Segunda Prova de POO – Prof. Bruno Travençolo

Nome:_____

Considere o código fonte do aplicativo que armazena informações sobre bicicletas.

1) Faça um programa principal que armazene um vetor com as informações de **n** bicicletas. Cadastre pelo menos uma bicicleta de cada tipo. Para facilitar, não é necessário pedir para o usuário digitar as informações – os valores podem ser inicializados por meio de valores constantes no código. Faça um laço que percorra o vetor e chame um dos métodos das classes sem a necessidade de operações de *cast.*

2) Considere que a prefeitura queria cadastrar todos os veículos de duas rodas, o que inclui as bicicletas, patinetes e motos. Deverão ser armazenados para cada veículo um número de identificação, o nome do proprietário e uma data de validade. Também deve haver a implementação de um método que mostre todas as informações do veículo. Altere o projeto de forma que esse novo requisito seja atendido. Não é necessário implementar as classes para patinetes e motos.

3) Note que os métodos para acelerar e frear as bicicletas podem gerar valores inválidos. É possível que a velocidade fique negativa ou que atinja um valor muito alto, acima de 120km/h. Não deixe que isso ocorre. Para retornar o erro, crie uma classe de exceção baseada na classe IllegalStateException, que herda RuntimeException. Mostre na classe principal um exemplo de tratamento da exceção.

4) Considere a interface Qualidade. Utilize-a para as classes dos SpeedBike e MountainBike. Para comparar a qualidade entre esses tipos de bicicletas, use:

- SpeedBike: é melhor a bicicleta que tem menor espessura de pneu.

- MountainBike: Suspensão a óleo (String: “Óleo”) é melhor que a suspensão a ar (String: “ar”), que por sua vez é melhor que a suspensão a molas (String: “molas”).

```java
public class Bike {

    protected int marcha;
    protected double velocidade;

    public Bike(double velocidadeInicial, int marchaInicial) {
        marcha = marchaInicial;
        velocidade = velocidadeInicial;
    }

    public void setMarcha(int novoValor) {
        marcha = novoValor;
    }

    public void frear(int decremento) {
        velocidade -= decremento;
    }

    public void acelerar(int incremento) {
        velocidade += incremento;
    }

    public void imprimirDados(){
        System.out.println("\nBike está na marcha " + this.marcha+
                " e com velocidade de " + this.velocidade + " km/h. ");
    }
}
```

```java
public class SpeedBike extends Bike {
    // In millimeters (mm)
    private int espessuraPneu;

    public SpeedBike(double velocidadeInicial,
                     int marchaInicial,
                     int espessuraPneu){
        super(velocidadeInicial,
              marchaInicial);
        this.setEspessuraPneu(espessuraPneu);
    }

    public int getEspessuraPneu(){
        return this.espessuraPneu;
    }

    public void setEspessuraPneu(int espessuraPneu){
        this.espessuraPneu = espessuraPneu;
    }

    public void imprimirDados(){
        super.imprimirDados();
        System.out.println("A SpeedBike tem pneus de "
                + getEspessuraPneu() + " MM.");
    }
}
```

```java
public class MountainBike extends Bike {
    private String tipoSuspensao;

    public MountainBike(
            double velocidadeInicial,
            int marchaInicial,
            String tipoSuspensao){
        super(  velocidadeInicial,
                marchaInicial);
        this.setTipoSuspensao(tipoSuspensao);
    }

    public String getTipoSuspensao(){
        return this.tipoSuspensao;
    }

    public void setTipoSuspensao(String tipoSuspensao) {
        this.tipoSuspensao = tipoSuspensao;
    }

    public void imprimirDados() {
        super.imprimirDados();
        System.out.println("A " + "MountainBike possui"+
                "suspensão do tipo " +
                getTipoSuspensao());
    }
}
```

```java
public interface Qualidade {
     boolean ehMelhor(Qualidade obj);
}

```

CLASS/OBJECT TYPES:

INSTANTIATION:

```
public class Ball {//only 1 public per file
  //STATIC FIELDS/METHODS
  private static int numBalls = 0;
  public static int getNumBalls() {
    return numBalls;
  }
  public static final int BALLRADIUS = 5;

  //INSTANCE FIELDS
  private int x, y, vx, vy;
  public boolean randomPos = false;

  //CONSTRUCTORS
  public Ball(int x, int y, int vx, int vy)
  {
    this.x = x;
    this.y = y;
    this.vx = vx;
    this.vy = vy;
    numBalls++;
  }
  Ball() {
    x = Math.random()*100;
    y = Math.random()*200;
    randomPos = true;
  }
```

```
//Note, multi-dim arrays can have nulls
//in many places, especially object arrays:
Integer[][] x = {{1,2},{3,null},null};
```

FUNCTIONS / METHODS:

Static Declarations:

```
public static int functionname( … )
private static double functionname( … )
static void functionname( … )
```

Instance Declarations:

```
public void functionname( … )
private int functionname( … )
```

Arguments, Return Statement:

```
int myfunc(int arg0, String arg1) {
  return 5; //type matches int myfunc
}
//Non-void methods must return before ending
//Recursive functions should have an if
//statement base-case that returns at once
```

POLYMORPHISM:

Single Inheritance with "extends"

```
class A{ }
class B extends A{ }
abstract class C { }
class D extends C { }
class E extends D
```

Abstract methods

```
abstract class F {
  abstract int bla();
}
class G extends F {
  int bla() {  //required method
   return 5;
  }
}
```

Multiple Inheritance of interfaces with "implements" (fields not inherited)

```
interface H {
  void methodA();
  boolean methodB(int arg);
}
interface I extends H{
  void methodC();
}
interface K {}
class J extends F implements I, K {
  int bla() { return 5; } //required from F
  void methodA(){} //required from H
  boolean methodB(int a) { //req from A
    return 1;
  }
  void methodC(){} //required from I
}
```

Type inference:

```
A x = new B(); //OK
B y = new A(); //Not OK
C z = new C(); //Cannot instantiate abstract
//Method calls care about right hand type
(the instantiated object)
//Compiler checks depend on left hand type
```

throw vs throws

Java throw example

```
void a(){
throw new ArithmeticException("Incorrect");}
```

Java throws example

```
void a()throws ArithmeticException {}
```

Java throw and throws example

```
void a()throws ArithmeticException{
throw new ArithmeticException("Incorrect");
}
```

try
catch
finally
throws
throw

ARRAYS:

```
int[] x = new int[10]; //ten zeros
int[][] x = new int[5][5]; //5 by 5 matrix
int[] x = {1,2,3,4};
x.length; //int expression length of array
int[][] x = {{1,2},{3,4,5}}; //ragged array
String[] y = new String[10]; //10 nulls
//Note that object types are null by default

//loop through array:
for(int i=0;i<arrayname.length;i++) {
  //use arrayname[i];
}

//for-each loop through array
int[] x = {10,20,30,40};
for(int v : x) {
  //v cycles between 10,20,30,40
}

//Loop through ragged arrays:
for(int i=0;i<x.length;i++)
  for(int j=0;j<x[i].length;j++) {
     //CODE HERE
  }

//Note, multi-dim arrays can have nulls
//in many places, especially object arrays:
Integer[][] x = {{1,2},{3,null},null};
```

java.lang.String Methods

```
//Operator +, e.g. "fat"+"cat" -> "fatcat"
boolean equals(String other);
int length();
char charAt(int i);
String substring(int i, int j); //j not incl
boolean contains(String sub);
boolean startsWith(String pre);
boolean endsWith(String post);
int indexOf(String p); //-1 if not found
int indexOf(String p, int i); //start at i
int compareTo(String t);
//"a".compareTo("b") -> -1
String replaceAll(String str, String find);
String[] split(String delim);
```