Programando em Java
(Classes Simples e Tipos de Dados Básicos)

Material didáctico elaborado pelas diferentes equipas de Introdução à Programação

Luís Caires (Responsável), Armanda Rodrigues, António Ravara, Carla Ferreira, Fernanda Barbosa, Fernando Birra, Jácome Cunha, João Araújo, Miguel Goulão, Miguel Pessoa Monteiro, e Sofia Cavaco.

Mestrado Integrado em Engenharia Informática FCT UNL

Alguns Programas Simples

- Neste capítulo, vamos programar em Java um conjunto de classes simples.

- No caminho, serão introduzidas várias noções novas e importantes:
  - Operações com valores inteiros, reais e lógicos
  - Parâmetros de construtores e parâmetros de métodos
  - Definição de (nomes para) constantes
  - Definição nas classes de múltiplos construtores
  - Chamada de outros métodos dentro do método de um objecto

Rectângulo

Rectângulo

- **Objectivo**
  - Manipular rectângulos no plano XY.
- **Descrição**
  - Qualquer rectângulo está no plano XY, e tem os seus lados paralelos aos eixos. As coordenadas que o caracterizam devem ser valores reais de precisão muito grande.
- **Funcionalidades**
  - Pode-se deslocar o rectângulo ao longo de um dado vector <dx,dy>, e rodar o rectângulo 90º para a direita.
  - Deve ser sempre possível consultar a abcissa dos cantos esquerdos e direitos e a ordenada dos cantos superiores e inferiores do rectângulo. Para além disso, deve-se consultar também a altura, a largura, o perímetro, a área, e a ordenada e abcissa do centro do rectângulo.
  - Deve ser sempre possível verificar se um dado ponto ou um dado rectângulo no plano XY está no interior do rectângulo.

...

Rectângulo

- Se não indicarmos nada, o rectâgulo será um quadrado de lado 1, com centro na origem do plano XY. Em alternativa, pode-se:
  - Indicar:
    - A abcissa do canto superior esquerdo (left)
    - A ordenada do canto superior esquerdo (top)
    - A abcissa do canto inferior direito (right)
    - A ordenada do canto inferior direito (bottom)
  - Indicar
    - A abcissa do canto superior esquerdo
    - A ordenada do canto superior esquerdo
    - A dimensão do lado (length) – neste caso o rectângulo é um quadrado!

- **Interacção com o utilizador**
  - Após criar rectângulos, pode invocar as operações.

Rectângulo

- **Que objecto definir?**
  - Um rectângulo (classe Rect)
- **Interface:**

As coordenadas do rectângulo devem ser valores reais de precisão muito grande. → tipo `double`

`double` `getLeft()`

consulta a abcissa dos cantos esquerdos

`double` `getRight()`

consulta a abcissa dos cantos direitos

`double` `getTop()`

consulta a ordenada dos cantos superiores

`double` `getBottom()`

consulta a ordenada dos cantos inferiores

Rectângulo

double `getXCenter()`

consulta a abcissa do centro do rectângulo

double `getYCenter()`

consulta a ordenada do centro do rectângulo

double `getWidth()`

consulta (calcula) a largura do rectângulo

double `getHeight()`

consulta a altura do rectângulo

double `getPerimeter()`

consulta o perímetro do rectângulo

double `getArea()`

consulta a área do rectângulo

Rectângulo

boolean `isPointInRect`**(double** `x`**, double** `y`**)**

verifica se o ponto (`x`,`y`) está dentro do rectângulo

O parâmetro do método é um objecto rectângulo.

boolean `isRectInRect`**(Rect** `r`**)**

verifica se o rectângulo `r` está dentro do rectângulo

void `translate`**(double** `dx`**, double** `dy`**)**

desloca o rectângulo ao longo do vector <`dx`,`dy`>

void `turn`**()**

roda o rectângulo 90º para a direita

Múltiplos Construtores

```
public class Rect {
    ... ; /* Definição de constantes e variáveis de instância */
    /* Um quadrado de lado 1, com centro na origem do plano */
    public Rect() { ...; }
    /* Pre: (left < right) && (top > bottom) */
    public Rect(double left, double top, double right, double bottom)
    { ... ; }
    /* Pre: length > 0 */
    public Rect(double left, double top, double length)
    { ... ; }
    ...  /* Definição dos restantes métodos da classe */
}
```

Múltiplos Construtores

```
public class Rect {

    ... ;

    public Rect() { ...; }

    public Rect(double left, double top, double right, double bottom)

    { ... ; }

    public Rect(double left, double top, double length)

    { ... ; }

    ...

}
```

- É possível definir na mesma classe quantos construtores quisermos, para cobrir as várias maneiras pretendidas de criar os objectos.
- Todos os construtores têm o mesmo nome (o nome da classe) mas são distinguidos pelos parâmetros que possuem

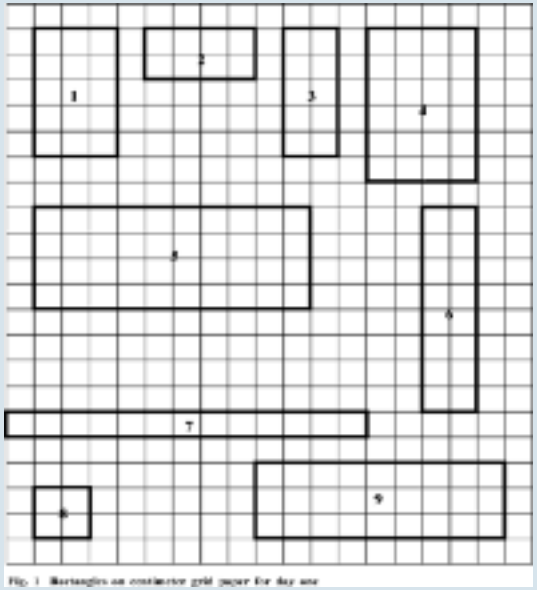
Rectângulo

```
Rect r = new Rect(10,50,120,20);
Rect r1 = new Rect();
r.getArea();
// 3300.0   (double)
r.getPerimeter();
// 280.0   (double)
r.isPointInRect(30,40);
// true   (boolean)
r.isPointInRect(130,40);
// false   (boolean)
r.translate(10,-30);
r.isPointInRect(130,40);
// false   (boolean)
r.isRectInRect(r1);
// false   (boolean)
```

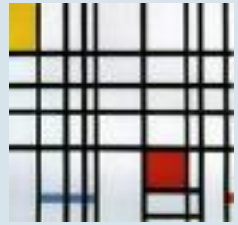
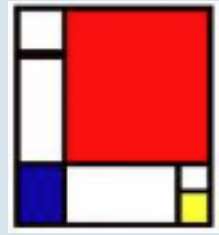
```
r.isPointInRect(30,40);
// false   (boolean)
r.turn();
r.getLeft();
// 60.0   (double)
r.getTop();
// 60.0   (double)
r.getRight();
// 90.0   (double)
r.getBottom();
// -50.0   (double)
```

Rectângulo

- Defina em Java uma classe Rect cujos objectos representam rectângulos num plano.
- Programe a sua classe no Eclipse.
- Teste um (ou vários) objectos Rect, e verifique se se comportam como esperado.



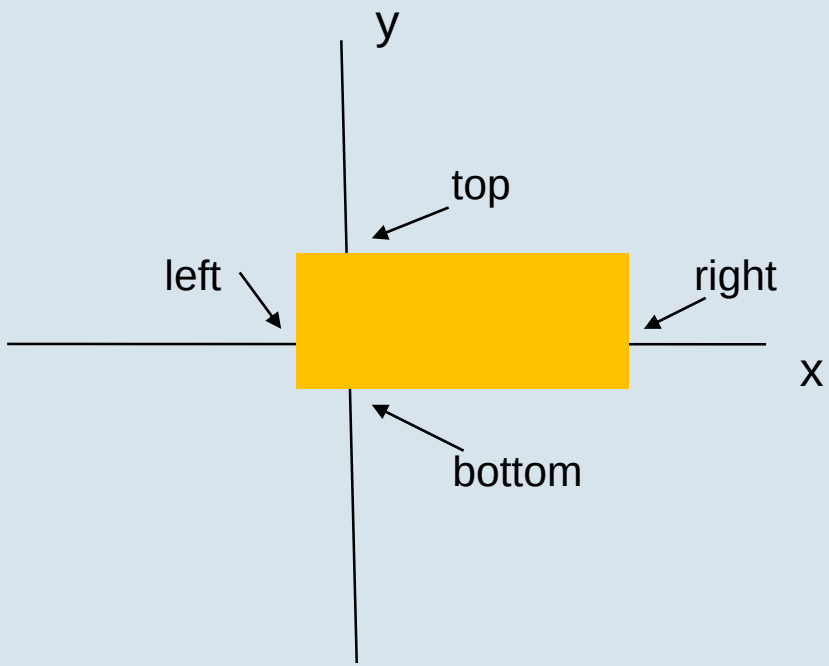


Programando o Rectângulo

```
public class Rect {
    private double left, top, right, bottom;
    ...
}
```

Variáveis de instância



Programando o Rectângulo

```
public class Rect {
   private double left, top, right, bottom;
   ...
   public Rect(){
     left = …
     top = …
     right = …
     bottom = …
   }
   ...
}
```

Variáveis de instância

Necessitamos de ter várias instruções para poder inicializar todas as variáveis de instância do objecto

construtores

Composição sequencial: ideia central

- Temos visto desde o inicio que um programa é uma sequência de ordens/instruções (recorde o exemplo do robot, do operador, etc)
- Nos exemplos de interacção com classes definidos no Eclipse, usaram-se sequências de ordens: primeiro criar objecto, depois chamar modificador e, finalmente, fazer consulta (para ver estado)
- Em Java usa-se o símbolo '`;`' para encadear instruções: é um operador binário que, dadas duas intruções `c1` e `c2`, constroi uma nova instrução que é a sequencialização `c1;c2`
- O comportamento da composição sequencial `c1;c2` é o seguinte: primeiro executa `c1` e, quando este acabar, executa `c2`

```
BankAccount b = new BankAccount();
b.deposit(10000);
```

Composição sequencial: onde surge, numa classe Java?

- Até agora, nos exemplos apresentados, o corpo de um metodo é simplesmente uma instrução (atribuição ou return), dita básica
- É útil fazer composições destas instruções básicas, por exemplo para alterar mais do que uma variável de instância – vamos ver corpos de metodos compostos
- Vimos também que uma classe Java tem vários "ingredientes": *constantes*, *variáveis*, *construtores* e *métodos*; a sua declaração **em sequência** recorre ao operador de composição sequencial, compondo assim as várias intruções para se obter o corpo da classe

Programando o Rectângulo

```
public class Rect {
   private double left, top, right, bottom;
   ...
   public Rect(){
     left = …;
     top = …;
     right = …;
     bottom = …;
   }
   ...
}
```

Variáveis de instância

Operador de sequência “;”

construtores

Programando o Rectângulo

```
public class Rect {
   private double left, top, right, bottom;
   public static final double DEFAULT_VALUE = 0.5;
   public Rect(){
     left = - DEFAULT_VALUE;
     top = DEFAULT_VALUE;
     right = DEFAULT_VALUE;
     bottom = - DEFAULT_VALUE;
   }
   ...
}
```

Constante

construtores

Declaração de Constantes

- Em certos problemas é conveniente definir o valor de certas **constantes globais**
- Porque dizemos “constantes globais”? Porque:
  - Globais, pois são usadas por **todos** os objectos de uma certa classe
  - Constantes, pois o seu valor nunca poderá ser alterado (é fixo)

- Para declarar numa classe uma constante, utiliza-se a forma

```
public static final tipo constantIdentifier = expr ;
```

A expressão ***expr*** só pode usar valores constantes. Por exemplo:

```
public static final int HOURS_IN_DAY = 24;
public static final int HOUR_TO_MINUTES = 60;
public static final int MINUTES_IN_DAY = HOURS_IN_DAY * HOUR_TO_MINUTES;
```

Declaração de Constantes

- Note que **não é possível reafectar** o valor associado a uma constante (experimente ver o que acontece no Eclipse)
- Tal como as variáveis, as constantes são referidas por um identificador, escolhido pelo programador
- Por convenção, é costume usar-se um identificador iniciado por maiúscula, ou mesmo constituído apenas por maiúsculas
- Por exemplo:
  - **`MAXSIZE`**
  - **`MONTHS_IN_YEAR`**

```
public static final int MONTHS_IN_YEAR = 12;
```

Declaração de constantes

`public` especifica que o valor desta constante pode ser usado fora desta classe.

`static` especifica que o valor da constante é partilhado por todos os objectos da classe.

`final` especifica que estamos a declarar uma constante.

```
public static final tipo constantIdentifier = expr ;
```

O valor guardado na constante é do tipo ***tipo***.

A constante é identificada por ***constantIdentifier***.

A constante assume o valor de ***expr***.

Programando o Rectângulo

```
public class Rect {
   private double left, top, right, bottom;
   ...
   public Rect(double left,double top,
               double right,double bottom){
     this.left = left;
     this.top = …;
     this.right = …;
     this.bottom = …;
   }
   ...
}
```

Variáveis de instância

Os parâmetros do construtor e as variáveis de instância da classe tem o mesmo nome. Como diferenciar o parâmetro **`left`** e a variável de instância **`left`**?

O "pronome" **`this`** refere o próprio objecto. Usa-se como prefixo na variável de instância, distinguindo-a do parâmetro.

this: referência ao objecto corrente

construtores

Programando o Rectângulo

```
public class Rect {
   private double left, top, right, bottom;
   ...
   public double getXCenter(){
     return (left+right)/2;
   }
   public double getWidth(){
     return right - left;
   }
   ...
}
```

métodos

Chamada dos Próprios Métodos

- Nos métodos de um objecto é possível invocar (chamar / utilizar) operações também definidas no mesmo objecto.
- Por exemplo, no caso da classe `Rect` podemos definir:

```
public double getArea() {
return this.getWidth() * this.getHeight();
}
```

retorna a multiplicacao dos valores **retornados** pelas chamadas aos métodos `getWidth()` e `getHeight()`

- Note que neste caso usamos o "pronome" **`this`** para referir o próprio objecto ao qual a operação deve ser aplicada.
- No exemplo, quando um objecto da classe `Rect` processa a operação `getArea()` vai calcular a área a partir das operações `getWidth()` e `getHeight()`, aplicadas a "si próprio" (**`this`**).
- **`this`** significa "eu" ou "a mim". É uma palavra reservada em Java.
- É sempre necessário indicar qual é o objecto ao qual o método deve ser aplicado, seja ele **`this`** ou outro (veremos no método `rectInRect`).

Programando o Rectângulo

```
public class Rect {

   private double left, top, right, bottom;

   ...

   public boolean isPointInRect(double x, double y){

     return left <= x && right >= x

         && bottom <= y && top >= y;

   }

   ...

}
```

métodos

Programando o Rectângulo

```
public class Rect {
```

O parâmetro do método é um objecto rectângulo, que já foi criado.

```
   ...

   public boolean isRectInRect(Rect r){

     return r.getLeft() >= left && r.getRight() <= right

      && r.getBottom() >= bottom && r.getTop() <= top ;

   }

   ...

}
```

Invocação dos métodos no objecto rectângulo passado como parâmetro (`r`)

Programando o Rectângulo

```
public class Rect {
    …
    public boolean isRectInRect(Rect r){
      return r.getLeft() >= left && r.getRight() <= right
       && r.getBottom() >= bottom && r.getTop() <= top ;
    }
    .
}
```

O parâmetro do método é um objecto rectângulo, que já foi criado.

Quando temos uma variável para guardar a referência de um objecto (caso `r`), como saber que o objecto ainda não foi criado?

Com que valor inicializamos? Algum dos valores dos tipos vistos serve?

- Em Java, há um valor especial → **`null`**.

Exemplos de declarações e inicializações de variáveis

```
Rect r = null; // Ainda não existe o objecto
Rect r = new Rect(); // O objecto é criado e a sua referência guardada
                     // na variável r
```

Rectângulo

- Enriquece-se a interface com um contrato de utilização.
- Na interface coloca-se a pré-condição:

```
boolean isRectInRect(Rect r)
```

verifica se o rectângulo `r` está dentro do rectângulo

Pre: r != null

O parâmetro r é um objecto rectângulo, que já foi criado.

Programando o Rectângulo

```
public class Rect {
    private double left, top, right, bottom;
    ...
    public void turn(){
      double cx = this.getXCenter();
      double cy = this.getYCenter();
      double halfWidth = this.getWidth() / 2;
      double halfHeight = this.getHeight() / 2;
      left = cx - halfHeight;
      right = cx + halfHeight;
      bottom = cy - halfWidth;
      top = cy + halfWidth;
    }
    ...
}
```

Variáveis locais ao método

A declaração de variáveis locais é bastante parecida com a declaração de variáveis de instância, mas acontece **dentro** de um método.

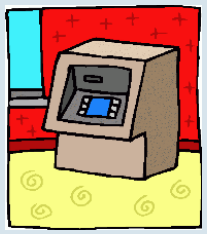
DescritorDeTipo Identificador

Variáveis locais

- Por vezes, para implementar um método, temos de salvaguardar temporariamente alguns valores, para garantir que os resultados finais da execução do método são correctos
- Dentro de um método, é possível definir **variáveis locais**, também conhecidas como **variáveis temporárias**
- A existência das variáveis locais é confinada entre a sua declaração e o fim da execução do método:
  - Quando o método terminar, a variável temporária deixa de existir
  - Se o método for invocado de novo, a variável é criada novamente, ou seja, esta variável não guarda memória do seu valor entre chamadas sucessivas ao método em que está definida
- Não confunda variáveis locais com variáveis de instância!
  - As variáveis locais existem apenas durante a execução do método
  - As variáveis de instância existem durante toda a vida do objecto

Alguns Programas Simples

- Vimos como programar em Java várias classes simples.



- Foram introduzidos vários aspectos importantes:
  - Operações com valores inteiros, reais e lógicos
  - Parâmetros de construtores e parâmetros de métodos
  - Definição de (nomes para) constantes
  - Definição nas classes de múltiplos construtores
  - Chamada de métodos dentro do método de um objecto
- Certifique-se de que **compreendeu bem** cada aspecto, não só no contexto do problema, mas também em geral