Comparação de Strings

1 - Valores constantes do tipo String têm a mesma referência.
"XPTO" = = "XPTO" ➔ expressão com valor true

2 – Strings construídas em tempo de compilação são tratadas como valores constantes do tipo String.

String s1 = "XPTO";
String s2 = "XPTO"

s1 = = s2 ➔ expressão com valor true

3 – Strings construídas em tempo de execução têm referências distintas:

String s1 = new String(“XPTO”);
String s2 = new String (“XPTO”);

s1 = = s2 ➔ false

mas

s1.equals(s2) ➔ true

Ex.lo 1

```
public class Exemplo {

    static String s0;

    public static void setS0 (String s){

            s0 = s;

      }
```

// ainda na classe exemplo.

```
public static void main (String [] args) {
   String s = "XPTO";
   setS0 (s);

  System.out.println ( s + "    "  +  s0);     // XPTO   XPTO


  System.out.println ( s = =  s0);            //     true

  s = "XX";           //é criada uma nova instância da String s;

  System.out.println ( s + "    "  +  s0)      //  XX  XPTO

}
```

E se no método **setS0**

substituirmos

s0 = s

por **s0= new String (s);**

O que acontece?

s == s0 ??? // false

s.equals (s0) ??? // true

```
public static void main(String[] args) {
    String s1 = new String ("XPTO");
    String s2 = "XPTO";
    System.out.println ( s1 == s2);
    System.out.println ( s1.equals(s2) );
    String s3 = "XPTO";
    System.out.println ( s2 == s3);
    System.out.println ( s2.equals(s3) );
}
```

Qual o output do programa?

false
true
true
true

Regra:

- Comparar Strings sempre com o método equals definido na classe String.

Listas Dinâmicas

A principal limitação dos arrays resulta do seu carácter estático.
É necessário estabelecer a dimensão do array aquando da sua definição e não é possível exceder este limite máximo.

A classe ArrayList disponível no pacote java.util distingue-se dos arrays pelas seguintes características:

Listas Dinâmicas

- Uma ArrayList pode crescer ou decrescer de tamanho;

- Uma ArrayList armazena objetos (os tipos primitivos são "embrulhados" em objetos... Lembram-se das classes Integer, Double,...?).

- Uma ArrayList pode conter objetos de diferentes tipos.

Listas Dinâmicas

Em conclusão, a classe ArrayList implementa uma abstracção de dados que representa uma estrutura linear indexada a partir do índice 0 (deste ponto de vista, análoga ao array) sem limite de dimensão.

Alguns métodos da classe ArrayList:

Java.util.ArrayList

ArrayList() // construtor sem parâmetros , dimensão inicial zero.

boolean **add**(Object element)
// adiciona o elemento especificado ao final da lista

void **add**(int index, Object obj)
//insere o elemento especificado na posição index

Object **remove**(int index)//remove o elemento da posição index

boolean **remove**(Object o)
//remove a primeira ocorrência do objeto dado como parâmetro

Java.util.ArrayList

Object **set** (int index, Object obj)
// substitui o elemento da posição index pelo elemento dado

Object **get** (int index)//devolve o elemento da posição index

void **clear**() // remove todos os elementos da lista

Object **clone**() // devolve uma cópia da lista

boolean **contains**(Object element)
// devolve true se a lista contém o elemento especificado

Java.util.ArrayList

boolean **equals** (Object obj)
// permite comparar duas listas

int **indexOf**(Object element)
// procura o índice da 1ª ocorrência de elemento

boolean **isEmpty**() // verifica se a lista não tem componentes

int **size**() // devolve a dimensão actual

String **toString** ()

Exemplo

```
import java.util.ArrayList;

public class Teste{
public static void main (String [] args) {

ArrayList  lista = new ArrayList();
lista.add( “Maria”);
lista.add (“João”);
String s =  (String) lista.get(0);
System.out.println (lista.toString()  + “ , ”+   s);
}
```

Output?

Output?

[Maria, João] , Maria

A ArrayList pode conter objetos de qualquer tipo, não havendo verificação de tipos. A partir da versão 5 do Java, a verificação de tipos pode ser feita durante a compilação, usando **tipos genéricos**:

Um tipo genérico é um tipo referenciado que usa na sua definição um ou mais tipos de dados como parâmetros.

Tipos genéricos

Seja o tipo ArrayList <E> em que **E** pode ser qualquer classe (ou interface!!)

A instanciação de um tipo genérico para um valor concreto de E, dá origem a um tipo parametrizado.

Exemplos:

```
ArrayList <String>  lista1;  // lista de objetos do tipo String

ArrayList<Aluno>  lista2 // lista de objetos do tipo Aluno
```

Tipos genéricos

Por exemplo, o código :

```
ArrayList <String>   lista1;
lista1 = new ArrayList <String> ();
lista1.add ("Joana");
lista1.add ("Manuel");

String ss = lista1.get(0);
System.out.println (lista1.toString() + " ," + ss);
```

Usa o tipo Parametrizado ArrayList <String>, com verificação estática de tipos (isto é, verificação de tipos em tempo de compilação).

Tipos genéricos

Percorrer uma ArrayList:

```
ArrayList <String>  lista1;
   lista1 = new ArrayList <String> ();
   lista1.add ("Joana");
   lista1.add ("Manuel");
  ...

  for (int i = 0; i < lista1.size(); i++) {

       System.out.println (    lista1.get(i)  );
}
```

Exercício:

Construa a classe Pessoa. Uma pessoa tem um nome, um número de identificação fiscal e um conjunto de contactos (objeto do tipo ArrayList de objetos do tipo Telefone (ver T04, página 13).

- Defina um construtor que receba o nome como parâmetro;
- Defina os getters;
- Defina os setters;
- Defina o método toString;
- **Defina um método que consulte o número do telefone móvel da pessoa;**

Recordando:

```
public class Telefone {
    private String tipo;   // Casa | Emprego | Móvel | ...
    private int numero;

  public Telefone (){   ...}
  public Telefone (String tipo, int numero){  ...}
  public String getTipo (){  ...}
  public int getNumero (){  ...}
  public void setTipo (String tipo){  ...   }
  public void setNumero (int numero){  ...}
  public String toString (){
      String s = "Tipo: " + tipo + " Número: " + numero;
      return s;
  } }
```

```
public class Pessoa {
    private String nome;
    private long NIF;
    private  ArrayList<Telefone>    contactos;

    public Pessoa(String nome) {
        this.nome = nome;
        NIF = 0;
        contactos = new ArrayList<Telefone>();   instanciar contactos
    }
     public String toString() {
        String s = "Nome: " + nome + " Nif: " + NIF + " Contactos: " +
                                                        contactos;  //???
        return s;
    }
```

```
import java.util.ArrayList;

public class T04a {

public static void main(String[] args) {
        Pessoa p;
        p = new Pessoa ("António Costa");
        System.out.println(p.toString());
    }
```

Output:

```
Nome: António Costa Nif: 0 Contactos: []
```

Voltando à classe Pessoa:

```
public String getNome() {   return nome;   }
public void setNome(String nome) {  this.nome = nome;    }
public long getNIF() {   return NIF;   }
public void setNIF(long NIF) {   this.NIF = NIF;   }
```

```
public  ArrayList<Telefone>   getContactos() {
    return contactos;
}

public void setContactos(ArrayList<Telefone> contactos) {
    this.contactos = contactos;   //!! Só copia o endereço
}
```

Voltando à classe Teste:

```
…
Telefone t1 = new Telefone ("Móvel", 96123456);
Telefone t2 = new Telefone ("Casa" , 275123456);
//ArrayList<Telefone>  contact = new ArrayList<Telefone>();
// Podemos simplificar,
ArrayList<Telefone> contact = new ArrayList<>();
contact.add(t1);
contact.add(t2);

p.setContactos(contact);
System.out.println(p);
```

//Output:

Nome: António Costa Nif: 0 Contactos: [Tipo: Móvel Número: 96123456, Tipo: Casa Número: 275123456]

Voltando à classe Teste:

…

E, se fizermos

```
t1.setTipo("XXXXXXX");
t1.setNumero(222222222);
System.out.println(p);
```

O que acontece com a Pessoa p?

System.out.println(p);

Nome: António Costa Nif: 0 Contactos: [Tipo: XXXXXXX Número: 222222222, Tipo: Casa Número: 275123456]

Voltando à classe Teste:

…

Se queremos reusar a variável t1, podemos evitar alterar o telefone da Pessoa, se voltarmos a instanciar o telefone t1 com os novos valores:

```
t1 = new Telefone("XXXXXXXX", 222222222);
System.out.println(p);
```

Nome: António Costa Nif: 0 Contactos: [Tipo: Móvel Número: 96123456, Tipo: Casa Número: 275123456]

A pessoa p não será alterada.

Exercício: **Após estudarmos o método clone, redefinir o método setContactos da classe Pessoa !!**

Exercício:

Para a classe Pessoa, defina um método que consulte o número do telefone móvel da pessoa;

```
public int numeroTlm (){
    int n = 0;
    for (int i = 0; i < contactos.size(); i++) {
        if (contactos.get(i).getTipo().equals("Móvel"))
            return contactos.get(i).getNumero();
    }
    return n;
}
```

Exercício:

- **Para a classe de Teste, construa um método de classe (public static) que dada uma ArrayList de objetos do tipo Pessoa devolva o número de pessoas que têm telemóvel.**

```
public class Teste{
…
public static int pessoasTelemovel ( ArrayList<Pessoa> pessoas){
        Pessoa p;
        int tlm = 0;
        for (int i = 0; i < pessoas.size(); i++) {
            p = pessoas.get(i);
            ArrayList<Telefone> contactos;
            contactos = p.getContactos();
            for (j = 0; j< contactos.size(); j++)  {
                  if ( contactos.get(j).getTipo().equals ("Móvel")  )
                      tlm ++;
             } }
        return tlm;
    }
 … main …
```

Exercícios (*1ª Frequência 2017/18, parte prática*)

1- Uma Pergunta de um teste pode ser representada por um identificador único (atributo **numero**), um **texto** com o conteúdo da pergunta, e um **valor** entre 0 e 20 correspondente à cotação da pergunta. A listagem abaixo apresenta o cabeçalho da classe Pergunta e a declaração dos seus atributos. Pretende-se que o número das perguntas seja gerado de forma automática usando o atributo **ultimo** para armazenar o último número de pergunta gerado.

```
public class Pergunta {
   private static int ultimo =0;
   private int numero;
   private String texto;
   private double valor;
```

Exercícios (*1 ª Frequência 2017/18, parte prática*)

a) Para a classe Pergunta defina o construtor sem parâmetros.

b) Para cada atributo da classe Pergunta defina os getters e os setters.

c) Para a classe Pergunta defina o método toString.

Exercícios (*1ª Frequência 2017/18, parte prática*)

2 – Uma frequência pode ser representada pelo nome da respectiva **disciplina**, por uma lista de perguntas (atributo do tipo array de objetos do tipo Pergunta) e ainda um atributo com a **data** da frequência (atributo do tipo LocalDate (pode consultar a API da classe LocalDate no final do teste). Supondo a declaração abaixo,

```
import java.time.LocalDate;

public class Frequencia {
   private String disciplina;
   private LocalDate data;
   private Pergunta [] perguntas;
```

Exercícios (*1ª Frequência 2017/18, parte prática*)

a) Defina um construtor que receba como parâmetros o nome da disciplina e o número de perguntas do teste. Deve inicializar o atributo data com o valor da data do sistema operativo.

b) Defina o get e o set do atributo perguntas. Para o resto do teste, supomos que os restantes getters e setters estão definidos.

c) Defina o método "to String" para a classe Frequencia.

d) Para a classe Frequencia defina um método, **totalCot**, que calcule a soma das cotações de todas as perguntas do teste.

Classe LocalDate:

Nota: a classe LocalDate é imutável, não possui um construtor público. Para criar um objeto do tipo LocalDate pode usar entre outros os métodos de classe **now** e **of** descritos abaixo.

public static LocalDate **now**()
- devolve a data actual do sistema operativo da sua máquina, no formato: aaaa-mm-dd;

public static Localdate **of** (int aaa, int mm, int dd)
permite criar um novo objeto do tipo LocalDate com o valor aaa-mm-dd;

Métodos de instância:

public String toString()
- devolve uma String com a data no formato aaaa-mm-dd. ...

e) Para o programa abaixo, desenhe **as variáveis que existem**, indique o seu **valor** e o **output** do programa:

```
public static void main(String[] args) {
        Pergunta p1, p2;
        p1 = new Pergunta();
        p1.setTexto("O que é uma variável de instância");
        p2 = p1;
        p2.setTexto("O que é um construtor");
        p2.setValor(6.0);
        Pergunta p3 = new Pergunta();
        p3.setTexto("O que é um modificador de acesso");
        p3.setValor(8.0);
        Pergunta [] pp = new Pergunta[3];
        pp[0] = p1;    pp[1] = p2;   pp[2] = p3;
        LocalDate dataFreq = LocalDate.of(2022, 10, 12);
```

e) Para o programa abaixo, desenhe **as variáveis que existem**, indique o seu **valor** e o **output** do programa:

```
    …
    Frequencia f = new Frequencia ("POO", 3);
    f.setPerguntas(pp);
    f.setData(dataFreq);
    System.out.println(f);
    System.out.println(f.totalCot());
}
```

Resolução:

1- a)

```
public class Pergunta {
   private static int ultimo =0;
   private int numero;
   private String texto;
   private double valor;


public Pergunta (){
       ultimo++;
       numero = ultimo;
       texto = "";
        double = 0.0;
   }
```

Resolução:

1- b)

b)

```
 public static int getUltimo() {
       return ultimo;
   }
   public static void setUltimo (int ultimo) {
       Pergunta.ultimo = ultimo;
   }

   public int getNumero() {   return numero; }

   public void setNumero(int numero) { this.numero = numero;    }
```

Resolução:

1- b) …

```
   public String getTexto() {   return texto;  }

    public void setTexto(String texto) {  this.texto = texto;     }

   public double getValor() {  return valor;     }

   public void setValor(double valor) {    this.valor = valor;   }

    public String toString() {
      return "Pergunta{" + "numero=" + numero + ", texto=" + texto + ",
                                                          valor=" + valor + '}';
   }
} // fim da classe
```

Resolução:

2- a)

```
public class Frequencia {
    private String uc;
    private LocalDate data;
    private Pergunta [] perguntas;

    public Frequencia (String uc, int dim){
        this.uc = uc;
        data = LocalDate.now();
        perguntas = new Pergunta[dim];
        for (int i = 0; i < perguntas.length; i++) {
            perguntas[i] = new Pergunta();
       }
    }
```

Resolução:

2- b)

```
 public String getUc() {     return uc;     }

public void setUc(String uc) {      this.uc = uc;      }

public LocalDate getData() {       return data;     }

public void setData(LocalDate data) {
                this.data = data;}

public Pergunta[] getPerguntas() {      return perguntas;       }
```

Resolução:

2- b)

```
public void setPerguntas(Pergunta[] perguntas) {
    for (int i = 0; i < perguntas.length; i++) {
        this.perguntas[i].setNumero ( perguntas[i].getNumero() );
        this.perguntas[i].setTexto ( perguntas[i].getTexto() );
        this.perguntas[i].setValor ( perguntas[i].getValor() );
    }
}
```

Resolução:

2- c)

```
public String toString() {
    String s=  "Frequencia:" + "uc=" + uc + ", data=" + data + ",
           perguntas=" ;
           for (int i = 0; i < perguntas.length; i++) {
                                s = s + ","  + perguntas[i];
           }
    return s;
 }
```

Resolução:

2- d)

```
public double totalCot (){
    double total = 0;

    for (int i = 0; i < perguntas.length; i++) {
     total= total +  perguntas[i].getValor();
    }

    return total;

}
```

Exercício (extra teste):

Resolva agora o exercício 2, mas supondo que a lista de perguntas da classe Frequencia é uma ArrayList de objetos do tipo Pergunta em vez de um array.

Outros métodos:

- Qual a pergunta que tem mais cotação?
- Quais as perguntas que contêm uma dada palavas dada como parâmetro