Exercícios Revisão Programação Orientada a objetos.

1) Métodos construtores são utilizados para inicializar atributos de instância. Analise a classe abaixo.

```
public class Cliente {

        private String nome;
        private double peso;

        //get/set
}
```

Qual dos métodos abaixo não pode ser considerado um método construtor da classe Cliente? Justifique.

A)
```
protected Cliente(){}
```

B)
```
private Cliente(String nome){
        this.nome = nome;
}
```

C)
```
Cliente(double peso){
        this.peso =peso;
}
```

D)
```
void Cliente(double peso){
        this.peso =peso;
}
```

E)
```
public Cliente(String nome2,
double peso){
        nome = nome2;
        this.peso =peso;
}
```

Em Java, o construtor não pode possuir do tipo de retorno. Caso o tipo de retorno seja informado o membro será um método de instância.

2) Analise as duas classes abaixo.

```
package br.aula.oo;

public final class Cliente {

        protected String cpf;

}
```

```
package br.aula;

import br.aula.oo.Cliente;

public class Serasa {

        public boolean consultar(Cliente cliente){

                return verificaCPF(cliente.cpf);
        }
```

	//Método verificaCPF(String) }

A classe Serasa é responsável por verificar se o CPF de um cliente não possui pendências. Lembrando que o atributo CPF na classe Cliente é *protected*. A classe Serasa **não** possui acesso ao atributo CPF gerando um erro de compilação na linha "**return** verificaCPF(cliente.cpf);". Qual solução você propõe para a classe Serasa acessar o atributo CPF da classe Cliente, uma vez que você não pode alterar a classe Cliente?

A classe Serasa não pode estender a classe Cliente, pois ela é final. A solução seria alterar o package da classe Serasa para br.aula.o

3) Um erro muito comum no desenvolvimento de software é o "*NullPointerException*", que acontece quando tentamos acessar algum membro de instância através de uma referência, que não está apontando para nenhum objeto em memória. Considere a declaração do método abaixo:

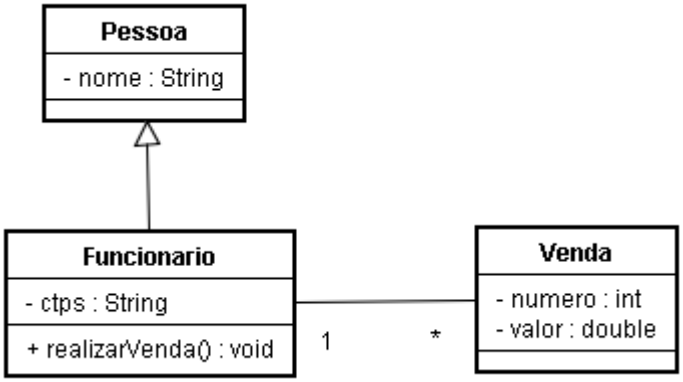
```
public static boolean consultar(Cliente cliente){
        return verificaCPF(cliente.cpf);
}
```

O código abaixo irá compilar? Justifique.

```
Serasa s = null;
s.consultar(new Cliente("0000"));
```

Sim, o método consultar é estático então não é necessária uma instância para executá-lo.

4) Em Java podemos reusar uma classe através de uma composição e de herança. Crie as classes e associações de acordo com o diagrama de classe abaixo:



public class Pessoa
{
  private String nome;
 	
//Métodos get/set
}
	public class Funcionario extends Pessoa
	{
	  private String ctps;
	
	  private List<Venda> vendas;
	
	//Métodos get/set
	
	  public void realizarVenda()
	  {
	  ...implementação
	  }
	}

public class Venda
{
  private int numero;
  private double valor;
  private Funcionario funcionario;
 	
//Métodos get/set
}	

5) Classes abstratas não podem ser instanciadas, elas apenas podem ser superclasses e definir métodos abstratos para a implementação nas subclasses. Qual classe abstrata abaixo é inválida? Justifique

A) **public abstract final class** Boleto { B) **public abstract class** Boleto {

```
        public abstract int
    gerarNumero();
    }
```

C)
```
public abstract class Boleto {

        public int gerarNumero(){
                return
    (int)(Math.random() * 1000);
        }
    }
```

E)
```
public abstract class Boleto {


    }
```

D)
```
public abstract class Boleto {

        protected abstract int
    gerarNumero();
    }
```

Não podemos ter uma classe abstrata como final, pois a classe abstrata é utilizada apenas em herança..

6) Considerando-se o padrão de projetos para interfaces, é CORRETO afirmar que os métodos das interfaces:
   a) Podem ser públicos ou privados, dependendo da declaração do programador.
   b) São sempre privados.
   c) São públicos some se declarados explicitamente; caso contrário, são considerados privados.
   d) São sempre abstratos e públicos, independentemente de haver uma declaração explícita para isso.

7) O polimorfismo permite que uma referência de objeto tenha várias formas. Considere seguintes códigos:

```
public interface Boleto {

        String codigoBancario();
}
```

```
public class BoletoBancoBrasil implements Boleto {

        public void gerarCodigoBarra(){
```

```
	}

	@Override
	public String codigoBancario() {
		return "13";
	}
}


public class BoletoUnicred  extends BoletoBancoBrasil{

	@Override
	public String codigoBancario() {
		return "21";
	}
}
```

Qual será a saída ao executar esse método.

```
public static void main(String[] args) {

	Boleto b1 = new BoletoBancoBrasil();
	Boleto b2 = new BoletoUnicred();
	BoletoBancoBrasil b3 = new BoletoUnicred();

	System.out.println(b1.codigoBancario());
	System.out.println(b2.codigoBancario());
	System.out.println(b3.codigoBancario());

}
```

13

21

21

8) Ao criar um arquivo em disco podem ocorrer alguns erros, tais como, o caminho não existe ou você não possui permissão de escrita. O código abaixo permite a criação de um arquivo em disco, realize o tratamento de exceção para caso um erro ocorra o software não pare de funcionar**?**

```
File file = new File("D:\\arquivo.pdf");

file.createNewFile();
```

File file = new File("D:\\arquivo.pdf");

try {

file.createNewFile();

} catch (IOException ex) {

//Tratamento do erro

}

CORRETO

9) *Collection* é uma estrutura Java que permite armazenar elementos dinamicamente. Qual a diferença entre as interfaces *List*, *Set* e *Map*.

List	Set	Map
É uma coleção não ordenada, permite elementos duplicados e ordenada por ordem de inserção.	**É uma coleção ordernada, não permite elementos duplicados e ordenada pelo código hash.**	**Um mapa é, abstratamente, um conjunto de mapeamentos, ou associações, entre um objeto chave e um objeto valor-associado, onde as chaves são únicas.**

10) Ao criar uma instância de uma coleção devemos definir o tipo de objeto da coleção (*Generics*). Qual seria o impacto na definição de uma coleção sem a definição do tipo do objeto (Sem o uso do *Generics*), como no exemplo abaixo?

```
Set elementos = new HashSet();
```

Sem o uso do Generics a Collection criada poderá aceitar qualquer item do tipo Object, podendo ocasionar erros se adicionado tipos diferentes como, “String” e “Integer”.