Aluno: Italo Pereira de Lima
Matrícula: 1223080033

Exercício 1

01) Analisar os requisitos levantados e projetar todo o software a partir deles (escolher tecnologias e serviços a serem utilizados – Prova de Conceito).

02) Reutilização, manutenção, custo e tempo, nível de abstração, padronização, segurança, etc.

03) Requisitos funcionais definem o comportamento do sistema.

04) ContaBancaria contaBancaria1 = new ContaBancaria(“123123”); contaBancaria1.setDeposito(1000.00); System.out.println(getSaldo);

05) É um padrão de projeto que 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.

06)
```
public class ContaBancaria{
private String nome;
private String cpf;
private double saldo;
public ContaBancaria(String nome, String cpf) {
super();
this.nome = nome;
this.cpf = cpf;
}
public void depositar(double valor){
saldo += valor;
}
public void sacar(double valor){
saldo -= valor;
}
public String getNome() {
return nome;
}
public void setNome(String nome) {
this.nome = nome;
}
public String getCpf() {
```

```
return cpf;
}
public void setCpf(String cpf) {
this.cpf = cpf;
}
public double getSaldo() {
return saldo;
}
public void setSaldo(double saldo) {
this.saldo = saldo;
}
}
```

07) O Throw para a execução do método e lança uma exceção, enquanto o Return retorna um valor.

08) Usando a classe ContaBancaria da questão 6, criamos mais duas classes:

```
public class ContaCorrente extends ContaBancaria {
public ContaCorrent(String nome, String cpf) {
super(nome, cpf);
// TODO Auto-generated constructor stub
}
}
public class ContaPoupanca extends ContaCorrent {
public ContaPoupanca(String nome, String cpf) {
super(nome, cpf);
}
}
```

09) Com polimorfismo podemos um maior controle sobre as subclasses geradas a partir de uma classe pai. Podemos dar vários comportamentos ao mesmo método.

10) Nas classes abstratas pelo fato de poder ter métodos implementados, eles não poderão ser modificados. Já na interface, as assinaturas de métodos serão implementadas pela classe que compôs da maneira que a classe quiser

11) Herança de classe (ou de implementação), define a implementação de um objeto em função da implementação de outro e Mecanismo para compartilhamento de código. Herança de tipo (ou de interface), define quando um objeto pode ser utilizado no lugar do outro ou cumprindo a mesma promessa que o outro prometeu.

12) A desvantagem é a dependência de um método para o funcionamento dos demais métodos. Já a vantagem é a possibilidade de deixar os objetos mais seguros e difíceis de serem modificados, aumentando a coesão.
14) Responsabilidade de uma classe é o comportamento que ela possui. Cada classe deve ter sua responsabilidade específica.
15) Porque através de composição eu estou acoplando menos o meu projeto e a classe que implementar poderá fazer os métodos da maneira que ele quiser.