

Desenvolvimento de aplicações multiplataforma

Contornos de desenvolvimento 2ª avaliação

Fecha: 17 Marzo/2021

Nombre y apellidos:

Dni:

VALORES DE LAS PREGUNTAS

Ejercicio 1 Documentación: 1 pts

Ejercicio 2 Testing JUnit: 2.5 pts

Ejercicio 3 Optimización: 3 pts

Ejercicio 4 Complejidad algorítmica: 0.5 pts

Ejercicio 5 Refactorización: 3 pts

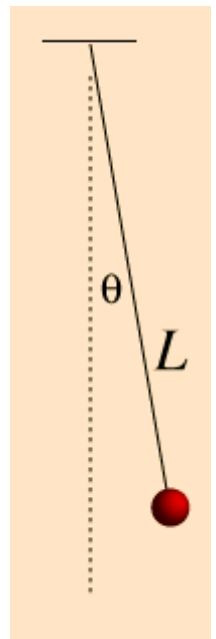
Pendulo simple

Un péndulo es un cuerpo suspendido de un soporte fijo a través de una cuerda o varilla, y que puede balancearse de un lado a otro debido a la influencia de la gravedad.

Las fórmulas matemáticas para describir el movimiento de los péndulos son complicadas. Sin embargo, pueden hacerse algunas suposiciones físicas para simplificar el problema.

El péndulo simple es un caso idealizado de un péndulo real para el cual podemos obtener fácilmente una fórmula que describa su movimiento.

- La longitud de la cuerda vendrá expresada en metros (m)
 - La cuerda que sostiene a la bola es de longitud constante y no tiene masa.
- La fuerza será la aceleración de la gravedad.
 - En la tierra es 9.81 (m/s²), en la luna 1.62 (m/s²), ...
- El movimiento del péndulo está contenido en un plano.
- No existe ningún tipo de rozamiento.



El ejercicio calcula el **periodo** de oscilación (en segundos) y la **frecuencia** en Hercios(Hz) de un péndulo dada su longitud, y la gravedad del planeta en el que te encuentres.

$$T = 2\pi\sqrt{\frac{L}{g}}$$

El ejercicio también calcula la **aceleración angular** del péndulo, en función de la gravedad y su ángulo.

Te muestro a continuación el ejercicio terminado:



```
class Pendulo{
    float longitud;

    public Pendulo(float longitud) {
        super();
        this.longitud = longitud;
    }

    double getPeriodo(float gravedad) throws ArithmeticException{
        if(longitud < 0) throw new ArithmeticException("El pendulo no tiene longitud correcta");
        if(gravedad <= 0) throw new ArithmeticException("El periodo es infinito");
        return 2*Math.PI * Math.sqrt(longitud / gravedad);
    }

    double getFrecuencia(float gravedad) {
        if(longitud < 0) throw new ArithmeticException("El pendulo no tiene longitud correcta");
        if(gravedad <= 0) throw new ArithmeticException("El periodo es infinito");
        return 1/getPeriodo(gravedad);
    }

    double getAceleracionAngular(float gravedad, int anguloEnGrados) {
        if(longitud < 0) return 0;
        double anguloEnRadianes = Math.toRadians(anguloEnGrados);
        return -(gravedad/longitud) * Math.sin(anguloEnRadianes);
    }
}
```

Ejercicio 1: Documentación

Tu tarea será la de documentar correctamente el ejercicio completo del péndulo:

*Puntos del ejercicio:
Documentar completamente el ejercicio 1 pto*

Solucion:

```
**
* Copyright (c) 2021, Angel. All rights reserved.
* Use is subject to license terms.
*/

/**
 * Este ejercicio calcula el periodo de oscilación (en segundos) y la frecuencia en
 * Hercios(Hz) de un <b>péndulo</b> dada su longitud, y la gravedad del planeta en el que te
 * encuentres. También calcula la aceleración angular del péndulo, en función de la gravedad
```



y su ángulo.

```
* @author Angel
* @version 1.0
* @see https://es.wikipedia.org/wiki/P%C3%A9ndulo
*/
```

```
public class Ej1 {
    float longitud;
```

```
/**
```

```
 * Se declara la longitud de la cuerda
```

```
 * @param longitud en metros
```

```
*/
```

```
public Ej1(float longitud) {
    super();
    this.longitud = longitud;
}
```

```
/**
```

```
 * Este método calcula el periodo de oscilación del péndulo
```

```
 * @param gravedad la gravedad del planeta en el que te encuentres
```

```
 * @return el periodo
```

```
 * @throws ArithmeticException lanzamos una excepción cuando el péndulo no tenga
longitud correcta o el periodo sea infinito
```

```
*/
```

```
double getPeriodo(float gravedad) throws ArithmeticException {
```

```
    if (longitud < 0)
```

```
        throw new ArithmeticException("El pendulo no tiene longitud correcta");
```

```
    if (gravedad <= 0)
```

```
        throw new ArithmeticException("El periodo es infinito");
```

```
    return 2 * Math.PI * Math.sqrt(longitud / gravedad);
```

```
}
```

```
/**
```

```
 * Este método calcula la frecuencia del péndulo
```

```
 * @param gravedad la aceleración de la gravedad del planeta en el que estemos
```

```
 * @return frecuencia en Hercios(Hz)
```

```
 * @throws ArithmeticException lanzamos una excepción cuando el péndulo no tenga
longitud correcta o el periodo sea infinito
```

```
*/
```

```
double getFrecuencia(float gravedad) {
```

```
    if (longitud < 0)
```

```
        throw new ArithmeticException("El pendulo no tiene longitud correcta");
```

```
    if (gravedad <= 0)
```

```
        throw new ArithmeticException("El periodo es infinito");
```

```
    return 1 / getPeriodo(gravedad);
```

```
}
```



```
/**
 * Este método calcula la aceleración angular del péndulo
 * @param gravedad gravedad del planeta en el que te encuentres
 * @param anguloEnGrados ángulo del péndulo en grados
 * @return aceleración angular en m/s^2
 */
double getAceleracionAngular(float gravedad, int anguloEnGrados) {
    if (longitud < 0)
        return 0;
    double anguloEnRadianes = Math.toRadians(anguloEnGrados);
    return -(gravedad / longitud) * Math.sin(anguloEnRadianes);
}
```

Ejercicio 2: Testing Unitario con JUnit

Realiza correctamente los test unitarios de los métodos:

- getPeriodo
- getAceleracionAngular

No te limites sólo a testear el Happy Path (casos típicos).

Testea alguna situación excepcional, como por ejemplo:

- Longitud del péndulo nula o negativa (el cual se lanzará una excepción)
- Gravedad nula (el cual se lanzará una excepción)

Puntos del ejercicio:

Testear los happy path 1 pto

Testear situaciones excepcionales 1.5 pto

Solución:

```
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
```

```
public class PenduloTest {
```

```
    Pendulo pendulo;
    float GRAVEDAD_TIERRA = 9.81f;
    float GRAVEDAD_LUNA = 1.62f;
```

```
    @Test
    public void testGetPeriodo() {
```



```
pendulo = new Pendulo(40);
double periodo = pendulo.getPeriodo(GRAVEDAD_LUNA);
Assertions.assertEquals(31.22, periodo, 0.01);
}

@Test
public void testGetPeriodoLongitudInvalida() {
    pendulo = new Pendulo(-10);
    Assertions.assertThrows(ArithmeticException.class, () -> {
        pendulo.getPeriodo(GRAVEDAD_LUNA);
    });
}

@Test
public void testGetPeriodoGravedadInvalida() {
    pendulo = new Pendulo(40);
    Assertions.assertThrows(ArithmeticException.class, () -> {
        pendulo.getPeriodo(-10);
    });
}

@Test
public void testGetAceleracionAngular() {
    pendulo = new Pendulo(40);
    double aceleracion = pendulo.getAceleracionAngular(GRAVEDAD_TIERRA, 30);
    Assertions.assertEquals(-0.122, aceleracion, 0.01);
}

@Test
public void testGetAceleracionAngularLongitudInvalida() {
    pendulo = new Pendulo(-10);
    double aceleracion = pendulo.getAceleracionAngular(GRAVEDAD_TIERRA, 30);
    Assertions.assertEquals(0, aceleracion);
}
}
```

Ejercicio 3: Optimización

Algunos de los métodos de este ejercicio tienen mejoras de optimización. Tu tarea será encontrarlas y solucionarlas.

Además añade un comentario (encima del método) indicando, mediante una breve reseña, que has hecho para solucionar el problema.

*Puntos del ejercicio:
Optimizar los métodos adecuados 3 pto*



```
import java.util.Arrays;

public class Optimiza {

    int numeros[] = {-5, 3,6, 66, 55,2,-7,6,1};

    /* Este método busca si un numero está en una lista de números */
    boolean busca(int numeroBuscado){
        boolean esta = false;
        for(int n: numeros){
            if(numeroBuscado == n){
                esta = true;
            }
        }
        return esta;
    }

    /* Este cuenta cuantos números positivos hay en la lista de numeros */
    int cuentaPositivos() {
        int contador = 0;
        int num = contador;
        for(int n: numeros) {
            if(n>=0) {
                num = contador;
                contador++;
            }
        }
        num=num/numeros.length;
        return contador;
    }

    /* Este método calcula la media de todos los números guardado en la lista de números */
    float calculaMedia() {
        float cont = 0;
        for (double num : numeros) {
            cont += num;
        }
        return cont / numeros.length;
    }

    /* Este método divide cada número de la lista entre la media de todos los numeros */
    float[] dividelosPorLaMedia() {
        float nuevosNumeros[] = new float[numeros.length];
        for(int i=0;i<numeros.length;i++) {
            nuevosNumeros[i] = numeros[i] / calculaMedia();
        }
        return nuevosNumeros;
    }

    /* Este método calcula la mediana de la lista de numeros.
    * Recueda que la mediana representa el valor de la variable de posición central en un conjunto de datos */
    double calculaMediana(){
```



```
int[] copiedArray = numeros.clone();
Arrays.sort(copiedArray);

int mediana;

if (copiedArray.length % 2 == 0) { // Si la longitud es par, se deben promediar los del centro
    mediana = (copiedArray[copiedArray.length / 2 - 1] + copiedArray[copiedArray.length / 2]) / 2;
} else {
    mediana = copiedArray[copiedArray.length / 2];
}
return mediana;
}

int moda(){
    int maximaVecesQueSeRepite=0;
    int moda =0;
    for(int i=0; i<numeros.length; i++){
        int vecesQueSeRepite = 0;
        for(int j=0; j<numeros.length; j++){
            if(numeros[i] == numeros[j]) vecesQueSeRepite++;
        }
        if(vecesQueSeRepite > maximaVecesQueSeRepite){
            moda = numeros[i];
            maximaVecesQueSeRepite = vecesQueSeRepite;
        }
    }

    return moda;
}

public static void main(String[] args) {
    new Optimiza();
}

public Optimiza() {
    System.out.println("Numeros: " + Arrays.toString(numeros));
    System.out.println("Tiene el 5:" + busca(5));
    System.out.println("Tiene el 2:" + busca(2));
    System.out.println("Hay " + cuentaPositivos() + " números positivos");
    System.out.println("La media vale: " + calculaMedia());

    System.out.println("Cada número dividido por la media de todos:
"+Arrays.toString(dividelosPorLaMedia()));
    System.out.println("La mediana vale:" + calculaMediana());
    System.out.println("La moda vale:" + moda());
}
}
```



Solucion:

```
import java.util.Arrays;
```

```
public class Optimiza {
```

```
    int numeros[] = {-5, 3,6, 66, 55,2,-7,6,1};
```

```
    /* Este método busca si un numero está en una lista de números*/
```

```
    boolean busca(int numeroBuscado){
```

```
        for(int n: numeros){
```

```
            if(numeroBuscado == n){
```

```
                return true;
```

```
            }
```

```
        }
```

```
        return false;
```

```
    }
```

```
    /* Este funcion cuantos números positivos hay en la lista de numeros */
```

```
    int cuentaPositivos() {
```

```
        int contador = 0;
```

```
        for(int n: numeros) {
```

```
            if(n>=0) {
```

```
                contador++;
```

```
            }
```

```
        }
```

```
        return contador;
```

```
    }
```

```
    /* Este método calcula la media de todos los números guardado en la lista de números*/
```

```
    float calculaMedia() {
```

```
        float cont = 0;
```

```
        for (int num : numeros) {
```

```
            cont += num;
```

```
        }
```

```
        return cont / numeros.length;
```

```
    }
```

```
    /* Este método divide cada número de la lista entre la media de todos los numeros*/
```

```
    float[] dividelosPorLaMedia() {
```

```
        float nuevosNumeros[] = new float[numeros.length];
```

```
        float media = calculaMedia();
```

```
        for(int i=0;i<numeros.length;i++) {
```

```
            nuevosNumeros[i] = numeros[i] / media;
```

```
        }
```

```
        return nuevosNumeros;
```

```
    }
```

```
    /* Este método calcula la mediana de la lista de numeros.
```

```
    * Recueda que la mediana representa el valor de la variable de posición central en un conjunto de datos*/
```

```
    double calculaMediana(){
```




```
int[] copiedArray = numeros.clone();
Arrays.sort(copiedArray);

int mediana;
int mitad = copiedArray.length >> 1;

if (copiedArray.length % 2 == 0) { // Si la longitud es par, se deben promediar los del centro
    mediana = (copiedArray[mitad - 1] + copiedArray[mitad]) >> 1;
} else {
    mediana = copiedArray[mitad];
}
return mediana;
}

/*
 * Este método calcula el valor que más se repite en la lista
 */
int moda(){
    int maximaVecesQueSeRepite=0;
    int moda =0;
    for(int i=0; i<numeros.length; i++){
        int vecesQueSeRepite = 0;
        for(int j=0; j<numeros.length; j++){
            if(numeros[i] == numeros[j]) vecesQueSeRepite++;
        }
        if(vecesQueSeRepite > maximaVecesQueSeRepite){
            moda = numeros[i];
            maximaVecesQueSeRepite = vecesQueSeRepite;
        }
    }

    return moda;
}

public static void main(String[] args) {
    new Optimiza();
}

public Optimiza() {
    System.out.println("Numeros: " + Arrays.toString(numeros));
    System.out.println("Tiene el 5:" + busca(5));
    System.out.println("Tiene el 2:" + busca(2));
    System.out.println("Hay " + cuentaPositivos() + " números positivos");
    System.out.println("La media vale: " + calculaMedia());

    System.out.println("Cada número dividido por la media de todos:
"+Arrays.toString(dividelosPorLaMedia()));
    System.out.println("La mediana vale:" + calculaMediana());
    System.out.println("La moda vale:" + moda());
}
}
```



Ejercicio 4: Complejidad algorítmica

¿Qué tipo de complejidad algorítmica (usando notación BigO) dirías que tiene la función `moda()` del ejercicio anterior (Ejercicio 3)?

Solución: $O(n^2)$

*Puntos del ejercicio:
Indicar la notación correcta 0.5 pto*

Ejercicio 5: Refactorización

La siguiente clase tiene algunos problemas de diseño. Refactorízala para mejorarla. Yo veo por lo menos 5 refactorizaciones que puedes aplicar.

Acompaña tu código de una **breve reseña (texto)** de cada una de las refactorizaciones que has aplicado.

*Puntos del ejercicio:
Refactorizar el ejercicio correctamente
0.6 ptos por cada refactorización correcta, 3 ptos en total*

```
class Enemigo {
    String tipoArmadura;
    float v=100; // vida del enemigo
    float golpeBase = 40;
    private boolean muerto = false;
    enum TipoEnemigo {NORMAL, BOSS, FINAL_BOSS};
    TipoEnemigo tipoEnemigo;
    int durabilidadArmadura;
    int calidadArmadura;

    float calculapotenciagolpe() {
        switch(tipoEnemigo) {
            case NORMAL: return golpeBase * v;
            case BOSS: return golpeBase * 8 * v;
            case FINAL_BOSS: return golpeBase * 20 * v;
            default: return 0;
        }
    }
}
```

```
}  
  
    boolean estaMuerto(){  
        if(muerto) return true;  
        else return false;  
    }  
  
}
```

Solución:

Cambiamos v por vida y calculapotenciagolpe por calculaPotenciaGolpe
Quitamos enum y switch y usamos herencia
Creamos una clase llamada armadura

```
class Armadura{  
    String tipoArmadura;  
    int durabilidadArmadura;  
    int calidadArmadura;  
  
}  
abstract class Enemigo{  
    float vida=100;  
    float golpeBase = 40;  
    Armadura armadura = new Armadura();  
  
    boolean estaMuerto(){  
        return vida<=0;  
    }  
    abstract float calculapotenciagolpe();  
  
}  
class EnemigoNormal extends Enemigo{  
    float calculaPotenciaGolpe() {  
        return golpeBase * vida;  
    }  
}  
class EnemigoBoss extends Enemigo{  
    float calculaPotenciaGolpe() {  
        return golpeBase * 8 * vida;  
    }  
}  
class EnemigoFinalBoss extends Enemigo{  
    float calculaPotenciaGolpe() {  
        return golpeBase * 20 * vida;  
    }  
}
```