AULA 13

Objectivos

Esta aula faz uma breve introdução às noções de classe e objecto. No final da aula, os alunos devem conseguir fazer a distinção entre as noções de classe e objecto. Pretende-se que os alunos sejam capazes de definir uma classe (atributos, construtores e métodos) a partir de uma especificação apresentada pelos professores.

1. Copiar e compilar as duas classes apresentadas abaixo. As classes devem ser armazenadas em ficheiros .java separados. O nome dos ficheiros deve ser igual ao nome das classes.

   A classe Produto define um conjunto de atributos que caracterizam os produtos comercializados por uma empresa e dois métodos para processar objectos (instâncias) dessa classe: um construtor para inicializar os atributos quando se cria um novo objecto (instância) da classe (i.e, um novo produto); e um método para actualizar a quantidade em stock quando são recebidas novas unidades de um produto.

   A classe `Aula13_gestProdutos_vars` implementa um pequeno programa exemplo que define, inicializa e mostra dados de dois objectos da classe Produto.

```
import p1.*;
public class Produto extends P1App {

  // atributos da classe produto
  int    codigo;
  String designacao;
  double precoUnit;
  int    quantStock;

  // construtores para a classe Produto
  public Produto(int c, String d, double p, int q) {
    codigo     = c;
    designacao = d;
    precoUnit  = p;
    quantStock = q;
  }

  // outros métodos associados à classe produto
  public int entradaEmStock(int numUnidades) {
    quantStock = quantStock + numUnidades;
    return quantStock;
  }
}
```

```
import p1.*;
class Aula13_gestProdutos_vars extends P1App {
  static void main(String [] args) {

    int     cod, quant;
    double  preco;
    String  design;

    // declarar e inicializar um objecto da classe Produto
    Produto p1 = new Produto(1, "Computador portátil", 1100.00, 0);

    // declarar outro objecto da classe Produto
    Produto p2;                                                    (cont.)
```

```
    // ler os dados do 2º produto
    println("Produto #2");
    cod    = readInt   ("  . código do produto: ");
    design = readLine  ("  . designação       : ");
    preco  = readDouble("  . preço unitário   : ");
    println();

    // inicializar dados do 2º produto
    p2 = new Produto(cod, design, preco, 0);

    // mostrar valores dos atributos
    println("Produto #1:");
    println("  . código    : " + p1.codigo);
    println("  . designação: " + p1.designacao);

    println("Produto #2:");
    println("  . código    : " + p2.designacao);
    println("  . designação: " + p2.precoUnit);
   }
}
```

Supondo que a empresa comercializa apenas dois produtos, alterar o programa principal da classe `Aula13_gestProdutos_vars` a partir do comentário "`// mostrar valores dos atributos`", para que este passe implementar um pequeno sistema de gestão de produtos em stock, seguindo os passos enumerados abaixo. O programa deverá ter o seguinte menu de opções:

```
1. Listar produtos
2. Entrada de produto em stock
3. Venda de um produto
4. Produtos a encomendar
0. Terminar
```

a) Construir o menu de opções. Sugestão: consultar o exercício 1 das aulas 7 e 8.

b) Implementar a opção "`1. Listar produtos`". Esta opção deve listar todos os dados associados aos dois produtos no seguinte formato:

```
Produto #1
  . código              -> 1
  . designação          -> Computador portátil
  . preço unitário      -> 1100,00
  . quantidade em stock -> 0
Produto #2
  ...
> prima <enter> para continuar...
```

c) Implementar a opção "`2. Entrada de produto em stock`" da seguinte forma:

```
> código do produto: 2
> quantidade       : 10
. stock actualizado: 10 unidades
> prima <enter> para continuar...
```

(se o utilizador introduzir um código válido)

```
> código do produto: 22
> erro: código inválido
> prima <enter> para continuar...
```

(se o utilizador introduzir um código inválido)

Os valores sublinhados devem ser introduzidos pelo utilizador via teclado. Usar o método `entradaEmStock()` para actualizar a quantidade em stock.

d) Implementar a opção "`3. Venda de um produto`" da seguinte forma:

```
> código do produto: 2
> quantidade       : 2
. valor a pagar    : 110.00€
> prima <enter> para continuar...
```

Os valores sublinhados devem ser introduzidos pelo utilizador via teclado. Se o utilizador introduzir um código de produto inválido o tratamento deve ser idêntico ao realizado na opção 2. Deve ainda acrescentar à classe Produto um novo método com a seguinte assinatura:

```
public double venda(int numUnidades)
```

Este método deve receber como parâmetro o nº de unidades vendidas. Deve verificar se o nº e unidades do produto em stock é suficiente para satisfazer o pedido. Nesse caso, o método deve devolver o valor a pagar (igual à quantidade vezes o preço unitário); caso contrário, deve devolver um valor negativo.

Este método deve ser usado no programa principal para actualizar a quantidade em stock após a venda de um produto.

e) A opção "`4. Produtos a encomendar`" deve apresentar uma listagem com os dados dos produtos a encomendar no seguinte formato:

```
Produto #1
  . código              -> 1
  . designação          -> Computador portátil
  . preço unitário      -> 1100,00
  . quantidade em stock -> 0
  . ponto de encomenda  -> 6
  > quantidade mínima a encomendar: 4 unidades
```

Esta opção deve ser implementada da seguinte forma:

- acrescentar um novo atributo à classe Produto com o nome pontoEncomenda. Este atributo representa o nº mínimo de unidades que deverão existir em stock de forma a evitar eventuais rupturas de stock.
- acrescentar um novo método à classe produto com a seguinte assinatura:

  ```
  public int verificarPontoEncomenda()
  ```

  Se *n = pontoEncomenda – quantStock* for um valor positivo, então o método deve devolver o valor de `n` que corresponde ao nº mínimo de unidades a encomendar para regularizar o stock; senão deve devolver o valor zero.

  Este método deve ser usado no programa principal para determinar se é necessário ou não lançar a encomenda de um produto.

- actualizar o construtor da classe Produto.
- incluir o atributo *pontoEncomenda* nas operações de leitura e escrita dos dados de um produto.

2. Copiar e compilar a classe apresentada abaixo. O nome do ficheiro deve ser igual ao nome da classe.

   A classe `Aula13_gestProdutos_v2` implementa um pequeno programa exemplo que define, lê e imprime um *array* de produtos. O programa apresentado trabalha apenas com 4 produtos, mas poderia ser facilmente modificado para trabalhar com um nº indeterminado de produtos.

   Notar que esta classe utiliza a classe Produto apresentada no exercício anterior.

```
import p1.*;
class GestProduto_arrays extends P1App {
  static void main(String [] args) {

    int     i, cod, quant;
    double  preco;
    String  design;

    // definir um array para armazenar 4 produtos
    Produto [] hardware = new Produto[4];

    // Ler os dados dos 4 produtos
    for(i=0; i<hardware.length; i++) {
      println("Produto #" +(i+1));
      cod     = readInt   ("  . código do produto: ");
      design  = readLine  ("  . designação       : ");
      preco   = readDouble("  . preço unitário   : ");
      hardware[i] = new Produto(cod, design, preco, 0);
    }

    // mostrar valores dos atributos
    println();
    for(i=0; i<hardware.length; i++) {
      println("Produto #" +(i+1));
      println(" . " + hardware[i].designacao + ", " +
                      hardware[i].precoUnit);
    }
  }
}
```

   a) Alterar o programa principal da classe Aula13_gestProdutos_v2 a partir do comentário "// mostrar valores dos atributos", para que este passe implementar um pequeno sistema de gestão de produtos em stock com as mesmas opções propostas no exercício 1.

   b) Alterar o programa para que passe a ser feito o registo dos movimentos de entrada e saída de produtos em stock num ficheiro com o nome "movimentos.txt". O ficheiro deve ter o seguinte formato:

```
2007-05-06   10:15:20    (+)   10    150
2007-05-06   10:17:31    (-)   6     2
...
```

   A 1ª coluna tem a data do movimento (ano, mês e dia), a 2ª coluna tem a hora (horas, minutos e segundos), a 3ª coluna tem um (+) caso se trate de uma entrada ou um (–) se for uma saída de produto em stock, a 4ª coluna tem o código do produto e a 5ª coluna tem a quantidade. Utilizar o caracter tab (`'\t'`) para separar as colunas.

A data do movimento deve ser obtida automaticamente a partir do sistema operativo, tal como exemplificado a seguir:

```
import Java.util.Calendar;
...
    Calendar c = GregorianCalendar.getInstance();

    println("Ano     : " + c.get(Calendar.YEAR));
    println("Mês     : " + c.get(Calendar.MONTH));
    println("Dia     : " + c.get(Calendar.DAY_OF_MONTH));

    println("Horas   : " + c.get(Calendar.HOUR_OF_DAY));
    println("Minutos : " + c.get(Calendar.MINUTE));
    println("Segundos: " + c.get(Calendar.SECOND));
```

c) Finalmente, alterar o programa para que passe a ler os dados dos produtos a partir de um ficheiro no início do programa e a guardar os dados actualizados no memo ficheiro no final do programa. Cada linha do ficheiro deve corresponder a um produto. Os atributos de cada produto devem ser separados usando o caracter tab (`'\t'`).

3. Construir uma versão simplificada de um programa de gestão de movimentos de contas bancárias com as seguintes opções:

```
1. Listar contas
2. Debitar
3. Creditar
0. Terminar
```

Deve utilizar as duas classes propostas abaixo.

Classe ContaBancaria

Deve permitir representar os atributos da conta bancária e os incluir métodos que implementam as operações creditar e debitar o saldo de uma conta;

atributos	
long	**nib** Número de identificação bancária.
double	**saldo** Saldo da conta.
string	**titular** Nome do titular da conta.

construtor
ContaBancaria(long número, string nome, double saldoInicial) Inicializa uma nova conta bancária com o *número* da conta e o *nome* do titular indicados como argumentos.

métodos	
double	**creditar**(double valor) Acrescenta o *valor* passado como argumento ao *saldo* da conta e devolve o valor do saldo actualizado.

double	**debitar**(double valor) Retira o *valor* passado como argumento ao *saldo* da conta e devolve o valor do *saldo* actualizado. Se o *saldo* da conta for inferior ao *valor* a debitar, a operação não deve ser realizada, ou seja, o *saldo* da conta mantém-se inalterado e o método deve devolver o valor -1.

Classe Aula13_gestContas

Deve implementar a interface com o utilizador.

Esta classe não tem atributos nem construtores associados. O programa deve começar por ler os dados a partir de um ficheiro de texto e armazenar os valores num *array* de objectos da classe ContaBancaria. Assumir que o número máximo de contas é 100. Em seguida deve apresentar o menu e processar os pedidos do utilizador até ser escolhida a opção terminar. No final, o programa deve gravar novamente o ficheiro com os saldos das contas actualizados. O ficheiro de texto deverá ter o seguinte formato:

```
25      2000  Alberto Pereira
32      200   Joana Sousa
...
```

Cada linha do ficheiro representa o registo de uma conta e as colunas representam sucessivamente o nib, o saldo e o nome do titular da conta. Utilizar o caracter tab (`'\t'`) para separar as colunas.

métodos	
void	**main**(String [ ] args) O programa principal deve: Declarar um *array* para armazenar até 100 contas bancárias; Preencher o *array* usando o método lerContas; Apresentar o menu e processar os pedidos do utilizador; Gravar o *array* actualizado usando o método gravarContas.
int	**lerContas**(ContaBancaria [ ] contas) Lê um ficheiro de texto e preenche um *array* com os dados das contas bancárias armazenados nesse ficheiro. Assume-se que o *array* contas recebido como argumento se encontra vazio. Se não for possível abrir o ficheiro, o programa deve devolver o valor -1; senão, devolve o valor zero, significando que a operação foi realizada com sucesso.
int	**gravarContas**(ContaBancaria [ ] contas) Cria novamente um ficheiro de texto com os dados actualizados das *contas* armazenadas no *array* passado como argumento. Se não for possível abrir o ficheiro, o programa deve devolver o valor -1; senão, devolve o valor zero, significando que a operação foi realizada com sucesso.
int	**menu**() Imprime o menu de opções no ecrã, lê e devolve a opção escolhida pelo utilizador..