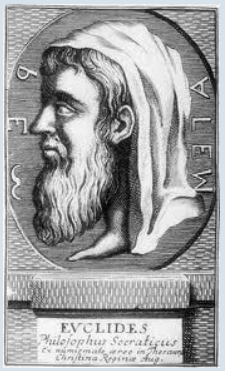
Repetição de Comandos

Fernanda Barbosa
Fernando Birra
Luís Caires
Armanda Rodrigues

A nossa biblioteca

- De modo a praticar os ciclos vamos fazer uma biblioteca com operações matemáticas diversas.
- Para isso vamos criar uma classe `Mathematics`, onde **todos os métodos são `static`** (estes métodos denominam-se “métodos de classe”).
- Note que **esta classe não tem construtores**, pois nunca vamos criar objectos desta classe. É como a classe `Math` do Java.
- De igual modo, **esta classe também não tem variáveis de instância**. Repare, uma variável de instância é uma variável que pertence a uma instância, ou seja, a um objecto. Se nunca vamos criar um objecto desta classe, não faria sentido criar variáveis de instância.



Números Primos

- Um número inteiro positivo é primo se tiver apenas dois divisores distintos
- Por exemplo, 3, 11 e 17 são números primos:
  - 3 é divisível apenas por 1 e 3
  - 11 é divisível apenas por 1 e 11
  - 17 é divisível apenas por 1 e 17

Números primos

- Crie um método na classe “Mathematics” que indica se um dado número é primo

```
public static boolean isPrime(int n)
```

Indica se o número n é primo

- Teste o método implementado, num programa principal e verifique que o método `isPrime` se comporta como o esperado. Por exemplo, experimente fazer um programa principal em que o utilizador vai dando valores inteiros positivos, que vão sendo testados. Para terminar, o utilizador dá o número `-1`.

Números primos

- O método implica a divisão inteira do número *n* por todos os números inteiros menores que *n* e maiores que um e a verificação do resto da operação

- Note que a operação de teste de divisibilidade deve ser repetida um certo número de vezes
  - O número de repetições depende do valor n
  - É necessário utilizar um comando de **repetição** ou **iteração** de comandos

Números primos

```
n % n-1   != 0
n % n-2   != 0
n % n-3   != 0
...
n % 2     != 0
```




Números primos

```
n % v  != 0
n % v  != 0
n % v  != 0
...
n % v  != 0
```




Em cada passo

```
v--;
```

Instrução `while`

```
n % v  != 0
n % v  != 0
n % v  != 0
...
n % v  != 0
```

Em cada passo

```
v--;
```

```
while (condicao)
    instrucao
```

Esta condição é suficiente? Não

```
while ( n % v !=0)
    v--;
```

Instrução `while`

```
n % v  != 0
n % v  != 0
n % v  != 0
...
n % v  != 0
```




Em cada passo

```
v--;
```

```
while (condicao)
    instrucao
```

As duas condições parecem redundantes mas preenchem dois requisitos

```
while ((v > 1) && ( (n % v) != 0))
    v--;
```

Instrução `while`

Condição booleana de manutenção da execução do ciclo

```
while ((v > 1) && ( (n % v) != 0))
      v--;
```

- (v > 1) – executar a divisão inteira de n por todos o números entre 1 e n.
- (n % v) != 0 – enquanto não for encontrado um divisor

Números primos

```
public static boolean isPrime(int n){
   int v = n-1;
   while ((v > 1) && ( (n % v) != 0)) {
          v--;
   }
   return (v==1);
}
```

As duas condições parecem redundantes mas preenchem os dois requisitos apresentados

Análise de casos

- `n <= 0` – não faz sentido pois o método só deve ser aplicado a números inteiros positivos
- `n = 1` → `v = 0` – O ciclo não é executado pois `v` não se encontra dentro dos valores a serem testados como divisores
  - Desta forma, porque `v != 1`, o método devolve `false` → 1 não é primo
- `n = 2` → `v = 1` – O ciclo não é executado pois v não se encontra dentro dos valores a serem testados como divisores
  - `v == 1`, devolve `true` → 2 é primo
- `n > 2`
  - Se o número for primo, o ciclo é executado até `v == 1`, o método devolve `true` → o número é primo
  - Se o número não for primo, o ciclo termina quando `v` igual ao 1º divisor de `n`, assim `v!=1,` o método devolve `false` → o número não é primo

Análise de casos

Esta condição não é suficiente porque, se `n=1` o programa aborta, pois tenta-se executar uma divisão por zero

```
while ( n % v !=0)
    v--;
```

Class **Main**

```
import java.util.Scanner;

public class Main {

  public static void main(String[] args) {
    Scanner in=new Scanner(System.in);

    int pr;

    System.out.print("Introduza um número para verificar se e primo: ");
    pr=in.nextInt();
    if (Mathematics.isPrime(pr))
       System.out.println("É primo.");
    else
       System.out.println("Não é primo.");
    System.out.println("Goodbye!!!");
  }
}
```

Vamos utilizar o -1 (que não é positivo) como um valor sentinela que determina o fim do input

Class **Main**

```
import java.util.Scanner;

public class Main {

  public static void main(String[] args) {
    Scanner in=new Scanner(System.in);

    int pr;

    System.out.print("Introduza um número para verificar se e primo (-1 para terminar): ");
    pr=in.nextInt();
    while (pr != -1){
       if (Mathematics.isPrime(pr))
              System.out.println("É primo.");
       else
              System.out.println("Não é primo.");


    }
    System.out.println("Goodbye!!!");
  }
}
```

Estas duas linhas têm de ser repetidas em cada passo para:

- Informar o utilizador sobre o input;
- Guardar o valor introduzido.

Class **Main**

```
import java.util.Scanner;

public class Main {

  public static void main(String[] args) {
    Scanner in=new Scanner(System.in);

    int pr;

    System.out.print("Introduza um número para verificar se e primo (-1 para terminar): ");
    pr=in.nextInt();
    while (pr != -1){
       if (Mathematics.isPrime(pr))
              System.out.println("É primo.");
       else
              System.out.println("Não é primo.");
       System.out.print("Introduza um número para verificar se e primo (-1 para terminar): ");
       pr=in.nextInt();
    }
    System.out.println("Goodbye!!!");
  }
}
```

Estas duas linhas têm de ser repetidas em cada passo para:

- Informar o utilizador sobre o input;
- Guardar o valor introduzido.

Class **Main**

```
import java.util.Scanner;

public class Main {

  public static void main(String[] args) {
    Scanner in=new Scanner(System.in);

    int pr;

    System.out.print("Introduza um número para verificar se e primo (-1 para terminar): ");
    pr=in.nextInt();
    while (pr != -1){
       if (Mathematics.isPrime(pr))
              System.out.println("É primo.");
       else
              System.out.println("Não é primo.");
       System.out.print("Introduza um número para verificar se e primo (-1 para terminar): ");
       pr=in.nextInt();
    }
    System.out.println("Goodbye!!!");
  }
}
```

Sequência de Fibonacci

- Em 1225, Leonardo Fibonacci participou numa competição matemática em Pisa onde surgiu a seguinte questão:
- Começando apenas com um par de coelhos, se em cada mês cada par produtivo de coelhos produzir um novo par de coelhos (o qual se torna produtivo apenas um mês de depois de nascer), quantos pares de coelhos existirão ao fim de n meses?

Sequência de Fibonacci

- A resposta é a chamada sequência de Fibonacci:
  fib(0)=0, fib(1)=1,fib(2)= 1,fib(3)= 2,
  fib(4)=3, fib(5)=5,fib(6)= 8, ...

- Esta sequência é muito fácil de obter: cada número da sequência é a soma dos dois números anteriores!

Sequência de Fibonacci

- Crie um método na classe `Mathematics` que calcule o valor de sequência de Fibonacci para um determinado número `n` (ou seja, quantos coelhos existem passados `n` meses).

```
public static int fibonacci(int n)
```

Calcula o valor da sequência de Fibonacci na posição n. Assuma que o n é um inteiro maior ou igual a zero.

- Crie depois um programa principal que imprima os primeiros 10 valores da sequência de Fibonacci.

Números perfeitos

- Um número perfeito é um inteiro positivo cuja soma dos seus divisores próprios é o próprio número
- Os divisores próprios de um número são os divisores que são diferentes do próprio número
- Por exemplo, **6** é um número perfeito:

  1, **2** e **3** são os divisores próprios de **6**

  6 = 1 + 2 + 3

Números perfeitos

- Crie um método na classe `Mathematics` que determina se um número `n` é um número perfeito

```
public static boolean isPerfectNumber(int n)
```

Indica se o número **n** é um número perfeito.

- Crie depois um programa principal que imprima todos os números perfeitos menores que **10000**.

Máximo Divisor Comum

- O Algoritmo de Euclides para cálculo do Máximo Divisor Comum (**mdc**) é o seguinte:
- Dados dois números naturais **a** e **b**, em que pelo menos um deles é diferente de zero
  - Ver se **b** é zero
    - Se sim, **a** é o **mdc**;
    - Caso contrário, repetir o processo usando respectivamente **b** e **a % b**;

MDC(a,0)=a
MDC(a,b)=MDC(b,a%b)

- Por exemplo MDC(8,12) = 4

Máximo Divisor Comum

- Crie um método na classe `Mathematics` que calcula o MDC entre dois números naturais `a` e `b`, utilizando o algoritmo de Euclides

```
public static int mdc(int a, int b)
```

Devolve o mdc dos números a e b.

- Teste o método num programa principal que vá pedindo ao utilizador pares de números naturais e devolvendo, para cada par, o seu MDC. O programa deve terminar quando o utilizador tentar indicar números menores ou iguais a 0.

Máximo Divisor Comum

- O Algoritmo de Euclides para cálculo do Máximo Divisor Comum (**mdc**) é o seguinte:
- Dados dois números naturais **a** e **b**, em que pelo menos um deles é diferente de zero
  - Ver se **b** é zero
    - Se sim, **a** é o **mdc**;
    - Caso contrário, repetir o processo usando respectivamente **b** e **a % b**;

MDC(a,0)=a
MDC(a,b)=MDC(b,a%b)

Em pseudo-código →

```
function gcd(a, b)
    while b ≠ 0
        t := b
        b := a mod b
        a := t
    return a
```