Exercícios

Nos exercícios abaixo, onde aparecer "Java", você poderá também usar C++/STL.

1. Defina o que é feito na etapa de *Análise* e o que é feito na etapa de *Projeto* ao desenvolver software.

   Nesta fase é denominada por um composto de análises por exemplo: análise e especificação de requisitos, análise de riscos a fim de guiar um planejamento que será realizado durante toda a evolução de software.

   Na fase de projeto, parte do princípio de: conhecer as necessidades do cliente, avaliar a viabilidade do projeto, documentar todos os procedimentos, escolher a metodologia de desenvolvimento, testar as funcionalidades criadas.

2. Enumere as vantagens da abordagem Orientada a Objetos para o desenvolvimento de software.

   1 - Maximiza o reaproveitamento de código.

   2 - Dividir para conquistar.

   3 - Maior manutenabilidade

   4 - Design do código mais arrojado.

3. Qual é o motivo de levantar Requisitos Funcionais para desenvolver software e o que faz parte de uma descrição de Requisitos Funcionais?

   O motivo: definir a função de um sistema ou seu componente.

   Descrição: o que deve ser feito.

4. Mostre como instanciar um objeto da classe ContaBancária em Java fornecendo o CPF (um string) do titular como argumento. Com o objeto resultante, faça um depósito de R$100,00 e imprima o saldo. Você pode escolher nomes apropriados para os métodos.

```
public class ContaBancaria {
	private String cpf;
	private double saldo;

	public ContaBancaria(String cpf) {
		this.cpf = cpf;
		this.saldo = 0;
	}

public void depositar(double montante) throws ValorInvalidoException {

		if(montante > 0) {
			saldo += montante;
		} else {
			throw new ValorInvalidoException();
		}
	}
public double getSaldo(){
	return saldo;
}
public void setSaldo(double saldo) {
		this.saldo = saldo;
	}
```

```
public class ContaApp{
        public static void main (String[] args){
                ContaBancaria conta = new ContaBancaria(“999.999.999-11”);
                conta.depositar(100);
                system.out.println(conta.getSaldo());
}
}
```

5. Explique o que é um Iterator em Java. Qual é sua principal vantagem?

Padrão Iterator fornece uma maneira de acessar sequencialmente os elementos de um objeto agregado sem expor a sua representação subjacente. Portanto, o padrão Iterator permite acessarmos um a um os elementos de um agregado mesmo sem saber como eles estão sendo representados, assim torna-se irrelevante se a coleção de objetos está num ArrayList, HashTable ou que quer que seja. Além disso, o Padrão Iterator assume a responsabilidade de acessar sequencialmente os elementos e transfere essa tarefa para o objeto Iterador, dessa forma o objeto agregador tem a sua interface e implementação simplificadas, não sendo mais o responsável pela iteração.

6. Mostre a implementação de uma classe ContaBancária. Invente

atributos e métodos.

```
private String nome;
private int conta, saque;
private double saldo;

public Conta(String nome, int conta){
        this.nome=nome;
        this.conta=conta;
```

```
        saldo=0;
        saque=0;
    }
    public void sacar(double valor){
        if(saldo >= valor){
            saldo -= valor;
            saque++;
            System.out.println("Sacado: " + valor);
            System.out.println("Novo saldo: " + saldo);
        } else {
            System.out.println("Saldo insuficiente. Faça um
depósito");
    }
    }
     public void depositar(double valor){
        saldo += valor; System.out.println("Depositado: " + valor);
System.out.println("Novo saldo: " + saldo );
    }
    }
```

7. Explique a diferença de funcionamento entre um "return" e um "throw". Seja específico.

Return retorna um valor esperado pelo método ou função implementada na aplicação.

Throw é um tipo de retorno diferente, pois ele está preparado para alertar para possíveis erros na aplicação.

8. Mostre, usando Java, como especializar uma classe ContaBancária para criar uma ContaCorrente e uma ContaPoupança.

```
public final class ContaCorrente extends ContaBancaria {



    public static final double
TARIFA_MOVIMENTACAO_CONTA_CORRENTE = 0.5;
```

```
	public ContaCorrente(int agencia, int conta, String titular, double saldo)
{
		super(agencia, conta, titular, saldo);
	}

	public ContaCorrente(int agencia, int conta, String titular) {
		super(agencia, conta, titular);
	}

	@Override
	public void depositar(double montante) throws ValorInvalidoException
{
		montante = montante - getTributo();
		super.depositar(montante);
	}

	@Override
	public void sacar(double montante) throws SaldoInsuficienteException
{
		montante = montante - getTributo();
		super.sacar(montante);
	}
```

```
	@Override
	public double getTributo() {
		return TARIFA_MOVIMENTACAO_CONTA_CORRENTE;
	}

}
```

```
public class ContaPoupanca extends ContaBancaria{

	private static final double JUROS_CONTA_POUPANCA = 0.10;

	public ContaPoupanca(int agencia, int conta, String titular) {
		super(agencia, conta, titular);

	}

	@Override
	public double getTributo() {
		return JUROS_CONTA_POUPANCA;
	}

	@Override
```

```
    public void depositar(double montante) throws ValorInvalidoException
{
        montante = montante + getTributo();
        super.depositar(montante);
    }

    @Override
    public void sacar(double montante) throws SaldoInsuficienteException
{
        super.sacar(montante);
    }

}
```

9. Explique as vantagens e desvantagens do polimorfismo. Dê exemplos.

   Vantagens:

   Os objetos filhos herdam as características e ações de seus pais.

   Permite que vários objetos de um mesmo tipo sejam tratados da mesma maneira.

   Aumentar o software de maneira mais controlada.

Facilita a generalização de algoritmos e estruturas de dados.

Desvantagens:

Problema de acoplamento.

Dificuldade de reuso

Exemplo:

Objeto genérico que possui a ação de “ligar()”. Existe mais objetos que serão ligados de formas diferentes, então, para cada classe filha deve reescrever o método “ligar()” e implementar como desejar.

10. Explique a afirmação: "Em Java, o conceito de interfaces permite obter mais polimorfismo do que seria possível com classes abstratas".

Classe abstrata pode ter implementação e atributo, enquanto **interface**, até o **java** 7 só podia ter método abstrato(declaração de método sem corpo). No **java** 8 a **interface** pode ter métodos com implementação, mas com algumas restrições

11. Qual é a diferença entre "herança de tipo" e "herança de implementação"?
    Herança de tipo é capacidade de uma classe herdar uma interface
    Herança de estado é a capacidade de herdar o estado de outras classes através de seus atributos.

12. Quais são as vantagens e desvantagens de acoplamento forte entre objetos?
    Vatagens: Baixo acoplamento, alta coesão.
    Manutenção de código

    Desvantagens:
    Complexidade

13. Ao falar de boa programação, fala-se: "A decomposição deve esconder algo." O que poderia ser escondido, por exemplo?

    Ela restringe o acesso do objeto a tudo de uma classe possuir.

14. O que é uma "responsabilidade de uma classe"? Por que queremos minimizar o número de responsabilidades? "Mais" não seria melhor?

    Responsabilidade de uma classe é ela atende efetivamente a funçôes que estão ao seu compromisso como papel, garantindo uma consistência. Diminuindo o número de responsabilidades, quer dizer que está havendo uma boa programação por parte do programador, pois está diminuindo as dependências entre as classes.

15. Por que modelar papeis (roles) através de herança é inferior a modelá-los através de composição?

    Porquê herança torna o código muito frágil, já com composição, torna o código mais forte/consistente, garantindo uma futura manutenção sem afetar os demais objetos.