FICHA DE EXERCÍCIOS – 05 – POLIMORFISMO

1. Tipos estáticos e tipos dinâmicos duma variável
2. Tempos de programação e de execução
3. Princípio da substituição
4. Dynamic binding
5. Upcasting e downcasting
6. Programação genérica
7. Informações sobre a classe: getClass()
8. Informações sobre o objeto: instanceOf()

5.1. ARQUIVO DE EMPRESA

Pretende-se criar um **Arquivo** em que possam ser registados objetos que são Pessoas criadas na Ficha de Exercícios nº 3 – Herança (Pessoa; Fornecedor; Empregado; Administrador; Operário; Vendedor). Execute as seguintes tarefas:

5.1.1. Implementar a classe Arquivo, baseada num ArrayList, que possa receber objetos que são Pessoas. Comece por:

a. Povoar o arquivo com objetos dos vários subtipos de Pessoa.

De seguida, desenvolva o código necessário para realizar as seguintes operações:

b. Saber quantos Pessoas há no arquivo.

c. Listar as Pessoas do arquivo.

d. Listar o nome de todas as pessoas do arquivo.

e. Saber quantos Fornecedores há no arquivo.

f. Listar os Fornecedores do arquivo.

g. Saber quantos Empregados há no arquivo.

h. Saber quantos Operários e Vendedores há no arquivo.

i. Listar a comissão dos Operários do Arquivo.

5.1.2. Na Main, instancie um objeto da classe Arquivo e prove que as operações estão funcionais.

5.2. Bar

No SLB, Só Líquidos Bar, apenas se servem Cafés e Águas. O Bar está organizado por mesas e tem consumo mínimo: quando alguém se senta numa mesa, é-lhe debitado automaticamente o consumo de 1 Café e de 1 Água.

Sobre cada mesa é mantida a seguinte informação: Nome do cliente, nº de Cafés consumidos e nº de Águas consumidas. Depois de se sentar à mesa, e para além do Café e da Água que são automaticamente servidos, o cliente pode pedir mais Cafés (1 ou mais de cada vez) e pode também pedir mais Águas (1 ou mais de cada vez). Claro que o cliente pode englobar, num mesmo pedido, Cafés e Águas.

Há mesas especiais que, em vez de toalha de papel, usam uma toalha de veludo vermelho com um logotipo do SLB e que têm ainda uma vela aromática acesa, para criar ambiente. Nestas mesas o consumo mínimo é de 2 Cafés e de 2 Águas e, quanto a pedidos de Cafés e de Águas, o cliente tem de pedir 2 ou mais de cada vez. É, obviamente, uma mesinha para dates 😉

Deve ser possível, considerando todas as mesas:

5.2.1. Aceder, por nome do cliente, aos dados de uma mesa;

5.2.2. Dadas duas mesas, aceder à mesa em que se consumiu mais cafés;

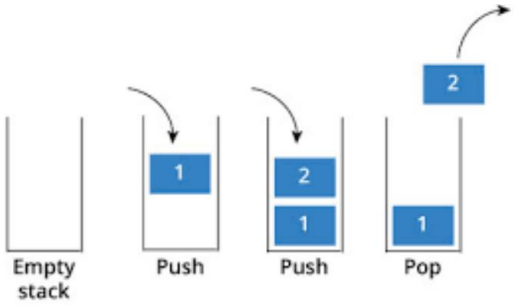
5.2.3. Saber quantas mesas normais estão montadas;

5.2.4. Listar o nome de todos os parzinhos em dating time.

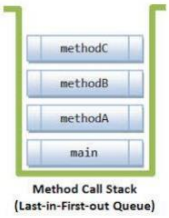
5.2.5. Calcular o nº total de cafés fornecidos a todas as mesas.

5.3. Jardim Zoológico

A Stack é uma fila de espera de elementos. Esses elementos são registados e recuperados num sistema LIFO (Last In, Fisrt Out), tal como a imagem representa:



Há vários casos em que este modelo é aplicado com sucesso:



Pretende-se criar uma **Stack** em que possam ser registados objetos que são os Animais de um Zoo. Esse Zoo tem duas categorias de animais: selvagens e domésticos:

- Os selvagens, que indicam o meio que habitam (terrestre, aquático, …) e que podem ser o Leão (que indica o Continente) e o Crocodilo (que indica o Oceano).
- Os domésticos, que indicam se são herbívoros ou carnívoros e que podem ser o Cão (que indica a raça: Labrador, …) ou a Ovelha (que indica o peso).

Todos os animais têm um nome e comunicam de alguma forma (zurrar, …).

Pretende-se:

a) Criar o diagrama das classes adequado para retratar este problema.

b) Criar a stack Zoo, baseada num ArrayList, que possa receber objetos que são animais.

c) Preencher o Zoo com animais.

d) Saber quantos animais há no Zoo.

e) Listar os animais do Zoo.

f) Saber o nome de todos os animais.

g) Ouvir a voz de todos os animais do Zoo.

h) Saber quantos animais selvagens são terrestres e quantos são aquáticos.

i) Saber o peso total das ovelhas.

5.4. Exercício de exame

Apresente o output gerado por cada um dos métodos (use as classes definidas no anexo). Se um método gerar um erro, caracterize-o (compilação / execução) e justifique a resposta.

public static void exercicio_1(){
System.out.println("====== Exercício 1 =====");
LetraB obj1 = new LetraF();
LetraE obj2 = (LetraE) obj1;
System.out.println(obj2.mostra());
}	

public static void exercicio_2(){
System.out.println("====== Exercício 2 =====");
LetraE obj1 = (LetraE) new LetraF();
LetraF obj2 = (LetraF) obj1;
System.out.println(obj2.mostra());
}	

public static void exercicio_3(){
System.out.println("====== Exercício 3 =====");
LetraF obj1 = new LetraE();
System.out.println(obj1.mostra());
}	

public static void exercicio_4(){
System.out.println("====== Exercício 4 =====");
LetraB obj1 = (LetraC) new LetraF();
System.out.println(obj1.mostra());
System.out.println(((LetraC) obj1).mostra());
}	

public static void exercicio_5(){
System.out.println("====== Exercício 5 =====");
LetraF obj1 = (LetraF) new LetraE();
System.out.println(obj1.mostra());
}	

public static void exercicio_6(){
System.out.println("====== Exercício 6 =====");
LetraB obj1 = (LetraC) new LetraD();
System.out.println(obj1.mostra());
}	

public static void exercicio_7(){
System.out.println("====== Exercício 7 =====");
LetraE obj1 = (LetraE) new LetraC();
System.out.println(obj1.mostra());
}	

public static void exercicio_8(){
System.out.println("====== Exercício 8 =====");
LetraC obj1 = new LetraE();
System.out.println(obj1.mostra());
}	

public static void exercicio_9(){
System.out.println("====== Exercício 9 =====");
LetraC obj1 = (LetraB)((LetraC) ((LetraB) new LetraD()));
System.out.println(obj1.mostra());
}	

public static void exercicio_10(){
System.out.println("====== Exercício 10 =====");
LetraB obj1 = new LetraF();
LetraE obj2 = obj1;
System.out.println(obj2.mostra());
}	

public static void exercicio_11(){
System.out.println("====== Exercício 11 =====");
LetraF obj1 = new LetraF();
System.out.println(obj1.mostra());
}	

public static void exercicio_12(){
System.out.println("====== Exercício 12 =====");
LetraE obj1 = new LetraC();
System.out.println(obj1.mostra());
}	

ANEXO

```
public abstract class LetraA {
	protected char cA;

	public LetraA() { cA = 'A' ; }

	@Override
	public abstract String toString() ;

	public String mostra() {
		return " / " + "cA = " + cA  ;
	}
}
```

```
public class LetraB extends LetraA{
	protected char cB;

	public LetraB() { cB = 'B' ; }

	@Override
	public String toString() {
		return mostra();
	}

	public String mostra() {
		return " / " + "cB = " + cB ;
	}
}
```

```
public abstract class LetraC extends LetraB{
	protected char cC;

	public LetraC() { cC = 'C' ; }

	@Override
	public abstract String toString() ;

	@Override
	public String mostra() {
		return " / " + "cC = " + cC + " / " +
			super.mostra();
	}
}
```

```
public class LetraD extends LetraC{
	protected char cD;

	public LetraD() { cD = 'D' ; }

	@Override
	public String toString() {
		return null;
	}

	@Override
	public String mostra() {
		return  " / " + "cD = " + cD + " / " +
			super.mostra();
	}
}
```

```
public class LetraE  extends LetraC{
	protected char cE;

	public LetraE() { cE = 'E' ; }

	@Override
	public String mostra() {
		System.out.println(">> " + cE);
		return super.mostra() + " / " + "cE = " + cE + " /
		";
	}

	@Override
	public String toString() {
		return mostra();
	}
}
```

```
public class LetraF  extends LetraE{
	protected int cF;

	public LetraF() { cF = 'F' ; }

	@Override
	public String mostra() {
		return super.mostra() + " / " + "cF = " + cF ;
	}
}
```

public abstract class LetraA {
protected char cA;
public LetraA() { cA = 'A' ; }
@Override
public abstract String toString() ;
public String mostra() {
return " / " + "cA = " + cA ;
}
}
<<Java Class>>
LetraA
(default package)
cA: char
LetraA()
toString():String
mostra():String
public class LetraB extends LetraA{
protected char cB;
public LetraB() { cB = 'B' ; }
@Override
public String toString() {
return mostra();
}
public String mostra() {
return " / " + "cB = " + cB ;
}
}
<<Java Class>>
LetraB
(default package)
cB: char
LetraB()
toString():String
mostra():String
public abstract class LetraC extends LetraB{
protected char cC;
public LetraC() { cC = 'C' ; }
@Override
public abstract String toString() ;
@Override
public String mostra() {
return " / " + "cC = " + cC + " / " + super.mostra();
}
}
<<Java Class>>
LetraC
(default package)
cC: char
LetraC()
toString():String
mostra():String
public class LetraD extends LetraC{
protected char cD;
public LetraD() { cD = 'D' ; }
@Override
public String toString() {
return null;
}
@Override
public String mostra() {
return " / " + "cD = " + cD + " / " + super.mostra();
}
}
<<Java Class>>
LetraD
(default package)
cD: char
LetraD()
toString():String
mostra():String
public class LetraE extends LetraC {
protected char cE;
public LetraE() { cE = 'E'; }
@Override
public String mostra() {
System.out.println(">> " + cE);
return super.mostra() + " / " + "cE = " + cE + " / ";
}
@Override
public String toString() {
return mostra();
}
}
<<Java Class>>
LetraE
(default package)
cE: char
LetraE()
mostra():String
toString():String
public class LetraF extends LetraE{
protected int cF;
public LetraF() { cF = 'F' ; }
@Override
public String mostra() {
return super.mostra() + " / " + "cF = " + cF ;
}
}
<<Java Class>>
LetraF
(default package)
cF: int
LetraF()
mostra():String