Programando em Java

Classes Simples

Coleção de Exercícios Extra 1

Licenciatura em Engenharia Informática FCT/UNL

http://ctp.di.fct.unl.pt/lei/ip/

2022/2023

Objetivos

- O aluno deverá ser capaz de:
  - A partir de uma especificação simples, em língua natural, identificar a classe a definir e propor uma interface para essa classe .
  - Usar o Eclipse na construção de classes simples, definindo:
    - Variáveis de instância (privadas);
    - Construtor de instâncias da classe (público);
    - Operações modificadoras (públicas);
    - Operações de consulta (públicas).
  - Usar operações aritméticas simples na implementação da classe, que envolvam valores inteiros, reais e lógicos.
  - Testar cuidadosamente as classes desenvolvidas, analisando com espírito crítico os resultados produzidos pelos testes que forem feitos na classe Main.
  - Interpretar corretamente as eventuais mensagens de erro resultantes de defeitos no código produzido.

Semáforo

Semáforo

- **Objetivo**
  - Simular um semáforo.
- **Descrição**
  - O estado de um semáforo pode ser: “vermelho”, “verde”, ou “amarelo”.
- **Funcionalidades**
  - O semáforo é sempre informado para mudar de estado, sendo que a mudança ocorre da seguinte forma:

    vermelho → verde → amarelo → vermelho → …

    É sempre possível consultar se o semáforo está vermelho, verde ou amarelo. Também se pode perguntar se é possível passar (estado “verde” ou “amarelo”) e se é obrigatório parar (estado “vermelho”).

    Quando o semáforo é criado, o seu estado é “vermelho”.
- **Interação com o utilizador**
  - Após criar um semáforo, pode invocar as operações do semáforo.

Semáforo

- Interface (classeTrafficLight):

```
boolean isRed()
   Testa se o semáforo está vermelho.
boolean isGreen()
   Testa se o semáforo está verde.
boolean isYellow()
   Testa se o semáforo está amarelo.
boolean canGo()
   Testa se se pode passar.
boolean mustStop()
   Testa se temos de parar.
void changeColor()
   Muda o estado do semáforo.
```

Semáforo

```
TrafficLight t1 = new TrafficLight();
System.out.println(t1.isGreen());
// false   (boolean)
System.out.println(t1.isRed());
// true   (boolean)
System.out.println(t1.canGo());
// false   (boolean)
System.out.println(t1.mustStop());
// true   (boolean)
t1.changeColor();
System.out.println(t1.isYellow());
// false   (boolean)
```

```
System.out.println(t1.canGo());
//  true   (boolean)
t1.changeColor();
System.out.println(t1.isGreen());
//  false   (boolean)
System.out.println(t1.isYellow());
//  true   (boolean)
System.out.println(t1.canGo());
//  true   (boolean)
t1.changeColor();
System.out.println(t1.canGo());
//  false   (boolean)
```

Semáforo

- Defina em Java uma classe TrafficLight cujos objetos representam Semáforos.
- Programe a sua classe no Eclipse.
- Teste um (ou vários) objetos TrafficLight e verifique se se comportam como esperado.

Dieta

Dieta

- **Objetivo**
  - Controlar uma dieta.
- **Descrição**
  - Numa dieta há ingestão de alimentos, com as odiadas calorias, e exercício físico, em que a pessoa em dieta se livra dos excessos cometidos.
- **Funcionalidades**
  - Em cada refeição ou exercício, registam-se sempre as calorias ganhas ou perdidas, respetivamente.
  - É necessário saber sempre as calorias retidas (ingeridas que não são perdidas). Assim como se o valor das calorias é negativo.
  - É sempre possível consultar o número de refeições e exercícios realizados. Também é possível consultar a média de calorias ingeridas e de calorias perdidas nos casos em que essas estatísticas estejam definidas.
  - Quando é criada, as calorias existentes são zero.
- **Interação com o utilizador**
  - Após criar uma dieta, pode invocar as operações da dieta.

Dieta

- Interface (classe `Diet`) - parte 1:

`void` `eat(`**`int`** `c)`
Ingere c calorias numa refeição
pre: `c > 0`

`void` `burn(`**`int`** `c)`
Consome c calorias a realizar um exercício
pre: `c > 0`

`int` `getEatTimes()`
Retorna o número de refeições realizadas

`int` `getBurnTimes()`
Retorna o número de exercícios realizados

Dieta

- Interface (classe `Diet`) - parte 2:

`int` `getBalance()`
Devolve o saldo total de calorias

`boolean` `isBalanceNegative()`
Testa se o saldo total de calorias é negativo

`double` `averageEatenCallories()`
Devolve o valor médio das calorias ingeridas
pre: `getEatTimes() > 0`

`double` `averageBurntCallories()`
Devolve o valor médio das calorias consumidas
pre: `getBurnTimes() > 0`

Dieta

```
Diet d = new Diet();
System.out.println(d.getBalance());
//  0   (int)
System.out.println(d.getEatTimes());
//  0   (int)
d.eat(50);
d.eat(75);
System.out.println(d.getEatTimes());
//  2   (int)
System.out.println(d.getBurnTimes());
//  0   (int)
d.burn(20);
System.out.println(d.getBalance());
//  105   (int)
d.eat(75);
d.eat(100);
d.burn(40);
```

```
d.burn(30);
System.out.println(d.getBalance());
//  210   (int)
System.out.println(d.getEatTimes());
//  4   (int)
System.out.println(d.getBurnTimes());
//  3   (int)
System.out.println(d.averageEatenCalories());
//  75.0   (double)
System.out.println(d.averageBurntCalories());
//  30.0   (double)
System.out.println(d.isBalanceNegative());
//  false   (boolean)
```

Dieta

- Defina em Java uma classe `Diet`.
- Programe a sua classe no Eclipse e forneça também um programa principal para testes.
- Teste no programa principal vários objetos da classe `Diet` e verifique que se comportam tal como esperado.

Speed

Speed

- **Objetivo**
  - Manipular valores de velocidades.
- **Descrição**
  - A velocidade é um valor real que pode ser expresso em diferentes unidades: metros por segundo (m/s), quilómetros por hora (km/h) e milhas por hora (mph) .
- **Funcionalidades**
  - É sempre possível registar e consultar o valor de velocidade em qualquer unidade. As conversões a considerar são:

	m/s	km/h	mph
1 m/s =	1	3.6	2.236936
1 km/h =	0.277778	1	0.621371
1 mph =	0.44704	1.609344	1

  - Quando é criado, o valor da velocidade é de 0 m/s.
- **Interação com o utilizador**
  - Registar valores de velocidades e fazer conversões entre unidades.

Speed

- Interface (classe `Speed`)

`double` `getInMS()`
Consultar a velocidade em metros por segundo.

`void` `setInMS(`**`double`** `speed)`
Definir a velocidade em metros por segundo
pre: `speed > 0`

`double` `getInKmH()`
Consultar a velocidade em kilómetros por hora

`void` `setInKmH(`**`double`** `speed)`
Definir a velocidade em kilómetros por hora
pre: `speed > 0`

`double` `getInMpH()`
Consultar a velocidade em milhas por hora

`void` `setInMpH(`**`double`** `speed)`
Definir a velocidade em milhas por hora
pre: `speed > 0`

Speed

```
Speed snail = new Speed();
snail.setInMS(0.001);
System.out.println(snail.getInMS());
// 0.001    (double)
System.out.println(snail.getInKmH());
// 0.003600    (double)
System.out.println(snail.getInMpH());
// 0.002237    (double)
Speed obikwelu = new Speed();
obikwelu.setInKmH(36.0);
System.out.println(obikwelu.getInMS());
// 10.0    (double)
System.out.println(obikwelu.getInKmH());
// 36.0    (double)
System.out.println(obikwelu.getInMpH());
// 22.369360    (double)
obikwelu.setInMpH(24.0);
System.out.println(obikwelu.getInKmH());
// 38.624261    (double)
```

Speed

- Defina em Java uma classe `Speed`, cujos objetos representam o conceito de velocidade.
- Programe a sua classe no Eclipse.
- Teste vários objetos da classe `Speed` e verifique que se comportam tal como esperado.

Decatlo

Decatlo

- **Objetivo**
  - Manipular os resultados de um atleta de Decatlo.
- **Descrição**
  - O decatlo consiste em 10 provas realizadas em dois dias: (1) 100 metros, salto em comprimento, tiro, salto em altura e 400 metros; (2) 110 metros barreiras, lançamento do disco, salto com vara, lançamento do dardo e 1500 metros.
  - Cada prova tem uma pontuação (valor real). Há 3 tipos de provas, cuja pontuação (P) se calcula do seguinte modo:
    - Corridas: $\mathbf{P = a * (b - T)^c}$
      [onde **T** é o Tempo em segundos; e.g. 10.43 para 100 metros]
    - Saltos: $\mathbf{P = a * (M - b)^c}$
      [onde **M** é a medida em centímetros; e.g. 808 para o salto em comprimento]
    - Lançamentos: $\mathbf{P = a * (D - b)^c}$
      [onde **D** é a distância em metros; e.g. 16.69 para a prova de tiro]

Decatlo

- As constantes a, b e c, para cada prova, definem-se de acordo com a seguinte tabela.

Unidade	Prova	a	b	c
Segundos (double)	100m	25.4347	18.00	1.81
Segundos (double)	400m	1.53775	82.00	1.81
Segundos (double)	1500m	0.03768	480.00	1.85
Segundos (double)	110m Barreiras	5.74352	28.50	1.92
Centímetros (int)	High Jump	0.8465	75.00	1.42
Centímetros (int)	Pole Vault	0.2797	100.00	1.35
Centímetros (int)	Long Jump	0.14354	220.00	1.40
Metros (double)	Shot	51.39	1.50	1.05
Metros (double)	Discus	12.91	4.00	1.10
Metros (double)	Javelin	10.14	7.00	1.08

Decatlo

A pontuação global do atleta é a parte inteira (arredondamento para baixo) do valor resultante da soma de todas as pontuações obtidas nas diferentes provas do decatlo.

- **Funcionalidades**
  - Registar o desempenho (tempo em segundos, medição em centímetros ou distância em metros) do atleta nas diferentes provas do decatlo (corridas, saltos ou lançamentos).
  - É sempre possível consultar a pontuação do atleta numa dada prova do decatlo, assim como a pontuação global (soma da pontuação de todas as provas).
- **Interação com o utilizador**
  - Registar e calcular pontuações das provas realizadas por um atleta no decatlo.

Decatlo

- Interface (classe `Decathlon`) - parte 1:

```
// set operations with the result of each event
// pre: t > 0, m > 0, d > 0 (on the corresponding methods)
void set100Meters(double t)
void set400Meters(double t)
void set1500Meters(double t)
void set110MetersHurdles(double t)
void setHighJump(int m)
void setLongJump(int m)
void setPoleVaultJump(int m)
void setShot(double d)
void setDiscus(double d)
void setJavelin(double d)
void reset() // set everything to zero
```

Decatlo

- Interface (classe `Decathlon`) - parte 2:

```
// get operations with the result of each event
double get100Meters()
double get400Meters()
double get1500Meters()
double get110MetersHurdles()
int getHighJump()
int getLongJump()
int getPoleVaultJump()
double getShot()
double getDiscus()
double getJavelin()

// total score, as sum of score of each proof, rounded down
int getPoints()
```

Decatlo

```
Decathlon d = new Decathlon();
d.set100Meters(10.4);
System.out.println(d.getPoints());
// 999    (int)
d.reset();
System.out.println(d.getPoints());
// 0    (int)
d.set100Meters(10.4);
d.setLongJump(736);
System.out.println(d.getPoints());
// 1900    (int)
d.setShot(13.53);
System.out.println(d.getPoints());
// 2600    (int)
d.setHighJump(210);
System.out.println(d.getPoints());
// 3497    (int)
```

Decatlo

- Defina em Java uma classe Decathlon.
- Programe a sua classe no Eclipse.
- Teste um (ou vários) objetos Decathlon e verifique se se comportam como esperado.



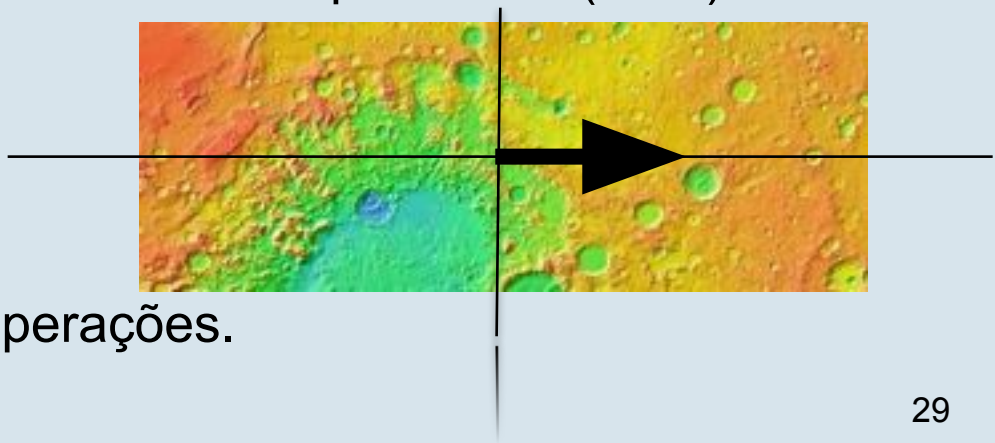
Mars Rover

Mars Rover

- **Objetivo**
  - Simular um veículo de exploração do planeta Marte.
- **Descrição**
  - Um veículo de exploração do planeta Marte desloca-se num plano imaginário sobreposto à superfície do planeta, em que cada posição é indicada por um par de coordenadas (números reais).
  - Um veículo encontra-se sempre numa dada posição (x,y) e com uma dada orientação (direção absoluta).

Mars Rover

- **Funcionalidades**
  - O veículo desloca-se em linha reta **d** metros a partir da posição corrente e na direção em que se encontra. O veículo pode também mudar de direção.
  - É sempre possível consultar as coordenadas da posição corrente do veículo, assim como a sua direção corrente.
  - Estes veículos conseguem medir distâncias (em linha reta) entre vários pontos no terreno. Mais precisamente, podem registar um dado ponto e depois calcular a distância desse último ponto registado até à posição corrente do veículo.
  - Quando é criado, o veículo encontra-se na origem do plano (X=0,Y=0), tem registado esse mesmo ponto (X=0,Y=0), e está virado para leste (α=0º).



- **Interação com o utilizador**
  - Após criar um veículo, pode invocar as operações.

Mars Rover

- **Interface** (classe MarsRover)

void moveForward**(double** distance**)**
O Rover avança `distance` unidades na direção corrente.
`pre: distance > 0.0`

void setHeading**(double** angle**)**
O Rover vira-se para a direção absoluta `angle`.
`pre: 0.0 <= angle && angle < 360.0` (unidades graus)

double getXPos**()**
Consultar o valor da coordenada `X`.

double getYPos**()**
Consultar o valor da coordenada `Y`.

double getHeading**()**
Consultar o valor da orientação do Rover.

void mark**()**
O Rover regista o ponto corrente como ponto base, para próximo cálculo de distância.

double getDistance**()**
O Rover calcula a distância (em linha reta) entre o último ponto base marcado e a sua posição corrente.

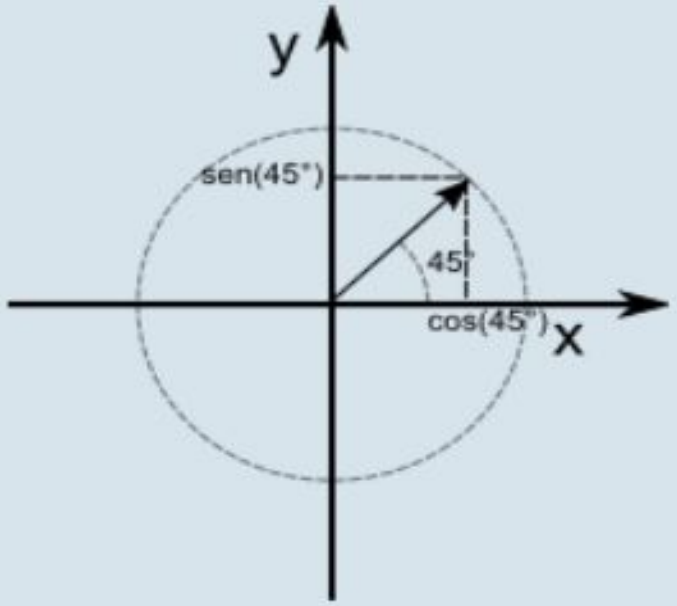
Mars Rover

- **Dicas**

  Nova posição em X = posição anterior em X + (distância) * cos(angle);
  Nova posição em Y = posição anterior em Y + (distância) * sin(angle);

Funções da biblioteca Math necessárias:

- Math.sin(radianos) - seno
- Math.cos(radianos) - cosseno
- Math.toRadians(graus) – converte graus em radianos



Mars Rover

```
MarsRover r = new MarsRover();
System.out.println(r.getXPos());
// 0.0   (double)
System.out.println(r.getYPos());
// 0.0   (double)
r.setHeading(90);
r.moveForward(1.0);
System.out.println(r.getXPos());
// 6.123234e-17 (double)
System.out.println(r.getYPos());
// 1.0   (double)
r.setHeading(270);
r.moveForward(2);
System.out.println(r.getYPos());
// -1.0   (double)
System.out.println(r.getXPos());
// -6.123234e-17 (double)
r.mark();
r.setHeading(90);
r.moveForward(2);
System.out.println(r.getDistance());
// 2.0   (double)
r.setHeading(0);
r.moveForward(2);
System.out.println(r.getDistance());
// 2.828427   (double)
```

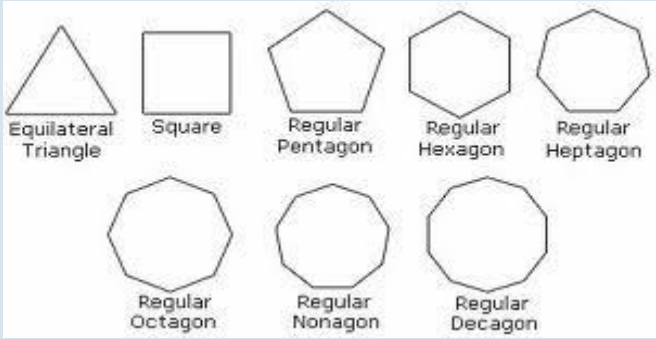
Mars Rover

- Defina em Java uma classe MarsRover cujos objetos têm a funcionalidade indicada.
- Programe a sua classe no Eclipse.
- Teste um (ou vários) objetos MarsRover e verifique se se comportam como esperado.

Polígonos regulares

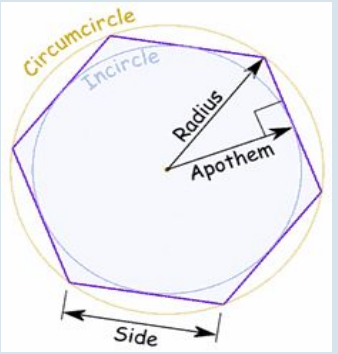
Polígonos regulares

- Defina em Java uma classe `RegularPolygon`, cujos objetos representam polígonos regulares. A sua classe deverá permitir a construção de um polígono regular com qualquer número de lados maior ou igual a 3, bem como o cálculo de diversas estatísticas, como o perímetro e a área.
- Programe a sua classe no Eclipse.
- Teste vários objetos da classe `RegularPolygon` e verifique que se comportam da forma esperada.

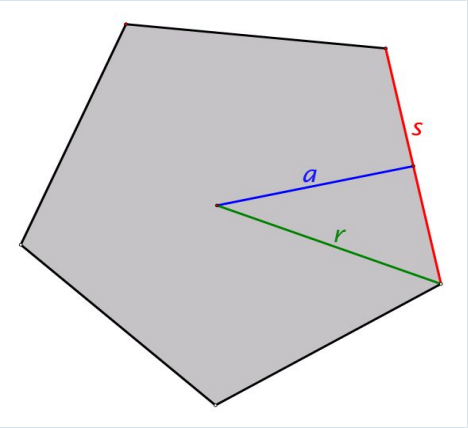


Polígonos regulares

- Normalmente, define-se um polígono regular usando os seguintes dados:
  - n = número de lados (n >= 3)
  - r = circum-raio, o raio da circunferência envolvente (r > 0)



- Eis outras propriedades notáveis dum polígono regular, que podem ser calculadas:
  - a = apótema [a = r * cos(π/n)]
  - s = comp. do lado [s = 2 * r * sin(π/n)]
  - P = perímetro [P = n * s]
  - A = área [A = ½ * P * a]



Observações:

- Todos os vértices situam-se sobre uma circunferência de raio r.
- Todos os lados e ângulos são iguais.
- Apótema (ou in-raio) é o raio do círculo inscrito.

Polígonos regulares

- Interface (classe RegularPolygon)

double `getNumberOfSides()`
Retorna n, o número de lados.

double `getRadius()`
Retorna r, o circum-raio.

double `getSide()`
Retorna s, o tamanho de qualquer lado.

double `getApothem()`
Retorna a, o apótema.

double `getPerimeter()`
Retorna P, o perímetro.

double `getArea()`
Retorna A, a área.

Polígonos regulares

```
RegularPolygon t = new RegularPolygon(3, 1.0);
System.out.println(t.getPerimeter());
// 5.196152 (double)
System.out.println(t.getArea());
// 1.299038 (double)
RegularPolygon q = new RegularPolygon(4, 1.0);
System.out.println(t.getPerimeter());
// 6.928203 (double)
System.out.println(t.getArea());
// 2.0   (double)
```

Conversor

Conversor

- Defina em Java uma classe CurrencyConverter cujos objetos guardam um valor monetário numa determinada moeda. Existem métodos para efetuar conversões em Euros, Dólares e Libras.
- Programe a sua classe no Eclipse.
- Teste um objeto CurrencyConverter e verifique se ele se comporta como esperado.

Conversor

- Um conversor de câmbio permite transformar uma importância expressa numa moeda noutra moeda.
  - Neste problema vamos considerar apenas conversões de e para Euros (EUR), Libras (GPB) e Dólares (USD)
  - Considere as taxas de conversão (de 20/10/2022):
    - 1 USD = 1.01 EUR
    - 1 EUR = 0.87 GBP

Conversor

- Interface (classe CurrencyConverter):

```
double getInEuros()
```
Consultar a importância em euros

```
void setInEuros(double amount)
```
Definir a importância em Euros
pre: amount >= 0

```
double getInDollars()
```
Consultar a importância em dólares

```
void setInDollars(double amount)
```
Definir a importância em dólares
pre: amount >= 0

```
double getInPounds()
```
Consultar a importância em libras

```
void setInPounds(double amount)
```
Definir a importância em libras
pre: amount >= 0

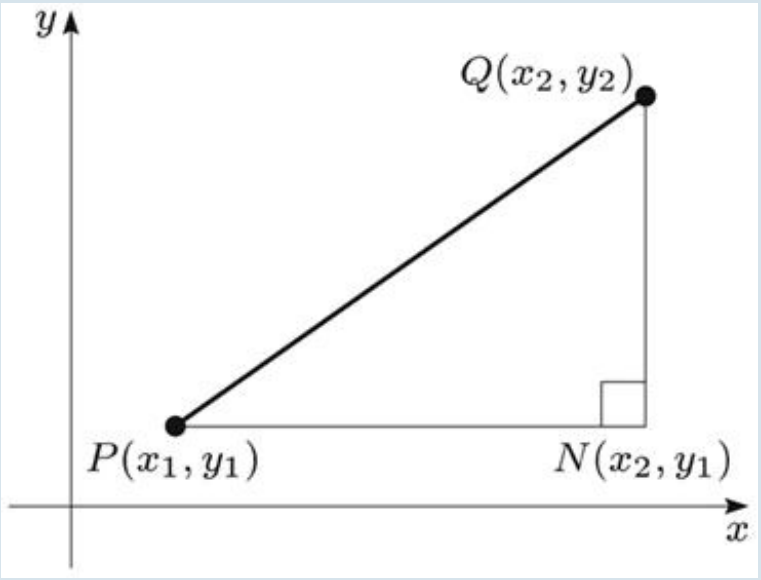
Conversor

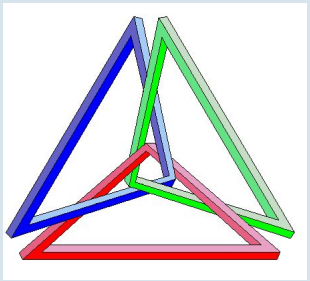
```
CurrencyConverter c = new CurrencyConverter();
c.setInEuros(100.0);
System.out.println(c.getInEuros());
//  100.0000  (double)
System.out.println(c.getInDollars());
//  99.0099  (double)
System.out.println(c.getInPounds());
//  87.0000  (double)
c.setInPounds(200.0);
System.out.println(c.getInEuros());
//  229.8851  (double)
System.out.println(c.getInDollars());
//  227.6090  (double)
System.out.println(c.getInPounds());
//  200.0000  (double)
```

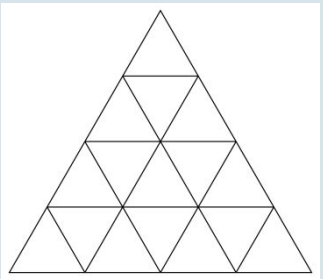
Triângulo

Triângulo

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







Triângulo

- Cada objeto Triangle
  - Representa um triângulo no plano, definido pelos seus três vértices
- Pretende-se construir objetos Triangle de **duas** maneiras:
  - Indicando:
    - A abcissa do primeiro vértice Px
    - A ordenada do primeiro vértice Py
    - A abcissa do segundo vértice Qx
    - A ordenada do segundo vértice Qy
    - A abcissa do terceiro vértice Rx
    - A ordenada do terceiro vértice Ry
  - Ou não indicando nada:
    o triângulo é criado com os seguintes vértices por omissão: (1, 1), (4, 4), (5, 1) (conveniente para testes)

Triângulo

- Interface (classe Triangle) - parte 1:

`double` `getPx()`
devolve a abcissa do primeiro vértice

`double` `getPy()`
devolve a ordenada do primeiro vértice

`double` `getQx()`
devolve a abcissa do segundo vértice

`double` `getQy()`
devolve a ordenada do segundo vértice

`double` `getRx()`
devolve a abcissa do terceiro vértice

`double` `getRy()`
devolve a ordenada do terceiro vértice

Triângulo

- Interface (classe Triangle) - parte 2:

double `getSideLength1`**()**
  devolve o comprimento do 1º lado (P-Q)

double `getSideLength2`**()**
  devolve o comprimento do 2º lado (Q-R)

double `getSideLength3`**()**
  devolve o comprimento do 3º lado (R-P)

double `getPerimeter`**()**
  devolve o perímetro

double `getArea`**()**
  devolve a área, segundo a fórmula de Heron

Triângulo

- Métodos privados
  - Poderá interessar definir operações privadas, para evitar repetições dos mesmos cálculos em pontos distintos.
  - Nas situações em que detete a vantagem de existir alguma dessas operações privadas, implemente-a e utilize-a na implementação de outras operações de modo a evitar repetição de código.

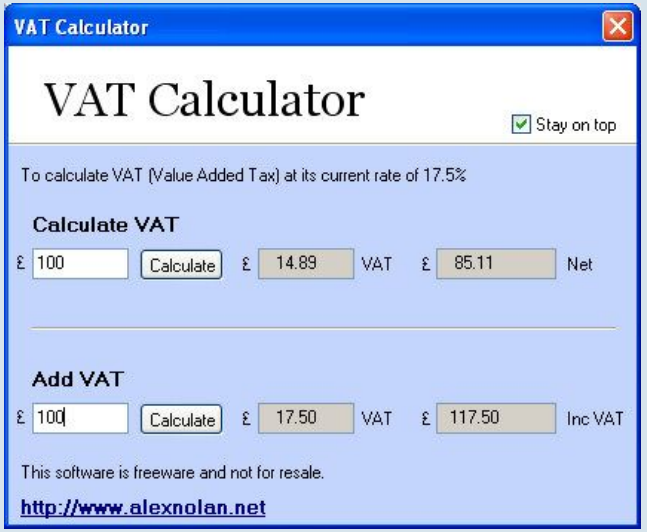
Triângulo

```
Triangle t = new Triangle();
System.out.println(t.getPx());
// 1.0000   (double)
System.out.println(t.getQy());
// 4.0000   (double)
System.out.println(t.getSideLength1());
// 4.2426   (double)
System.out.println(t.getSideLength2());
// 3.1623   (double)
System.out.println(t.getPerimeter());
// 11.4050  (double)
System.out.println(t.getArea());
// 5.9999   (double)
```

Cálculo do IVA

Cálculo do IVA

- Defina em Java uma classe `VATCalculator` cujos objetos permitem o cálculo do IVA para as taxas existentes (em tempos) em Portugal.
- Programe a sua classe no Eclipse.
- Teste um (ou vários) objetos VATCalculator e verifique se se comportam como esperado.



Cálculo do IVA

- Cada objeto ou serviço transacionado em Portugal implica a cobrança de Imposto sobre o Valor Acrescentado (IVA).
- Um empresário que fornece serviços deve cobrar IVA aos seus clientes, que depois entrega ao estado.
- Se, no processo dos seus negócios, adquirir bens ou serviços, pode descontar o IVA gasto naquele que tem de entregar ao estado.
- É por isso importante saber o valor de IVA gasto ao longo de um determinado período.
- A cada valor líquido transacionado aplica-se uma taxa de IVA, dependendo do tipo do produto ou serviço em causa.
  - O IVA é calculado multiplicando o valor líquido pela taxa.
- O valor bruto da transação calcula-se somando o valor líquido ao valor do IVA.

Cálculo do IVA

- Cada objeto VATCalculator:
  - Representa uma calculadora de IVA segundo três taxas: mínima, média e máxima
- Um objeto VATCalculator é criado:
  - Indicando a taxa mínima, média e máxima de IVA a aplicar a cada categoria de transação
  - Ou sem indicar nada, o que implica a utilização das taxas conhecidas (6%, 13% e 23%)

Cálculo do IVA

- Interface (classe VATCalculator) - Modificadores:

`void` `addValueMin(`**`double`** `netAmount)`
Adição de valor gasto líquido (sem IVA) dentro da taxa mínima.
pre: `netAmount > 0`

`void` `addValueMed(`**`double`** `netAmount)`
Adição de valor gasto líquido (sem IVA) dentro da taxa média.
pre: `netAmount > 0`

`void` `addValueMax(`**`double`** `netAmount)`
Adição de valor gasto líquido (sem IVA) dentro da taxa máxima.
pre: `netAmount > 0`

Cálculo do IVA

- Interface (classe VATCalculator) - Seletores:

double `getVATTotal()`

Método que devolve o valor total de IVA calculado.

double `getNetTotal()`

Método que devolve o valor total líquido transacionado.

double `getGrossTotal()`

Método que devolve o valor total bruto transacionado.

double `getVATMin()`

Método que devolve o valor de IVA dentro da taxa mínima.

double `getVATMed()`

Método que devolve o valor de IVA dentro da taxa média.

double `getVATMax()`

Método que devolve o valor de IVA dentro da taxa máxima.

Cálculo do IVA

```
VATCalculator v=
    new VATCalculator(0.05,0.12,0.20);
v.addValueMin(253);
v.addValueMed(3100);
v.addValueMax(500);
System.out.println(v.getVATMin());
//  12.65    (double)
System.out.println(v.getVATMed());
//  372.00   (double)
System.out.println(v.getVATMax());
//  100.00   (double)
System.out.println(v.getVATTotal());
//  484.65   (double)
System.out.println(v.getNetTotal());
//  3853.00  (double)
```

```
System.out.println(v.getGrossTotal());
//  4337.65   (double)
v.addValueMed(405);
System.out.println(v.getVATTotal());
//  533.25    (double)
System.out.println(v.getNetTotal());
//  4258.00   (double)
System.out.println(v.getGrossTotal());
//  4791.25   (double)
```