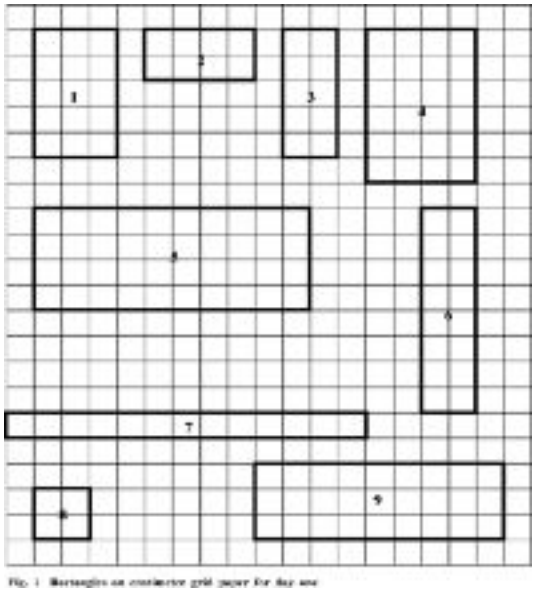
Programando em Java
(Condições e Decisões)

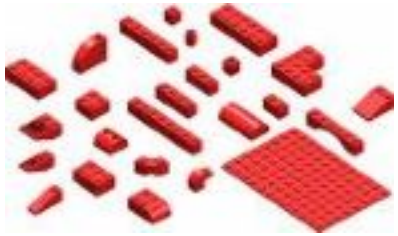
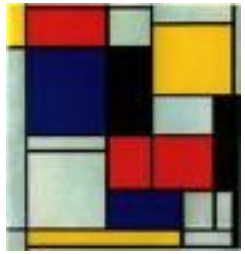
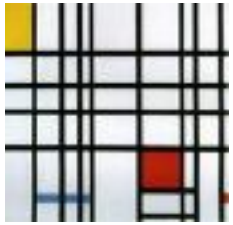
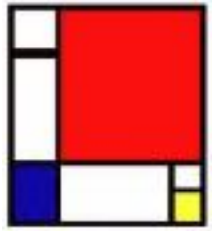
Mestrado Integrado em Engenharia Informática FCT UNL
http://ctp.di.fct.unl.pt/miei/ip/

Corpo Docente 2020/2021
António Ravara, Artur Miguel Dias, Bernardo Toninho,
Ema Vieira, Inês Fernandes, Margarida Mamede,
Miguel Monteiro, Rui Nóbrega

Problema do Rectângulo (1)

Dados dois rectângulos num plano, cujos lados são paralelos aos eixos, pretende-se calcular os seus perímetros e o perímetro do menor rectângulo que os contém (chamado *invólucro*), detectando se o invólucro é um quadrado.

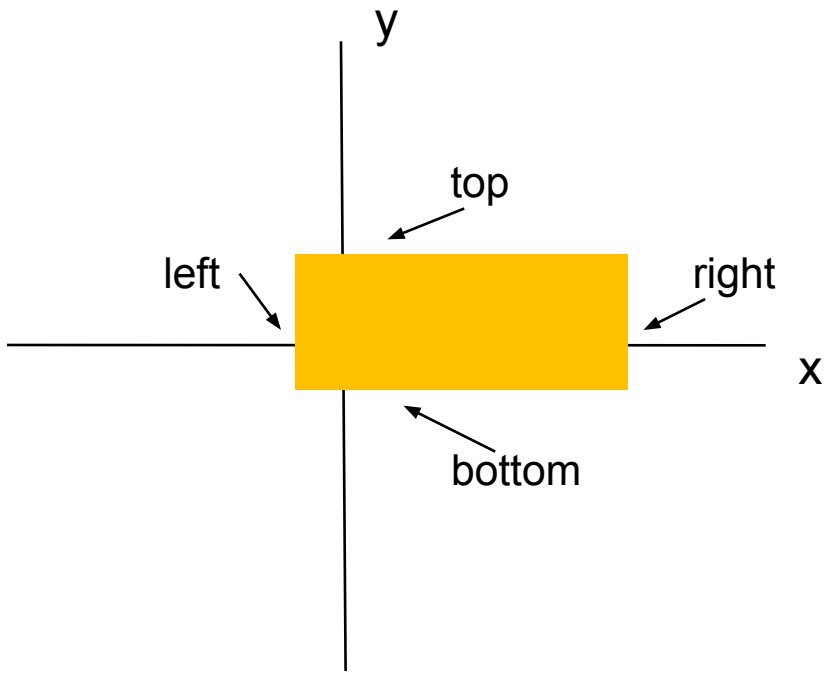


Problema do Rectângulo (2)

- Qualquer rectângulo está no plano XY e tem os seus lados paralelos aos eixos. As coordenadas que o caracterizam (*left*, *top*, *right* e *bottom*) devem ser valores reais de precisão muito grande.
- O programa lê duas linhas, cada uma com as coordenadas de um rectângulo, *left*, *top*, *right* e *bottom*, por esta ordem e separadas por um espaço.

- O programa escreve três linhas, cada uma com um número real que representa:
  - o perímetro do 1º rectângulo;
  - o perímetro do 2º rectângulo;
  - o perímetro do invólucro.
- Se o invólucro for um quadrado, o programa escreve mais uma linha com: "`The hull is a square`" (sem aspas).



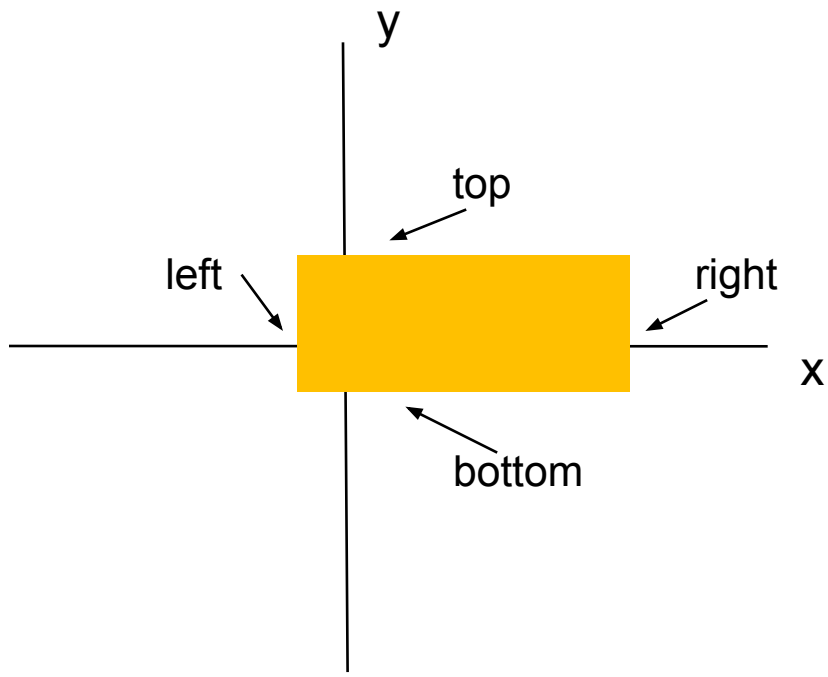
Problema do Rectângulo (3)

Input 1

```
1 4 3 2
2 5 4 1
```

Output 1

```
8.0
12.0
14.0
```



Input 2

```
-3 0.5 0 -2.5
0.4 5.0 4.5 -0.4
```

Output 2

```
12.0
19.0
30.0
The hull is a square
```

Leitura de dados

- `System.in` é um objecto pré-definido utilizado para fazer leitura de dados a partir do *standard input* (o teclado, por omissão)
- No entanto, o `System.in` só lê um carácter de cada vez
- No Java, a classe de biblioteca `Scanner` foi criada para fazer a leitura a partir do teclado de uma forma simples
- O Scanner é inicializado estabelecendo uma ligação a uma *stream* de entrada de dados (neste caso, o `System.in`)
  - Indicamos a *stream* a que queremos aceder no construtor

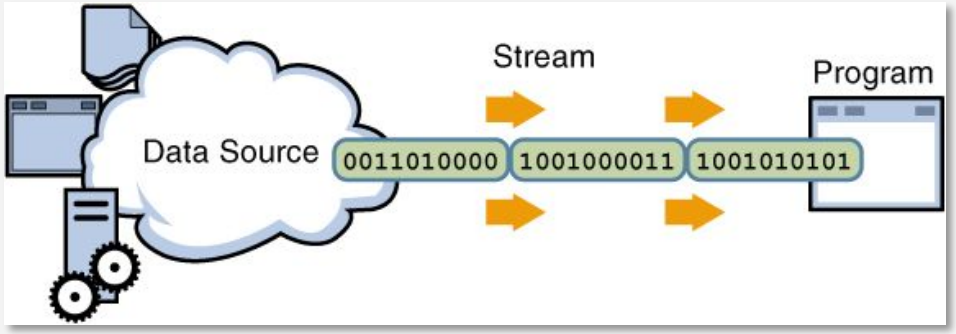
```
Scanner in = new Scanner(System.in);
```

  - Quando já não necessitamos do `Scanner`, devemos libertar a stream usando o método `close()` do `Scanner`, que encerra o acesso desse `Scanner` à *stream* indicada no construtor

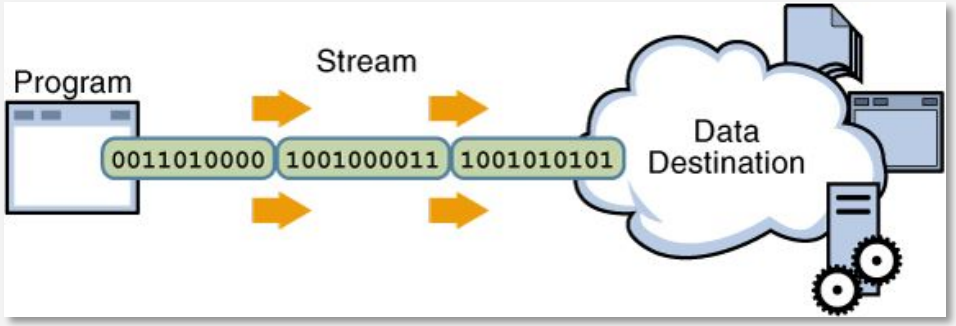
```
in.close();
```

System.*in* e System.*out*

- Os objectos `System.in` e `System.out` representam duas *streams* de entrada e saída de dados do programa, respectivamente
- Uma *stream* é um fluxo de dados de onde podemos ler ou onde podemos escrever informação.
  - `System.in` é a stream de entrada de dados usada por omissão no sistema



  - `System.out` é a stream de saída de dados usada por omissão no sistema



Disciplina de uso do `Scanner`

- Como a classe `Scanner` não faz parte do *core* do Java, temos que a importar para a usar

```
//incluir no inicio do ficheiro da classe que vai usar
import java.util.Scanner;
```

- Antes de ler ou escrever dados numa *stream*, temos de lhe aceder; depois devemos "libertar" o acesso

```
Scanner in = new Scanner(System.in); // Scanner acede a System.in
System.out.println("Quantidade: ");  // Informacao pretendida
...
in.close(); // Ja nao necessitamos do Scanner, devemos liberta-lo
```

Operações da classe Scanner

- **`int`** `nextInt()` – lê um inteiro (`int`)
- **`double`** `nextDouble()` – lê um real (`double`)
- `String next()` – lê uma `String` sem espaços
- `String nextLine()` – lê uma linha até ao fim (`String`)
- **`void`** `close()` – liberta os recursos do `Scanner`, desactivando-o

```
Scanner in = new Scanner(System.in); // Scanner acede a System.in
System.out.println("Quantidade: ");  // Informacao pretendida
...
in.close(); // Ja nao necessitamos do Scanner, devemos liberta-lo
```

Estude a classe `Scanner` no pacote `java.util`

Construção da classe `Main` (1)

```
import java.util.Scanner;
public class Main {
    public static void main( String[] args ) {
        Scanner input = new Scanner(System.in);
        double left = input.nextDouble();
        double top = input.nextDouble();
        double right = input.nextDouble();
        double bottom = input.nextDouble();
        input.nextLine();
        Rect rect1 = new Rect(left, top, right, bottom);
        left = input.nextDouble();
        top = input.nextDouble();
        right = input.nextDouble();
        bottom = input.nextDouble();
        input.nextLine();
        Rect rect2 = new Rect(left, top, right, bottom);
        input.close();
        //Continua no slide seguinte
    }
}
```

Temos que indicar onde está definida a classe `Scanner`

Temos de ler tudo o que está para além dos números (nomeadamente a mudança de linha)

Devemos encerrar o `Scanner input`, libertando assim os seus recursos, quando terminamos a leitura de dados.

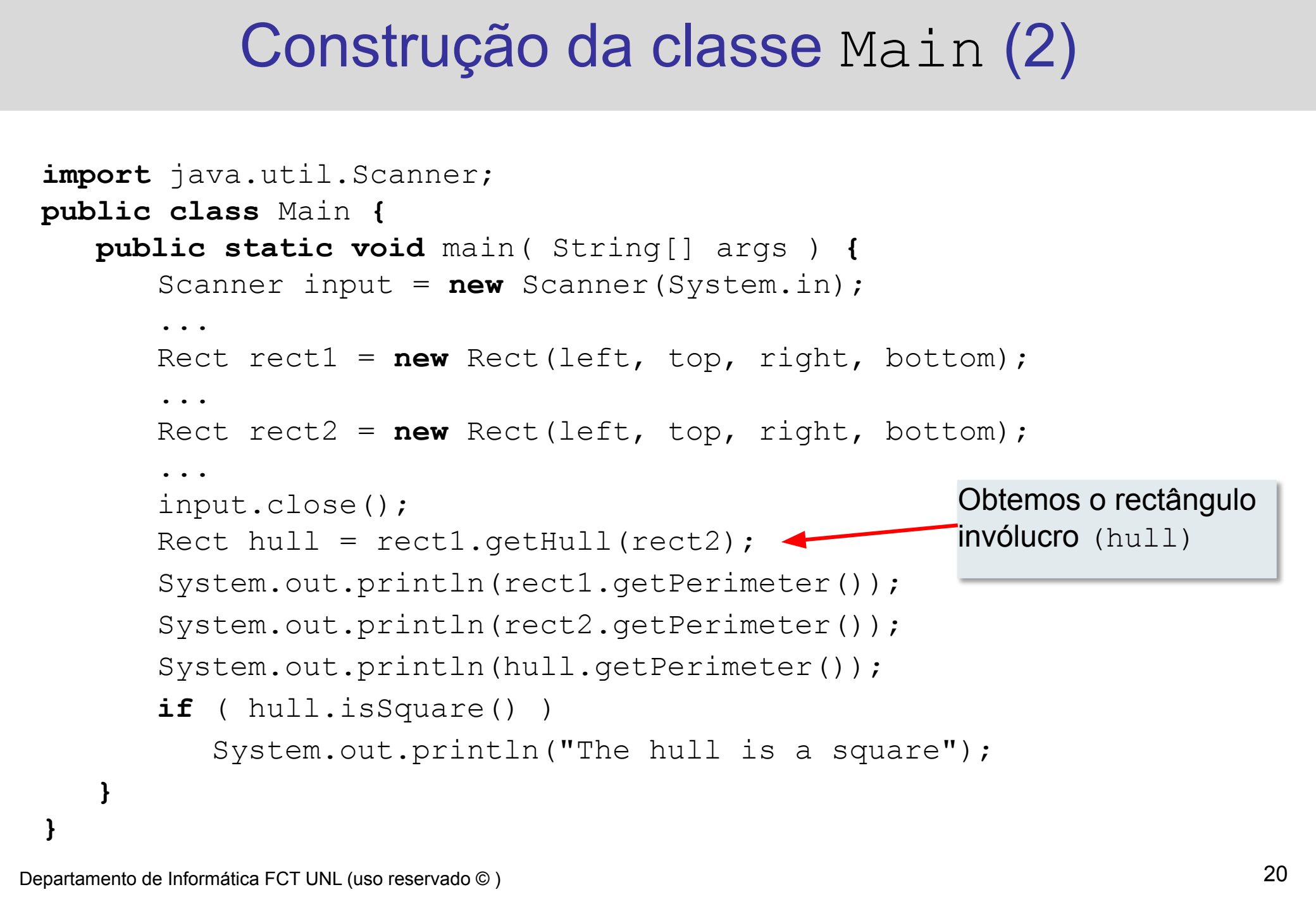
Construção da classe `Main` (1)

```
import java.util.Scanner;
public class Main {
    public static void main( String[] args ) {
        Scanner input = new Scanner(System.in);
        double left = input.nextDouble();
        double top = input.nextDouble();
        double right = input.nextDouble();
        double bottom = input.nextDouble();
        input.nextLine();
        Rect rect1 = new Rect(left, top, right, bottom);
        left = input.nextDouble();
        top = input.nextDouble();
        right = input.nextDouble();
        bottom = input.nextDouble();
        input.nextLine();
        Rect rect2 = new Rect(left, top, right, bottom);
        input.close();
        //Continua no slide seguinte
    }
}
```

Código repetido: temos de aprender a resolver o problema

Construção da classe `Main` (2)

```
import java.util.Scanner;
public class Main {
   public static void main( String[] args ) {
      Scanner input = new Scanner(System.in);
      ...
      Rect rect1 = new Rect(left, top, right, bottom);
      ...
      Rect rect2 = new Rect(left, top, right, bottom);
      ...
      input.close();
      Rect hull = rect1.getHull(rect2);
      System.out.println(rect1.getPerimeter());
      System.out.println(rect2.getPerimeter());
      System.out.println(hull.getPerimeter());
      if ( hull.isSquare() )
         System.out.println("The hull is a square");
   }
}
```



Parte da Interface do Rectângulo

`double` `getPerimeter()`

calcula o perímetro do rectângulo

`boolean` `isSquare()`

devolve true se, e só se, o rectângulo for um quadrado

O método devolve um objecto rectângulo.

O parâmetro do método é um objecto rectângulo.

```
Rect getHull( Rect other )
```

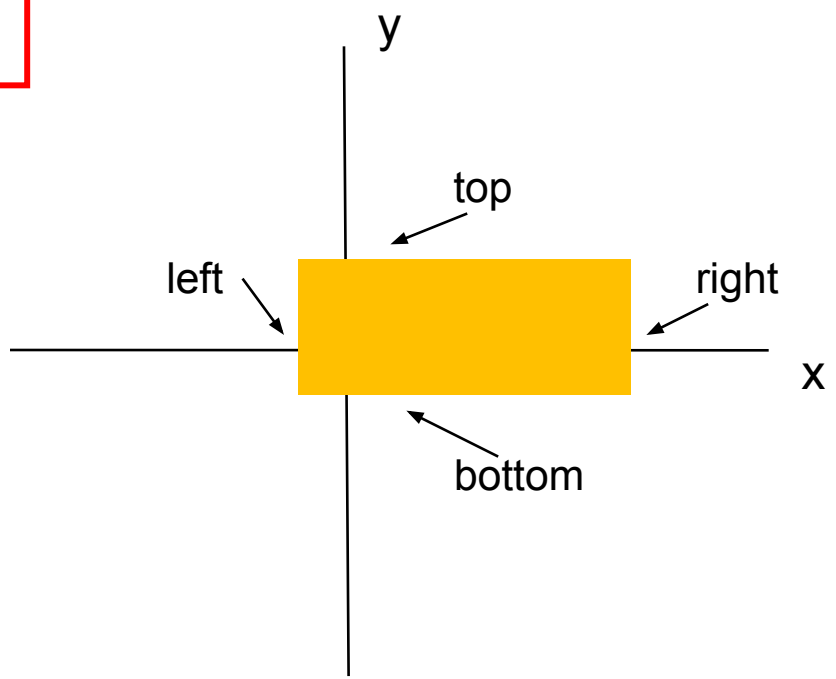
devolve o menor rectângulo que contém o rectângulo e o rectângulo especificado

Programando o Rectângulo

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

    ...
}
```

Variáveis de instância



Programando o Rectângulo

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

Variáveis de instância

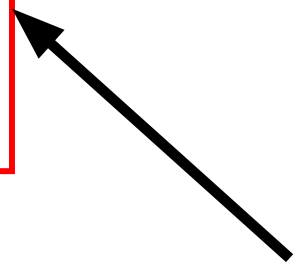
Construtor

Método `getPerimeter`

```
private double left;
private double top;
private double right;
private double bottom;
```

```
public double getPerimeter() {


}
```



Como se calcula o perímetro?
O perímetro é: 2 vezes (largura + altura)

Método `getPerimeter`

```
private double left;
private double top;
private double right;
private double bottom;
```

Retorna o dobro da soma dos valores **retornados** pelas chamadas aos métodos `getWidth()` e `getHeight()`

```
public double getPerimeter() {
    return 2 * (this.getWidth() + this.getHeight());
}
```

`this` denota o próprio objecto

```
public double getWidth() { … }
public double getHeight() { … }
```

Métodos da interface do rectângulo

Método `getHull`

```
private double left;
private double top;
private double right;
private double bottom;
```

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

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

```
public Rect getHull( Rect other ) {
  double xMin = Math.min(left, other.getLeft());
  double xMax = Math.max(right, other.getRight());
  double yMin = Math.min(bottom, other.getBottom());
  double yMax = Math.max(top, other.getTop());
  return new Rect(xMin, yMax, xMax, yMin);
}
```

Método `getHull`

```
public Rect getHull( Rect other ) {
   ...
}
```

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

Quando temos uma variável para guardar a referência de um objecto (no exemplo, `other`), como saber se o objecto ainda não foi criado?

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

- Em Java, há um valor especial → **`null`**.
- Por convenção, **`null`** representa a ausência de objecto.

Exemplos de declarações e inicializações de variáveis

```
Rect r = null;                  // Ainda não existe o objecto
Rect r = new Rect(1,2,2,1);     // O objecto é criado e a sua referência guardada
                                // na variável r
```

Método `getHull`

Pré-condição

- A execução correcta do método é garantida apenas se `other != null`
- A falha na pré-condição só acontece quando não estiver uma referência para um objecto `Rect` em `other`

```
/* Pre: other != null */
public Rect getHull( Rect other ) {
  double xMin = Math.min(left, other.getLeft());
  double xMax = Math.max(right, other.getRight());
  double yMin = Math.min(bottom, other.getBottom());
  double yMax = Math.max(top, other.getTop());
  return new Rect(xMin, yMax, xMax, yMin);
}
```

Método `getHull`

```
/* Pre: other != null */
public Rect getHull( Rect other ) {
  double xMin = Math.min(left, other.getLeft());
  double xMax = Math.max(right, other.getRight());
  double yMin = Math.min(bottom, other.getBottom());
  double yMax = Math.max(top, other.getTop());
  return new Rect(xMin, yMax, xMax, yMin);
}
```

Cria-se um objecto rectângulo, chamando o construtor de `Rect`, e retorna-se a sua referência

```
public double getLeft() { … }
public double getRight() { … }
public double getTop() { … }
public double getBottom() { … }
```

Métodos da interface do rectângulo

Interface do Rectângulo (1)

`double` `getLeft()`

consulta a abcissa dos cantos esquerdos do rectângulo

`double` `getRight()`

consulta a abcissa dos cantos direitos do rectângulo

`double` `getTop()`

consulta a ordenada dos cantos superiores do rectângulo

`double` `getBottom()`

consulta a ordenada dos cantos inferiores do rectângulo

`double` `getWidth()`

consulta a largura do rectângulo

`double` `getHeight()`

consulta a altura do rectângulo

Interface do Rectângulo (2)

`double` `getPerimeter()`

calcula o perímetro do rectângulo

`boolean` `isSquare()`

devolve true se, e só se, o rectângulo for um quadrado

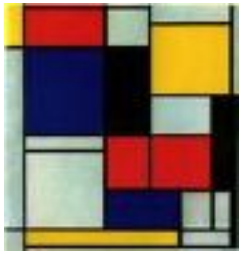
`Rect getHull( Rect other )`

devolve o menor rectângulo que contém o rectângulo e o rectângulo especificado

Pre: other != null

Alguns Programas Simples

- Vimos como programar em Java algumas classes simples.



- Foram introduzidos vários aspectos importantes:
  - Operações com valores reais e lógicos
  - Parâmetros de construtores e parâmetros de métodos
  - 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