

## FICHA PRÁTICA 3

### LABORATÓRIO DE CLASSES I

#### SÍNTESE TEÓRICA

Nas linguagens de PPO os objectos são divididos em duas grandes categorias de entidades: as INSTÂNCIAS (objectos computacionais puros) e as CLASSES (objectos fundamentalmente de definição, mas que também podem ser de definição e até de prestação de serviços).

CLASSES são objectos particulares que, entre outras funções, guardam a descrição da **estrutura** (*variáveis de instância*) e do **comportamento** (*métodos*) que são comuns a todas as instâncias a partir de si criadas.

As instâncias de uma classe são criadas usando a palavra reservada **new** e um método especial para criação de instâncias, designado **construtor**, que tem, obrigatoriamente, o mesmo nome da classe, podendo ter ou não parâmetros. Exemplos:

```
Triangulo tri1 = new Triangulo();  
Ponto p = new Ponto(); Ponto p2 = new Ponto(5, -1);  
Turma t = new Turma(); Aluno al1 = new Aluno();
```

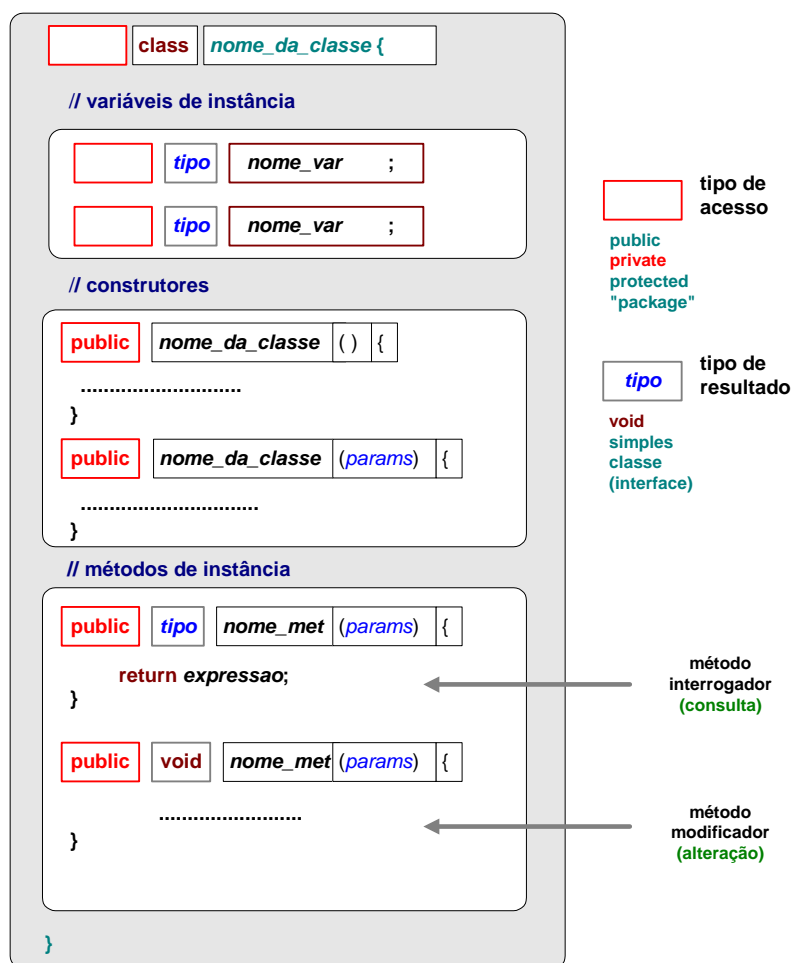


Fig.1 – Estrutura de definição de instâncias típica de uma classe.

## 1.- CLASSE SIMPLES COM MÉTODOS COMPLEMENTARES USUAIS

```
/**
 * Pontos descritos como duas coordenadas reais.
 */
import static java.lang.Math.abs;
public class Ponto2D {

    // Construtores usuais
    public Ponto2D(double cx, double cy) { x = cx; y = cy; }
    public Ponto2D(){ this(0.0, 0.0); } // usa o outro construtor
    public Ponto2D(Ponto2D p) { x = p.getX(); y = p.getY(); }

    // Variáveis de Instância
    private double x, y;

    // Métodos de Instância
    public double getX() { return x; }
    public double getY() { return y; }

    /** incremento das coordenadas */
    public void incCoord(double dx, double dy) {
        x += dx; y += dy;
    }
    /** decremento das coordenadas */
    public void decCoord(double dx, double dy) {
        x -= dx; y -= dy;
    }
    /** soma as coordenadas do ponto parâmetro ao ponto receptor */
    public void somaPonto(Ponto2D p) {
        x += p.getX(); y += p.getY();
    }
    /** soma os valores parâmetro e devolve um novo ponto */
    public Ponto2D somaPonto(double dx, double dy) {
        return new Ponto2D(x += dx, y += dy);
    }
    /** determina se um ponto é simétrico (dista do eixo dos XX o
        mesmo que do eixo dos YY) */
    public boolean simetrico() {
        return abs(x) == abs(y);
    }

    /** verifica se ambas as coordenadas são positivas */
    public boolean coordPos() {
        return x > 0 && y > 0;
    }

    // Métodos complementares usuais

    /** verifica se os 2 pontos são iguais */
    public boolean igual(Ponto2D p) {
        if (p != null) return (x == p.getX() && y == p.getY());
        else return false;
    }

    // outra versão de igual(Ponto2D p)
    public boolean igual1(Ponto2D p) {
        return (p == null) ? false : x == p.getX() && y == p.getY();
    }
}
```

```

/** Converte para uma representação textual */
public String toString() {
    return new String("Pt2D = " + x + ", " + y);
}

/** Cria uma cópia do ponto receptor (receptor = this) */
public Ponto2D clone() {
    return new Ponto2D(this);
}
}

```

## 2.- REGRAS FUNDAMENTAIS PARA A CRIAÇÃO DE CLASSES.

- As variáveis de instância devem ser declaradas como **private**, satisfazendo assim o princípio do encapsulamento;
- Pelo menos dois construtores devem ser criados: o que redefine o construtor de JAVA por omissão e o construtor para cópia (recebe um objecto da mesma classe e inicializa as variáveis de instância com os valores deste);
- Por cada variável de instância devemos ter um método de consulta **getVar()** ou equivalente, e um método de modificação **setVar()** ou equivalente;
- Devem ser sempre incluídos nas definições das classes, a menos que não se justifique por serem demasiado complexas, os métodos complementares: **equals()**, **toString()** e **clone()**;
- A classe deverá ser bem documentada, em especial os métodos, usando comentários, em particular os comentários **/\*\* ... \*/** para que seja produzida automaticamente documentação de projecto através do utilitário **javadoc**.

## 3.- DESENVOLVIMENTO EM BLUEJ.

O ambiente BlueJ fornece um ambiente integrado para a criação, desenvolvimento e teste das classes a conceber. As figuras seguintes mostram, de forma simples, as facilidades de edição, de criação de instâncias e teste destas no ambiente BlueJ.

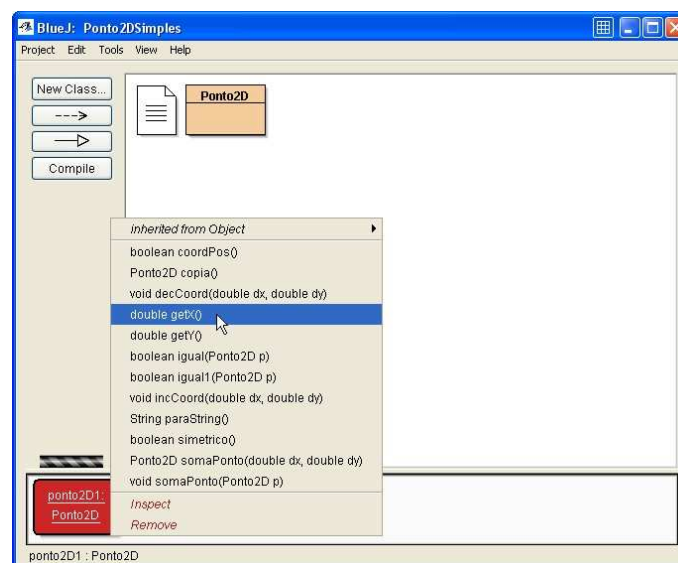


Fig.2 – API e Invocação de Método

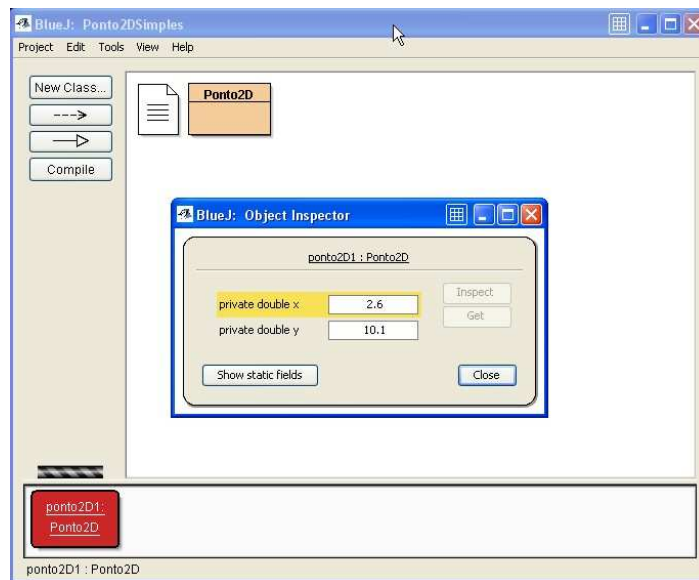


Fig.3 - Resultado de *Inspect*

## EXERCÍCIOS:

**Ex 1:** Um pixel é um ponto de coordenadas x e y inteiras mas que tem a si associada uma cor de 0 a 255. Crie uma classe **Pixel** que permita criar “pixels”, sabendo-se que, para além das usuais operações de consulta, cada pixel deve responder às seguintes mensagens:

- Deslocar-se para cima, para baixo, para a esquerda e para a direita de um valor real;
- Mudar a sua cor para uma nova cor de número dado (entre 0 e 255);
- Sendo o espectro de cores desde o 0 - Preto a 255 – Branco, sendo o preto de 0 a 2 e o branco de 253 a 255, entre o preto e o branco situam-se o cinza, o azul, o verde, o vermelho e o amarelo, sendo o espaço das 250 cores restantes dividido igualmente entre estas 5 (50 para cada). Exemplo: de 3 a 52 são os cinza e de 53 a 102 os azuis. Escrever um método que devolva uma String correspondente à cor actual do pixel.
- Não esquecer os métodos **equals()**, **toString()** e **clone()**.

**Ex 2:** Um Segmento de recta é representável por dois pontos de coordenadas reais: o início e o fim do segmento. Escreva uma classe **Segmento** que implemente os métodos seguintes:

- Calcular o comprimento do segmento;
- Determinar o declive do segmento, cf.  $(y_2 - y_1) / (x_2 - x_1)$ ;
- Determinar se o segmento sobe ou desce a partir do seu início (devolver uma String);
- Deslocar o segmento dx em XX e dy no eixo dos YY;
- Se o segmento for o diâmetro de uma circunferência, determinar qual o perímetro desta;

**Ex 3:** Um veículo motorizado é caracterizado pela sua matrícula, quilometragem total (Km), pelo número de litros total do seu depósito (capacidade), pelo número de litros contidos em tal depósito (reserva incluída = 10 litros). Sabe-se ainda o seu consumo médio aos 100 Km, que raramente varia. O veículo possui ainda um contador de viagens realizadas.

Crie uma classe **Veiculo** que implemente métodos de instância que permitam obter os seguintes resultados:

- Determinar quantos quilómetros é possível percorrer com o combustível que está no depósito;

- Registrar uma viagem de K quilómetros e actualizar os dados do veículo;
- Determinar se o veículo já entrou na reserva;
- Dado um valor médio de custo por litro, calcular o valor total gasto em combustível;
- Dado um valor médio de custo por litro, calcular o custo médio por Km;
- Meter L litros de gasolina, ou o máximo possível  $< L$ , sem transbordar.

**Ex 4:** Um Cartão de Cliente (actualmente tão em voga) é um cartão de compras que acumula pontos de bonificação à medida que são registadas compras, e que possui o valor total em dinheiro das compras realizadas, um código alfanumérico e um nome de titular. Num dado estabelecimento as regras são as seguintes: por cada compra efectuada em Euros o cliente recebe de bônus um número de pontos (inteiro) que é o arredondamento para baixo de 10% do valor da compra. Sempre que é atingido um valor múltiplo de 50 pontos o cliente acumula mais 5 pontos por cada 50, que são juntos aos que já tinha no cartão. Escrever uma classe **CartaoCliente** cujas instâncias exibam este comportamento, e permitam ainda:

- Descontar P pontos ao cartão devido a levantamento de um prémio;
- Modificar o titular do cartão;
- Modificar a taxa de prémio, passando, por exemplo de 10% para 11%;
- Descarregar os pontos de um cartão para o nosso cartão;
- Inserir no cartão a informação de uma nova compra de certo valor, e actualizar dados;

**Ex 5:** Um Rectângulo de base paralela ao eixo dos XX é representável por dois pontos de coordenadas reais, que são os extremos da sua diagonal. Desenvolva uma classe **Rectangulo** com métodos que realizem as operações seguintes:

- Calculem os comprimentos da base, da altura e da diagonal;
- Calcule o perímetro do rectângulo;
- Calcule a área do rectângulo;
- Realize o deslocamento do rectângulo em XX e em YY;

**Ex 6:** Um Produto de um dado stock de produtos possui as seguintes características de informação: código, nome, quantidade em stock, quantidade mínima, preço de compra e preço de venda a público. Desenvolva uma classe **Produto** e os seguintes métodos de instância:

- Alterar a quantidade de um produto, ou por saída ou por entrada de uma dada quantidade do produto no stock;
- Modificar o nome do produto;
- Modificar o preço de venda de um dado produto;
- Determinar o valor total da quantidade em stock em caso de venda;
- Determinar o lucro actual de tal stock em caso de venda total;
- Dada uma encomenda de X unidades do produto, determinar o preço total de tal encomenda;
- Verificar se um produto está já abaixo do nível mínimo de stock.

**Ex 7:** Um número Complexo na sua forma rectangular representa-se por  $z = a + bi$ , e possui uma parte real e uma parte imaginária, sendo a e b números reais que são os coeficientes real e imaginário do número. Outra forma de representar números complexos designa-se por representação polar e tem a forma  $z = r(\cos \Phi + \text{sen } \Phi i)$ , sendo  $r = \sqrt{a^2 + b^2}$ ,  $r \cos \Phi = a$ ,  $r \text{sen } \Phi = b$ , e sendo o valor de  $\Phi = \arctan b/a$ . Desenvolva uma classe **Complexo** com os seguintes métodos de instância:

- Cálculo da soma de dois complexos  $z + w = (a + bi) + (c + di) = (a+c) + (b+d)i$  dando um novo número complexo como resultado;
- Cálculo do produto de dois complexos  $z * w = (a + bi) * (c + di) = (ac-bd) + (bc+ad)i$  dando um novo número complexo como resultado;
- Cálculo do conjugado de um complexo, sendo  $conjugado(a+ bi) = a - bi$  ;
- Calcular a representação polar do complexo e devolvê-la como String;
- Métodos igual(Complexo c) e toString().

**Ex 8:** Um cronómetro marca as diferenças entre dois tempos registados (início e fim). Um cronómetro “double-split” faz ainda mais: Inicia uma contagem de tempo, faz uma primeira paragem, mas continua a contar o tempo até ser feita a segunda paragem. Criar uma classe **CronometroDS** que permita calcular:

- O tempo total em minutos, segundos e milissegundos entre o início e a primeira paragem;
- O tempo total em minutos, segundos e milissegundos entre o início e a segunda paragem;
- A diferença de tempos entre a primeira e a segunda paragem de tempo;
- Determinar o tempo absoluto em *hora-min-seg-miliseg* do arranque e de cada paragem;

**Ex 9:** Uma Conta Bancária a prazo é criada com um código, um titular, tem uma data de início (dia, mês e ano) de contagem de juros que é actualizada sempre que os juros são calculados e adicionados ao capital, tem um dado montante de capital depositado, com um prazo para cálculo de juros, e uma taxa de juro de t% para tal prazo, definida aquando da sua criação. Crie uma classe **ContaPrazo** que, para além dos construtores e dos métodos de consulta, permita realizar as seguintes operações:

- Calcular o número de dias passados desde a abertura da conta;
- Alterar o titular da conta ou alterar a taxa de juros;
- Atingido o prazo para juros, calcular tais juros, juntá-los ao capital, e registar a nova data de cálculo de juros;
- Verificar se hoje é o dia de calcular os juros;
- Fechar a conta calculando o valor total a pagar ao titular (capital inicial + juros);