Revisão

Nome: William Cardoso Barbosa

1. O que é uma classe?

   Uma classe é um gabarito para a definição de objetos. Através da definição de uma classe, descreve-se que propriedades -- ou **atributos** -- o objeto terá.

   Além da especificação de atributos, a definição de uma classe descreve também qual o comportamento de objetos da classe, ou seja, que funcionalidades podem ser aplicadas a objetos da classe. Essas funcionalidades são descritas através de **métodos**. Um método nada mais é que o equivalente a um procedimento ou função, com a restrição que ele manipula apenas suas variáveis locais e os atributos que foram definidos para a classe.

```
public class Pessoa {
  private String nome;
  private int idade;

  public Pessoa(int i, String n ){
    this.nome = n;
    this.idade = i;
  }

  public String falar(){
    return "Olá, meu nome é " + this.nome + " e tenho " + this.idade + " anos.";
  }

  public String getNome(){
    return this.nome;
  }

  public int getIdade(){
    return this.idade;
  }

}
```

2. Quais são os principais tipos primitivos de dados em Java? Dê exemplo de variáveis com cada tipo e exemplifique via código.

```
class Pessoa {
     public   static   void  main(String[] args) {
        int idade =  18 ;
        double peso =  70.5 ;
        char sexo =  'M' ;
        boolean vivo =  true ;
        String nome =  "João" ;
        long numeroGrande =  10000000000000000000L ;
        float numeroDecimal =  3.14F ;
        byte numeroPequeno =  127 ;
    }
  }
```

3. O que é um objeto?

   Um **objeto** é um elemento computacional que representa, no domínio da solução, alguma entidade (abstrata ou concreta) do domínio de interesse do problema sob análise. **Objetos**
   similares são agrupados em classes. No paradigma de orientação a **objetos** tudo pode ser potencialmente representado como um **objeto**.

```
public class Main {
  public static void main(String[] args) {
    Pessoa p = new Pessoa();
    p.setNome("João");
```

```
    p.setIdade(20);
    Syste.out.println(p.falar());
  }
}
```

4. O que é herança?

   Herançaé um princípio de orientação a objetos, que permite que classes compartilhem atributos e métodos, através de "**heranças**". Ela é usada na intenção de reaproveitar código ou comportamento generalizado ou especializar operações ou atributos.

```
class PessoaJuridica extends Pessoa {
    private String cnpj;

    public Pessoa(String nome, int idade, String sexo, String cnpj) {
        super(nome, idade, sexo);
        this.nome = nome;
        this.idade = idade;
        this.sexo = sexo;
        this.cnpj = cnpj;
    }
    public String toString() {
        rreturn super.toString() + "CNPJ: " + cnpj;
    }
}
```

5. O que é encapsulamento?

   Encapsular os dados de uma aplicação significa **evitar que estes sofram acessos indevidos**. Para isso, é criada uma estrutura que contém métodos que podem ser utilizados por qualquer outra classe, sem causar inconsistências no desenvolvimento de um código.

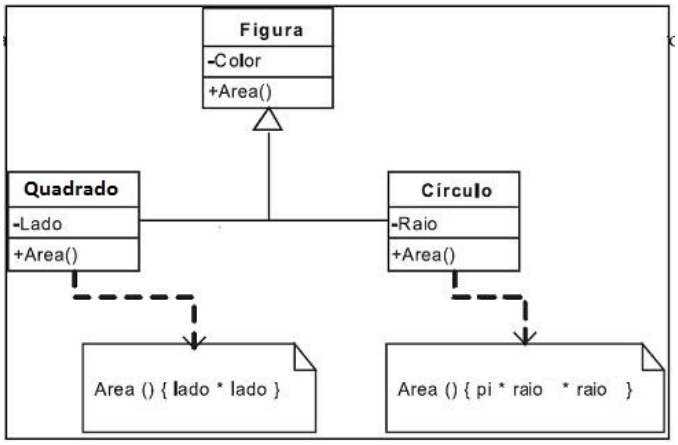
```
public class Exemplo {
  private String texto;
  Exemplo(String texto) {
    this.texto = texto;
  }

  public void setTexto(String texto) {
    this.texto = texto;
  }

  public String getTexto(){
    return this.texto;
  }
}
```

6. O que é polimorfismo?

Em programação orientada a objetos, **polimorfismo** é o princípio pelo qual duas ou mais classes derivadas da mesma superclasse podem invocar métodos que têm a mesma assinatura, mas comportamentos distintos. Comentários: É o principal conceito do **polimorfismo**.




```
public class Animal {
     public void comer() {
          System.out.println( "Animal Comendo..." );
     }
}


public class Cao  extends Animal {
     public void comer() {
          System.out.println( "Cão Comendo..." );
     }
}


public class Tigre extends Animal {
     public void comer() {
          System.out.println( "Tirgre Comendo..." );
     }
}


publci class Test {

  public void fazerAnimalComer( Animal animal ) {
       animal.comer();
  }

  public static void main( String[] args ) {
       Test t = new Test();
        t.fazerAnimalComer( new Animal() );
        t.fazerAnimalComer( new Cao() );
        t.fazerAnimalComer( new Trigre() );
   }
}
```

6. O que é classe abstrata?

   As **classes abstratas**são as que não permitem realizar qualquer tipo de instância. São **classes** feitas especialmente para serem modelos para suas **classes**derivadas. As **classes** derivadas, via de regra, deverão sobrescrever os métodos para realizar a implementação dos mesmos.

   • **Métodos Abstratos** – Os métodos abstratos estão presentes somente em classes abstratas, e são aqueles que **não possuem implementação**. A sintaxe deste tipo de método é a seguinte: abstract.

```
public abstract class Funcionario
     {
         public string Nome;
         public decimal Salario;

         public abstract void Reajustar();
     }
```

8. O que é interface?

   Podemos definir como interface o **contrato** entre a classe e o mundo exterior. Quando uma classe implementa uma interface, se compromete a fornecer o comportamento publicado por esta interface.

   As **classes** ajudam a definir um objeto e seu comportamento e as **interfaces** que auxiliam na definição dessas classes. As interfaces são formadas pela declaração de um ou mais métodos, os quais obrigatoriamente não possuem corpo

```
interface Pessoa {
    public String getNome();
    public String getEndereco();
    public String getTelefone();
    static final String nome = "João";
    static void Apresentar() {
        System.out.println("Olá, meu nome é " + nome);
    }
}
```

9. Pra que servem os construtores?

   os construtores são os responsáveis por criar o objeto em memória, ou seja, instanciar a classe que foi definida. Eles são obrigatórios e são declarados conforme a **Listagem 1.**

   Por padrão, o Java já cria esse construtor sem parâmetros para todas as classes, então você não precisa fazer isso se utilizará apenas construtores sem parâmetros. Por outro lado, se você quiser, poderá criar mais de um construtor para uma mesma classe. Ou seja, posso criar um construtor sem parâmetros, com dois parâmetros e outro com três parâmetros, como vemos no exemplo da **Listagem 3**

   .

```
public class Carro{
  public String nomeCarro;
  /* CONSTRUTOR DA CLASSE Carro */
  public Carro(){
  //Faça o que desejar na construção do objeto
    this.nomeCarro =  "padrão";
  }
  public Carro(String nome) {
    this.nomeCarro = nome;
  }
}

O construtor sempre tem a seguinte assinatura:

modificadores de acesso (public nesse caso) + nome da classe (Carro nesse caso) + parâmetros (nenhum definido
neste caso). O construtor pode ter níveis como: public, private ou protected.
```

10. O que são Modificadors de acesso?

    Os modificadores de acesso são palavras-chave na linguagem Java. Eles servem para definir a visibilidade que determinada classe ou membro terá diante das outras. Visibilidade neste caso tem o mesmo significado que acesso, pois se não está visível não pode ser acessado. Para entender como o controle de acesso é feito, primeiramente devemos estudar dois conceitos: níveis de acesso e modificadores de acesso. Níveis de acesso são conhecidos por public, private, protected e default. E os modificadores são apenas três: public, private, protected. O nível de acesso default (padrão) não exige modificador. Quando não se declara nenhum modificador, o nível default é implícito.

```
/*
 *
 * exemplo modificadores de acesso
 * public : pode ser acessado de qualquer lugar
 * private : pode ser acessado apenas dentro da classe
 * protected : pode ser acessado apenas dentro da classe e subclasses
```

```
  * default : pode ser acessado apenas dentro do pacote
  *
  */
```

11. Cite as principais diferenças entre classe abstrata e interfaces, o que cada uma pode e não pode ter e suas principais utilidades.

    Basicamente, a interface não permite a inserção de qualquer tipo de código, muito menos se ele for padrão. Já a classe abstrata pode oferecer uma codificação completa, o padrão ou apenas possuir a declaração de um esqueleto para ser sobrescrita posteriormente.

    Uma classe abstrata pode conter métodos completos ou incompletos. Uma Interface pode conter apenas a assinatura de um método, mas nenhum corpo ou implementação. Portanto, essa classe pode implementar métodos, mas em uma Interface, não.

Códigos de exemplo/ pesquisa

```
import java.util.List;

public interface INotasDisciplina {
  float mediaFinalAluno ( float prova1 , float pesop1 , float
  prova2 , float pesop2 , float trabalho1 , float pesot1 ,
  float trabalho2 , float pesot2 , float pesoProvas , float
  pesoTrabalhos );
  float mediaFinalAluno ( List < Float > notasProvas , List <
  Float > pesoProvas , List < Float > notasTrabalhos , List <
  Float > pesoTrabalhos, List < Float > notasAtividades, List < Float > pesoAtividades);
  void quantidadeParaAprovacao ( float notaFinal , float notaMinima );
}
```

```
import java.util.List;

public class NotasDisciplina implements INotasDisciplina {
  public NotasDisciplina() {}
  public float mediaFinalAluno ( float prova1 , float pesop1 , float
  prova2 , float pesop2 , float trabalho1 , float pesot1 ,
  float trabalho2 , float pesot2 , float pesoProvas , float
  pesoTrabalhos ){
    float mediaProvas = ( prova1 * pesop1 + prova2 * pesop2 ) /
    ( pesop1 + pesop2 );
    float mediaTrabalhos = ( trabalho1 * pesot1 + trabalho2 *
    pesot2 ) / ( pesot1 + pesot2 );
    return ( mediaProvas * pesoProvas + mediaTrabalhos *
    pesoTrabalhos ) / ( pesoProvas + pesoTrabalhos );
  }
  public float mediaFinalAluno ( List < Float > notasProvas , List <
  Float > pesoProvas , List < Float > notasTrabalhos , List <
  Float > pesoTrabalhos, List < Float > notasAtividades, List < Float > pesoAtividades){
    float mediaProvas = 0;
    float mediaTrabalhos = 0;
    float pesoTotalProvas = 0;
    float pesoTotalTrabalhos = 0;
    float pesoTotalAtividades = 0;
    for ( int i = 0 ; i < notasProvas.size(); i++ ) {
      mediaProvas += notasProvas.get(i) * pesoProvas.get(i);
      pesoTotalProvas += pesoProvas.get(i);
    }
    for ( int i = 0 ; i < notasTrabalhos.size(); i++ ) {
      mediaTrabalhos += notasTrabalhos.get(i) * pesoTrabalhos.get(i);
      pesoTotalTrabalhos += pesoTrabalhos.get(i);
    }
    for ( int i = 0 ; i < notasAtividades.size(); i++ ) {
      pesoTotalAtividades += pesoAtividades.get(i);
    }
    return ( mediaProvas / pesoTotalProvas + mediaTrabalhos / pesoTotalTrabalhos ) / 2;
```

```
  }

  public void quantidadeParaAprovacao ( float notaFinal , float notaMinima ){
    float  notaNecessaria = notaMinima - notaFinal;
    if ( notaNecessaria > 0 ) {
      System.out.println(String.format("Para ser aprovado, você precisa tirar %.2f na prova final.", notaNecessaria));
    } else {
      System.out.println("Você já está aprovado.");
    }
  }
}
```

```
import java.util.List;

class Main {
    public static void main(String[] args) {
      NotasDisciplina notas = new NotasDisciplina();
      List<Float> notasProvas = List.of(9.6f);
      List<Float> pesoProvas = List.of(0.5f);
      List<Float> notasTrabalhos = List.of(9.5f);
      List<Float> pesoTrabalhos = List.of(0.3f);
      List<Float> notasAtividades = List.of(10f, 10f , 8f);
      List<Float> pesoAtividades = List.of(0.2f , 0.2f , 0.2f);
      float notaFinal = notas.mediaFinalAluno(notasProvas, pesoProvas, notasTrabalhos, pesoTrabalhos, notasAtividades, pesoAtividades);
      System.out.println(String.format("Nota final: %.2f", notaFinal));
      notas.quantidadeParaAprovacao(notaFinal, 7f);
    }
}
```

```
import java.time.LocalDate;

public interface AnaliseData {
  int calcularIdade(LocalDate dataNascimento, LocalDate date);
  int quantosDiasParaAniversario(LocalDate dataNascimento);
  int quantosDiasDesdoUltimoAniversario(LocalDate dataNascimento);
}
```

```
import java.time.LocalDate;

public class TestarData implements AnaliseData {
  public int calcularIdade(LocalDate dataNascimento, LocalDate date){
    int idade = date.getYear() - dataNascimento.getYear();

    if (date.getMonthValue() < dataNascimento.getMonthValue()) {
      idade--;
    } else if (date.getMonthValue() == dataNascimento.getMonthValue()) {
      if (date.getDayOfMonth() < dataNascimento.getDayOfMonth()) {
        idade--;
      }
    }
    return idade;
  }
  public int quantosDiasParaAniversario(LocalDate dataNascimento){
    int dias = 0;
    LocalDate date = LocalDate.now();
    dias = dataNascimento.getDayOfYear() - date.getDayOfYear();
    if (dias < 0) {
      dias = 365 + dias;
    }
    return dias;
  }
  public int quantosDiasDesdoUltimoAniversario(LocalDate dataNascimento){
    int dias = 0;
```

```
    LocalDate date = LocalDate.now();
    dias = date.getDayOfYear() - dataNascimento.getDayOfYear();
    if (dias < 0) {
      dias = 365 + dias;
    }
    return dias;
  }
}
```

```
import java.time.LocalDate;

public class Main {
  public static void main(String[] args) {
    AnaliseData data = new TestarData();
    LocalDate dataNascimento = LocalDate.of(1954, 11, 24);
    LocalDate date = LocalDate.now();
    System.out.println("Idade: " + data.calcularIdade(dataNascimento, date));
    System.out.println("Dias para aniversário: " + data.quantosDiasParaAniversario(dataNascimento));
    System.out.println("Dias desde o último aniversário: " + data.quantosDiasDesdoUltimoAniversario(dataNascimento));
  }
}
```

```
import java.util.ArrayList;

abstract class RespostasProvaConcurso {
  private String concurso = "ProgramadorTJ";
  private int qtQuestoes = 10;
  private char respostas[];

  abstract float calculaNotaPessoa();

  public RespostasProvaConcurso(int qtQuestoes, char respostas[]) {
    this.setQtQuestoes(qtQuestoes);
    this.setRespostas(respostas);
  }

  public ArrayList<Boolean> verificaAcertos(char questoes[]) {
    ArrayList<Boolean> acertos = new ArrayList<Boolean>();
    for (int i = 0; i < this.qtQuestoes; i++) {
      acertos.add(this.isAcerto(this.getRespostaQuestao(i), questoes[i]));
    }
    return acertos;
  }

  public char getRespostaQuestao(int numeroQuestao) {
    return this.respostas[numeroQuestao];
  }

  private boolean isAcerto(char a, char b) {
    return a == b;
  }

  public String getConcurso() {
    return this.concurso;
  }

  public int getQtQuestoes() {
    return this.qtQuestoes;
  }

  public char[] getRespostas() {
    return this.respostas;
  }

  public void setRespostas(char[] respostas) {
    this.respostas = respostas;
  }

  public void setConcurso(String concurso) {
    this.concurso = concurso;
```

```
  }

  public void setQtQuestoes(int qtQuestoes) {
    this.qtQuestoes = qtQuestoes;
  }
}




import java.util.ArrayList;

public class Prova  extends RespostasProvaConcurso {
  private double notaQuestao = 1;
  private char gabarito[];

  public Prova(int qtQuestoes, char respostas[], char gabarito[]) {
    super(qtQuestoes, respostas);
    this.setGabarito(gabarito);
  }

  public float calculaNotaPessoa() {
    ArrayList<Boolean> acertos = this.verificaAcertos(this.getGabarito());
    int acertosInt = 0;
    for (Boolean acerto : acertos) {
      if (acerto) {
        acertosInt++;
      }
    }
    return (float) (acertosInt * this.notaQuestao);
  }

  public char[] getGabarito() {
    return this.gabarito;
  }

  public void setGabarito(char[] gabarito) {
    this.gabarito = gabarito;
  }

  public double getNotaQuestao() {
    return this.notaQuestao;
  }

  public void setNotaQuestao(double notaQuestao) {
    this.notaQuestao = notaQuestao;
  }


}






public class Main {
  public static void main(String[] args) {

    // valendo 1 ponto
    char respostas[] = { 'A', 'C', 'B', 'E', 'C', 'B', 'D', 'C', 'E', 'D' };
    Prova prova = new Prova(10, new char[] {'A', 'C', 'B', 'E', 'C', 'B', 'D', 'C', 'E', 'D'}, respostas);
    System.out.println(prova.calculaNotaPessoa());

    // valendo 2 pontos
    char respostas2[] = { 'A', 'C', 'B', 'E', 'C' };
    Prova prova2 = new Prova(5, new char[] {'A', 'C', 'B', 'E', 'A'}, respostas2);
    prova2.setNotaQuestao(2);
    System.out.println(prova2.calculaNotaPessoa());



  }
}
```