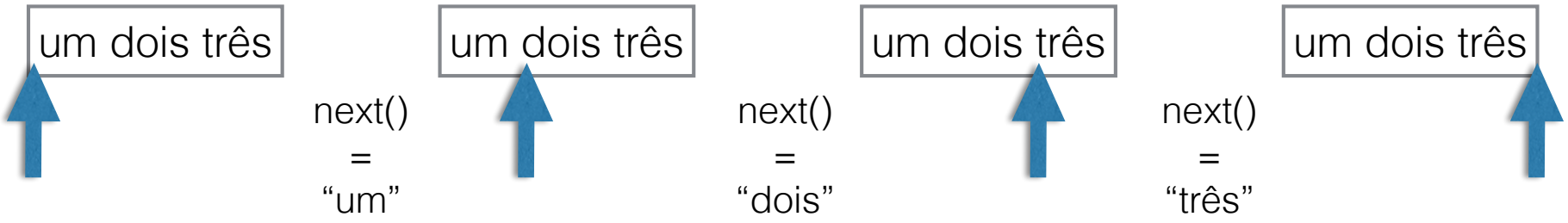
Escrita de ficheiros

```
import java.io.PrintWriter;
import java.io.File;
import java.io.FileNotFoundException;
```

```
try {
   PrintWriter writer = new PrintWriter(new File("data.txt"));
   for(char c = 'A'; c <= 'Z'; c++) {
      writer.println(c);
   }
   writer.close();
}
catch(FileNotFoundException e) {
   System.out.println("o ficheiro data.txt não pode ser escrito");
}
```

Tokenização com Scanner

```
int words = 0;
Scanner scanner = new Scanner("um dois três");
while(scanner.hasNext()) {
   String word = scanner.next();
   words++;
}
scanner.close();
```



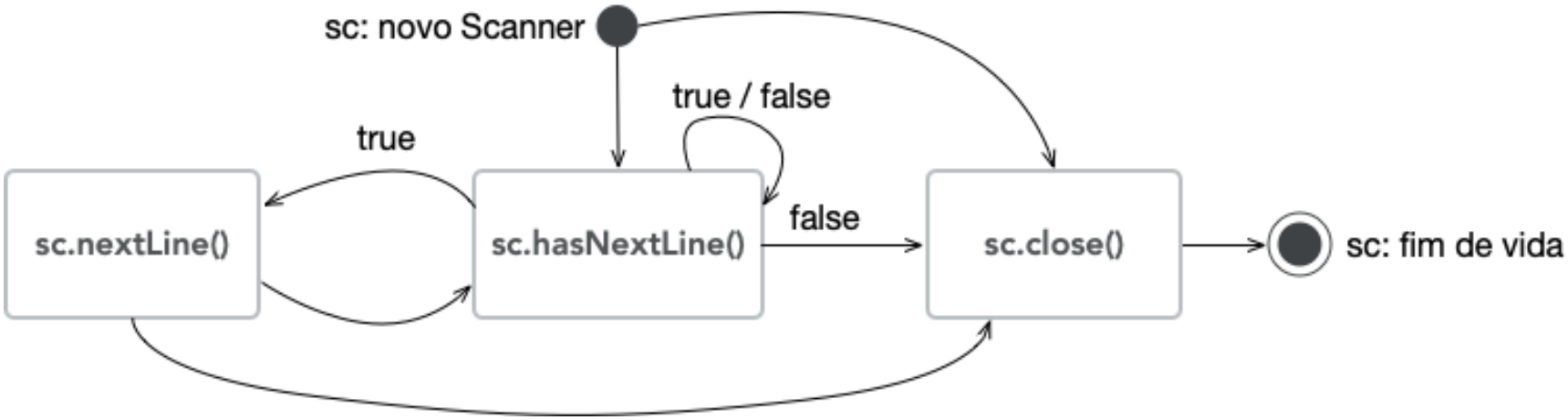
Scanner

(operações principais)

- hasNext()
  próxima palavra
- hasNextInt()
- hasNextDouble()
- hasNextBoolean()
- hasNextLine()
  próxima linha
- next()
  próxima palavra
- nextInt()
- nextDouble()
- nextBoolean()
- nextLine()
  próxima linha

Scanner

(protocolo)



ocorrerão exceções se não for respeitado o protocolo de sequência de invocações

(exemplo para leitura de ficheiro linha por linha)

Leitura de ficheiros

```
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;
```

```
int lines = 0;
try {
   Scanner scanner = new Scanner(new File("data.txt"));
   while(scanner.hasNextLine()) {
      String line = scanner.nextLine();
      lines++;
   }
   scanner.close();
}
catch(FileNotFoundException e) {
   System.out.println("ficheiro data.txt não encontrado");
}
```

A RETER

- Exceções
  - Lançamento (throw)
  - Tratamento (try-catch)
- Leitura de ficheiros
  - java.util.Scanner
- Escrita de ficheiros
  - java.io.PrintWriter