1º Avaliação Parcial POOI

Questão 1

Classe Funcionario

```
public class Funcionario {
    private String nome;
    //Construtor
    public Funcionario(String nome) {
        this.nome = nome;
    }
    //Getters e Setters
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
}
```

Classe Vendedor

```
public class Vendedor extends Funcionario{
    private double salario;
    //Construtor
    public Vendedor(String nome, double salario) {
        super(nome);
        this.salario = salario;
    }
    //Getters e setters
    public double getSalario() {
        return salario;
    }
    public void setSalario(double salario) {
        this.salario = salario;
    }
}
```

Classe AppAluno

```
public class AppAluno {
    public static void main (String args[]) {
        Vendedor vendedor1 = new Vendedor("Daniel Volski", 1500);

        System.out.println("Dados do vendedor1");
        System.out.println("Nome: "+ vendedor1.getNome());
        System.out.println("Salario: "+ vendedor1.getSalario());
    }
}
```

Questão 2

No contexto de programação orientada a objetos, o termo encapsulamento trata-se do controle da visibilidade de membros da classe em relação a outras classes. Em outras palavras, é controlar o acesso dos membros de forma a delimitar as classes que poderão ou não ter acesso a determinado membro.

Por exemplo, considere o fragmento da classe Teste1 abaixo:

```
public class Teste1 {
        private double atributo1;
        public double atributo2;
        protected double atributo3;
        double atributo4;
}
```

Dentro desta classe foram declarados quatro atributos do tipo *double* utilizando os respectivos modificadores de acesso *private, double, protected* e *default* (quando nenhum modificador é atribuído, esse o modificador usado por padrão). Com isso é possível realizar a seguinte análise:

- **atributo1**: possui o modificador *private*, o qual delimita que este atributo será visível apenas dentro da própria classe na qual foi criada, não sendo possível ter acesso direto a ela sem o uso de modificadores e acessadores (*getters* e *setters*);
- **atributo2**: possui o modificador de acesso *public*, o qual diz que este atributo será visível de forma direta em qualquer outra classe;
- **atributo3**: possui o modificador de acesso *protected*, qual diz que este atributo pode ser acessado por outras classes e subclasses contidas no mesmo pacote (*package*);
- **atributo4**: por mais que não esteja escrito, o modificador de acesso utilizado neste atributo é o *default* o qual indica que o atributo é visível apenas para classes que estejam no mesmo pacote.

Questão 3

Uma sobrecarga de métodos é quando se tem dois ou mais métodos os quais possuem o mesmo nome, porém diferem na assinatura do parâmetro, fazendo com que o método utilizado seja o método que coincide com os parâmetros passados. Segue um exemplo utilizando as classes Exemplo1 e Exemplo2:

Classe Exemplo1

```
public class Exemplo1 {
        public int soma(int param1, int param2) {
                return param1 + param2;
        }
}
```

Classe Exemplo2

```
public class Exemplo2 extends Exemplo1 {
        public int soma(int param1, int param2, int param3) {
                return param1 + param2 + param3;
        }
}
```

A classe Exemplo1 cria um método chamado soma() o qual recebe dois parâmetros do tipo inteiro e então retorna como resultado a soma de ambos os parâmetros. A classe Exemplo2 herda da classe Exemplo1, e possui também um método soma, porém ao invés de dois este recebe três parâmetros. O método a ser escolhido vai depender de acordo com os parâmetros que serão passados. Considere o seguinte trecho de código:

```
Exemplo2 exemplo = new Exemplo2(); //Cria um objeto do tipo Exemplo2

exemplo.soma(1, 2) //Chama o método soma da classe Exemplo1
```

```
exemplo.soma(1, 2, 3)//Sobrecarrega o método soma do Exemplo1, subsituindo pelo método definido em Exemplo2
```

A sobrescrita de métodos é muito semelhante a sobrecarga, porém na sobrescrita tanto o nome do método, a assinatura e a quantidade de parâmetros são exatamente os mesmos. Considere novamente as classes abaixo:

Classe Exemplo1

```
public class Exemplo1 {
	public void mensagem(){
		System.out.println("Método do Exemplo1");
	}
}
```

Classe Exemplo2

```
public class Exemplo2 extends Exemplo1 {
	@Override
	public void mensagem(){
		System.out.println("Método do Exemplo2");
	}
}
```

Neste caso, o método mensagem() do Exemplo2 substitui o método mensagem() do Exemplo1. A sobrescrita é indicada pelo marcador @Override (não é obrigatório mas é uma boa prática).

Questão 4

```
public class Q4 {
   public static void main(String args[]) {
      System.out.println("Media de kilometros percorridos: " + calcMedia(102, 22, 39));
      System.out.println("Media de kilometros percorridos: " + calcMedia(19, 100, 42));
   }

   public static double calcMedia(double ...numeros) {
      double resultado = 0;
      for (double d: numeros)
         resultado += d;
      return resultado/numeros.length;
   }
}
```

Questão 5

A) Não, apenas os métodos m1(), m2() e m3() tendo em vista que os métodos m4() e m5() pertencem a classe subClasse que herda da classe superClasse.

B) Todos os métodos citados podem ser acessados pela classe subClasse já que os métodos m1(), m2() e m3() são públicos e foram herdados da classe superClasse.

C) A partir do momento que m1() é declarado como private, ele não se torna mais acessível pela classe subClasse mesmo com herança. Logo os método que ainda podem ser acessados são os métodos criados dentro de subClasse e os métodos declarados como públicos em superClasse

D) Ainda apenas os métodos m1(), m2() e m3(), já que os métodos m4() e m5() pertencem a classe subClasse que herda da superClasse.