Disciplina...: Programação Orientada a Objetos
Professor...: José Antonio Gonçalves

1º Questionário

Obs.: As questões desta lista de exercícios deverão ser respondidas ao longo da disciplina e entregues (via Moodle) ao final da disciplina. As respostas serão encontradas nos slides (inclusive nos já postados).
- As questões foram elaboradas com o intuito de ser um check-list de estudos.
- Poderão ser usadas na elaboração das provas;
- Caso tenha dúvidas a respeito de qualquer uma delas, por favor, me envie uma mensagem **via Moodle.**

---

1)_ Por que falamos que Java é totalmente aderente às técnicas de Orientação a Objetos?

R: Tirando os tipos primitivos, a linguagem Java nos obrigada a utilizar os paradigmas de Orientação a Objetos, além de ter fácil reutilização de código e possuir a representação do sistema muito mais perto do que veríamos no mundo real.

2)_ Explique o que é e como se utiliza o processo de “abstração”.

R: A abstração é utilizada para trazer para a programação Java, na qual são criadas as classes, definições de entidades do mundo real. Por exemplo: os dados podem ser abstraídos para seus componentes de objetos e um conjunto de passos pode ser a comunicação entre esses objetos. Sendo assim, cada objeto possui seu próprio comportamento e seriam entidades concretas capazes de responder a comandos de métodos.

3)_ Quais são os artefatos produzidos na programação orientada a objetos?
R: Classe, métodos, interface e exceções.

4)_ O que é um “tipo primitivo” de dados?
R: São tipos de dados que não representam classes, mas sim valores básicos.
Exemplo: byte, short, int, long, float, double, char ou boolean.

5)_ O que é um “tipo abstrato” de dados?

R: São tipos de dados definidos como uma especificação de um conjunto de dados e operações que podem ser executadas sobre esses eles mesmos.

6)_ Explique o que é o “Garbage Collector”. Como este recurso pode dinamizar o funcionamento do sistema?

R: O Java Garbage Collection é o processo pelo qual os programas Java executam o gerenciamento automático de memória. Quando os programas Java são executados na JVM, os objetos são criados no heap, que é uma parte da memória dedicada ao programa.

Eventualmente, alguns objetos não serão mais necessários. O garbage collector localiza esses objetos não utilizados e os exclui para liberar memória.

7)_ Considerando o **modo Shell** (linhas de comando) do sistema operacional Windows, como se faz para:

7.a) _ Compilar um código fonte Java;

R: javac nome do arquivo.java.

7.b) _ Fazer com que a J.V.M. (Máquina Virtual Java) execute uma aplicação Java.

R: java nome da classe.

8)_ O que é o “ByteCode”?

R: Bytecode é um formato de código intermediário entre o código fonte, o texto que o programador consegue manipular, e o código de máquina que o computador consegue executar.

9)_ Explique o que é a característica “Portabilidade”. Como isto é possível com aplicações Java? Para esta resposta relacione 4 “personagens” deste cenário: o código fonte (arquivo .java), o byteCode (arquivo .class), o Sistema Operacional e a JVM (Java Virtual Machine).

R: A característica de portabilidade significa que um sistema desenvolvido em Java poderá ser executado em qualquer sistema operacional. Como o Java compila o código fonte para um código intermediário que é o byteCode, esse não é atrelado a um sistema operacional. Portanto, quem é responsável por executar em um sistema operacional especifico é a JVM.

10) _ Justifique a afirmação que diz que “a segurança em Java se dá em dois níveis: proteção de hardware e proteção de software”.

R: A proteção em hardware acontece devido ao fato de um código Java sempre rodar em cima de uma maquina virtual (JVM). Já a proteção de software é feita atráves das analisas de código pré-compilação.

11)_ Explique como aplicamos o conceito de “Modularidade” em Java. Na resposta desta questão deve-se trata dos conceitos sobre “Acoplagem” e “Coesão”.

11.a)_ Como esta característica pode ajudar na questão da “Manutenibilidade”?

R: Programação modular denota a construção de programas pela composição de partes pequenas para formar partes maiores. As partes são chamadas módulos o qual sugere que um problema complexo seja dividido em subproblemas que possam ser resolvidos separadamente.
A Manutenibilidade refere-se ao nível que o software é atualizado ou modificado depois da

entrega. Esse conceito é a melhoria da divisão de uma aplicação monolítica dentro de módulos que estão bem definidos. Para isso, deve-se usar dois princípios para ter uma arquitetura madura e sustentável: Coesão e Acoplagem.

Coesão está ligado ao princípio da responsabilidade única, ou seja, uma classe não deve assumir responsabilidades que não são suas. Já acoplagem significa o quanto uma classe depende da outra para funcionar. Em uma aplicação modular, modificar um módulo é mais fácil quando há menos dependências ligadas a ele.

12) _ Para servem os objetos:

12.a)_ this;

R: Apontar para um parametro da classe.

12.b)_ super.

R: Serve para referênciar um método da classe pai.

13) _ Usando Java, dê um exemplo que contemple as respostas das questões 12.a e 12.b.

R: This:

```
public void setNome(String nome) {
      this.nome = nome;
   }
```

Super

```
public class NumException extends Exception{
   public NumException(String s) {
      super(s);
   }
}
```

14)_ Dentre os conceitos de sustenta a Orientação a Objetos, explique:

14.a) _ Encapsulamento:

14.a.i) _Seus níveis (explique cada um dos três níveis);

R: O encapsulamento permite a possibilidade de se ocultar detalhes da implementação de uma classe.

Seus níveis são:

PÚBLICO (public): todos têm acesso. Um atributo pode ter seu valor alterado a partir de qualquer outro código, mesmo sendo este de uma classe qualquer.

PROTEGIDO (protected): em Java tem acesso quem está no mesmo pacote ou classes que

herdem a classe que contenha atributo ou método protegido.

PRIVADO (private): Restrição total fora da classe. Só têm acesso membros da própria classe.

14.a.ii) _ Como o Encapsulamento pode nos ajudar na padronização, segurança e "manutenibilidade" no desenvolvimento de sistemas;

R: Comportamento: será utilizado somente aquilo que o programador da classe permitir.

Manutenção: mantendo os mesmos serviços da classe, podemos alterar sua estrutura interna da classe (refinamentos de código, por exemplo) sem que outras classes (códigos) que dependam dela tenham que ser alteradas.

Segurança: Não se consegue corromper o estado de um objeto por distração, pois partes deste objeto está protegido pelo encapsulamento.

14.b) _ Herança:

14.b.i)_ Explique os conceitos que "Generalização" e "Especialização";

R: Generalização e especialização referem-se a estratégias de agrupar indivíduos com atributos comuns, manipulando de diferentes formas as diferenças que existem entre eles.

14.b.ii)_ Como o mecanismo de Herança pode nos ajudar na padronização, segurança e "manutenibilidade" no desenvolvimento de sistemas;

R: Comportamento: será utilizado somente aquilo que a classe pai permitir.

Manutenção: mantendo os mesmos serviços da classe, podemos alterar sua estrutura interna da classe filha sem que a classe pai tenha que ser alteradas.

Segurança: Permitir que classes filhas não sejam herdadas.

14.b.iii)_ Explique o conceito de "Reusabilidade". Como este é aplicado no mecanismo de Herança e, ainda, como esta possibilidade nos ajuda no dinamismo da codificação.

R: A possibilidade de utilizar o que está na classe pai nas classes filhas sem precisar escrever novamente. Isso é feito atráves do instanciamento de um novo objeto tipado na classe pai.

14.c) _ Polimorfismo:

14.c.i)_ Sobrecarga;

R: A sobrecarga consiste em permitir, dentro da mesma classe, mais de um método

com o mesmo nome. Entretanto, eles necessariamente devem possuir argumentos diferentes para funcionar.

14.c.ii)_ Sobrescrita;

R: Métodos com a mesma assinatura em classes distintas mas envolvidas num mecanismo de herança.

14.c.iii)_ Coerção.

R: Atribuição forçosa de tipo a objetos recuperados em tempo de execução. Técnica conhecida também por Casting.

15)_ Construa um programa para exemplificar as respostas das questões 14.a, 14.b e 14.c.

R: Sobrecarga

```
public class TesteA{
	private int total;

	public void incrementa(){
	this.total++;
	}

	public void incrementa(int valor){
	this.total=this.total+valor;
	}
}
```

Sobreescrita

```
public class Pai{
	protected int total;

	public void incrementa(){
	this.total++;
	}
}

public class Filha extends Pai{
	public void incrementa(){
	this.total=total+2;
	}
}
```

```
public class Filha2 extends Pai{
        public void incrementa(){
        this.total=total+3;
        }
}
```

Coersão

```
public class Pai{
        protected int total;

        public void incrementa(){
        this.total++;
        }
}

public class Filha extends Pai{
        public void incrementa(){
        this.total=total+2;
        }
}

public class Teste{
        public static main(){
        Filha f=new Filha();
        Pai p = (Filha)f;
        }
}
```

16)_ Explique o que são trocas de mensagens? Como isso acontece?

R: Um programa orientado a objetos é composto por um conjunto de objetos que interagem através de “trocas de mensagens”. Na prática, essa troca de mensagem traduz-se na aplicação de métodos a objetos.

17) _ O que é um “método construtor”? Qual sua importância? Faça um código que demonstre sua explicação.

R:Método construtor é um método especial chamado durante a instanciação da classe. Ele é importante para inicializar a classe e seus atributos com base em valores externos.

```
public class Pai{
```

```
    protected int total;

    public Pai(int inicial){
    this.total=inicial;
    }
    public void incrementa(){
    this.total++;
    }
}
```

18) _ Explique o que são como e quando utilizamos:

18.a) Classe *abstrata*;

R: Embora seja possível declarar um objeto de seu tipo, não é possível instanciar este objeto. Quase sempre é usada com classe como classe base (classe mãe) num mecanismo de herança.

18.b) Método *abstrato*;

R: Contido numa classe abstrata (classe-mãe) e não apresenta “corpo” na classe-mãe, somente sua assinatura. Também, torna obrigatório sua implementação (sobrescrição) na classe-filha.

18.c)_ Classe *final*;

R: Indica que a classe não pode ser herdada. Contrapondo-se ao conceito da classe abstrata que, geralmente, é projetada para ser utilizada como classemãe num mecanismo de herança, uma classe definida como “final” não poderá ser herdada.

18.d)_ Atributo *final*;

R: Indica que o atributo em questão será uma constante, isto é, uma estrutura de dados (assim como um variável) porém que não poderá ter seu valor modificado.

18.e)_ Método *final*.

R: Indica que o método não pode ser sobrescrito. Contrapondo-se ao conceito da método abstrato que é declarado em uma classe-mãe e deverá ser sobrescrito em uma classe-filha.

19) _ Dentro da tecnologia Java, explique o que é a estrutura de dados “*Interface*”. Quando a utilizamos?

R: Assim como uma classe também trata-se de um tipo de um tipo abstrato de dados. Porém todos os métodos que ela contiver deverão ser construídos nas classes que implementarem esta Interface, logo, em sua forma de uso, assemelha-se aos métodos abstratos. Caso a Interface tenha algum atributo, este será do tipo constante, isto é, não poderá ter seu valor alterado. Se comportarão como constantes (atributos “finais”). É utilizada para suprir a necessidade herança múltipla, já que não é possível implementar esta forma de herança em Java.