Navegación por el cuestionario

1 2 3 4 5 6 7 8 9 10

Finalizar revisión

Comenzado el	jueves, 7 de marzo de 2019, 16:19
Estado	Finalizado
Finalizado en	jueves, 7 de marzo de 2019, 20:23
Tiempo empleado	4 horas 3 minutos
Calificación	**5,47** de 10,00 (**55**%)

Pregunta **1**
Correcta
Puntúa 1,00 sobre 1,00
Marcar pregunta

Los modificadores de clase `private` y `abstract` son excluyentes. ¿Verdadero o falso?

Seleccione una:
- Verdadero ✓
- Falso

La respuesta correcta es 'Verdadero'
Correcta
Puntos para este envío: 1,00/1,00.

Pregunta **2**
Correcta
Puntúa 0,00 sobre 1,00
Marcar pregunta

Una clase puede adoptar distintos modelos de comportamiento establecidos en diferentes interfaces. Es decir una clase puede implementar varias interfaces. ¿Verdadero o falso?

Seleccione una:
- Verdadero ✓
- Falso

La respuesta correcta es 'Verdadero'
Correcta
Puntos para este envío: 1,00/1,00. Contando con los intentos anteriores, daría **0,00/1,00**.

Pregunta **3**
Correcta
Puntúa 1,00 sobre 1,00
Marcar pregunta

¿Qué modificadores incluyen implícitamente los métodos de una interfaz en Java y por tanto no es necesario indicarlos?

Seleccione una:
- a. `protected` y `final`.
- b. `protected` y `abstract`.
- c. `public` y `final`.
- d. `public` y `abstract`. ✓

La respuesta correcta es: `public` y `abstract`.
Correcta
Puntos para este envío: 1,00/1,00.

Pregunta **4**
Correcta
Puntúa 0,00 sobre 1,00
Marcar pregunta

Los modificadores de clase `public` y `abstract` son excluyentes. ¿Verdadero o falso?

Seleccione una:
- Verdadero
- Falso ✓

La respuesta correcta es 'Falso'
Correcta
Puntos para este envío: 1,00/1,00. Contando con los intentos anteriores, daría **0,00/1,00**.

Pregunta **5**
Correcta
Puntúa 0,90 sobre 1,00
Marcar pregunta

Tenemos la clase Triangulo :

```
public class Triangulo {
    private String id;
    private double base;
    private double altura;

    public Triangulo(String id, double base, double altura) {
        this.id = id;
        this.base = base;
        this.altura = altura;
    }

    public String toString() {
        return "Triangulo(" + id + "): " + base + " x " + altura;
    }

    public double getArea() {
        return base * altura;
    }
}
```

Corríge la siguiente versión para que implemente correctamente la interfaz Comparable y cuando se ejecute el siguiente código el array se ordene según la base de los triangulos, de mayor a menor:

```
Triangulo[] arrayTriangulo = {new Triangulo("A321", 4, 3), new Triangulo("C321", 5, 2), new Triangulo("A123", 2, 5), new Triangulo("B123", 3, 4)};

System.out.println(Arrays.toString(arrayTriangulo));
Arrays.sort(arrayTriangulo);
System.out.println(Arrays.toString(arrayTriangulo));
```

El resultado será:

```
[Triangulo(A321): 4.0 x 3.0, Triangulo(C321): 5.0 x 2.0, Triangulo(A123): 2.0 x 5.0, Triangulo(B123): 3.0 x 4.0]
[Triangulo(C321): 5.0 x 2.0, Triangulo(A321): 4.0 x 3.0, Triangulo(B123): 3.0 x 4.0, Triangulo(A123): 2.0 x 5.0]
```

Respuesta: (penalty regime: 10, 20, ... %)

Reiniciar respuesta

```
1  import java.util.*;
2
3  public class Triangulo implements Comparable<Triangulo> {
4      private String id;
5      private double base;
6      private double altura;
7
8      public Triangulo(String id, double base, double altura) {
9          this.id = id;
10         this.base = base;
11         this.altura = altura;
12     }
```

```
13
14    public String toString() {
15        return "Triangulo(" + id + "): " + base + " x " + altura;
16    }
17
18    public double getArea() {
19        return base * altura;
20    }
```

	Test

✓	`Triangulo[] arrayTriangulo = {new Triangulo("A321", 4, 3), new Triangulo("C321", 5, 2), new Triangulo("A123", 2, 5), new Triangulo("B123", 3, 4`
	`System.out.println(java.util.Arrays.toString(arrayTriangulo));`
	`java.util.Arrays.sort(arrayTriangulo);`
	`System.out.println(java.util.Arrays.toString(arrayTriangulo));`

✓	`Triangulo[] arrayTriangulo = {new Triangulo("A321", 2.5, 3), new Triangulo("C321", 2.8, 2), new Triangulo("A123", 3, 5), new Triangulo("B123",`
	`System.out.println(java.util.Arrays.toString(arrayTriangulo));`
	`java.util.Arrays.sort(arrayTriangulo);`
	`System.out.println(java.util.Arrays.toString(arrayTriangulo));`

Todas las pruebas superadas. ✓

Question author's solution:

```
public class Triangulo implements Comparable<Triangulo> {
    private String id;
    private double base;
    private double altura;

    public Triangulo(String id, double base, double altura) {
        this.id = id;
        this.base = base;
        this.altura = altura;
    }

    public String toString() {
        return "Triangulo(" + id + "): " + base + " x " + altura;
    }

    public double getArea() {
        return base * altura;
    }

    public int compareTo(Triangulo unTriangulo) {
        if (base > unTriangulo.base) {
            return -1;
        }
        if (base < unTriangulo.base) {
            return 1;
        }
        return 0;
    }
}
```

Correcta

Puntos para este envío: 1,00/1,00. Contando con los intentos anteriores, daría **0,90/1,00**.

Pregunta **6**

Correcta

Puntúa 0,67 sobre 1,00

Marcar pregunta

Indica qué necesitarías en cada uno de los siguientes casos:

Quiero poder ordenar arrays y colecciones de objetos de diferentes maneras.	Crear tantas clases como maneras de ordenar. Cada una de ellas implementará la interfaz Comparator de manera diferente. ✓
Quiero poder ordenar arrays y colecciones de objetos de una clase que no puedo modificar. No tiene implementada la interfaz Comparable.	Implementar la interfaz Comparator en una clase nueva y utilizarla on el método sort. ✓
Quiero poder ordenar arrays y colecciones de objetos de una clase diseñada por mí. Siempre los ordenaré por el identificador de la A la Z.	Implementar la interfaz Comparable en esa clase. ✓

Respuesta correcta

La respuesta correcta es: Quiero poder ordenar arrays y colecciones de objetos de diferentes maneras. – Crear tantas clases como maneras de ordenar. Cada una de ellas implementará la interfaz Comparator de manera diferente., Quiero poder ordenar arrays y colecciones de objetos de una clase que no puedo modificar. No tiene implementada la interfaz Comparable. – Implementar la interfaz Comparator en una clase nueva y utilizarla on el método sort., Quiero poder ordenar arrays y colecciones de objetos de una clase diseñada por mí. Siempre los ordenaré por el identificador de la A la Z. – Implementar la interfaz Comparable en esa clase.

Correcta

Puntos para este envío: 1,00/1,00. Contando con los intentos anteriores, daría **0,67/1,00**.

Pregunta **7**

Correcta

Puntúa 1,00 sobre 1,00

Marcar pregunta

Tenemos la clase Triangulo :

```
public class Triangulo {
    private String id;
    private double base;
    private double altura;

    public Triangulo(String id, double base, double altura) {
        this.id = id;
        this.base = base;
        this.altura = altura;
    }

    public String toString() {
        return "Triangulo(" + id + "): " + base + " x " + altura;
    }

    public double getArea() {
        return base * altura;
    }
}
```

Corríge la siguiente versión para que implemente correctamente la interfaz Comparable y cuando se ejecute el siguiente código el array se ordene según el área de los tríangulos, de mayor a menor:

```
Triangulo[] arrayTriangulo = {new Triangulo("A321", 4, 3), new Triangulo("C321", 5, 2), new Triangulo("A123", 2, 5), new Triangulo("B123", 3, 4)};

System.out.println(Arrays.toString(arrayTriangulo));
Arrays.sort(arrayTriangulo);
System.out.println(Arrays.toString(arrayTriangulo));
```

El resultado será:

```
[Triangulo(A321): 4.0 x 3.0, Triangulo(C321): 5.0 x 2.0, Triangulo(A123): 2.0 x 5.0, Triangulo(B123): 3.0 x 4.0]
[Triangulo(A321): 4.0 x 3.0, Triangulo(B123): 3.0 x 4.0, Triangulo(C321): 5.0 x 2.0, Triangulo(A123): 2.0 x 5.0]
```

Respuesta: (penalty regime: 10, 20, ... %)

Reiniciar respuesta

```
1  public class Triangulo implements Comparable<Triangulo> {
2      private String id;
3      private double base;
4      private double altura;
5
6      public Triangulo(String id, double base, double altura) {
```

```
        this.id = id;
        this.base = base;
        this.altura = altura;
    }

    public String toString() {
        return "Triangulo(" + id + "): " + base + " x " + altura;
    }

    public double getArea() {
        return base * altura;
    }

    @Override
```

Test
Triangulo[] arrayTriangulo = {new Triangulo("A321", 4, 3), new Triangulo("C321", 5, 2), new Triangulo("A123", 2, 5), new Triangulo("B123", 3, 4 System.out.println(java.util.Arrays.toString(arrayTriangulo)); java.util.Arrays.sort(arrayTriangulo); System.out.println(java.util.Arrays.toString(arrayTriangulo));
Triangulo[] arrayTriangulo = {new Triangulo("A321", 2.5, 3.5), new Triangulo("C321", 5.4, 2.6), new Triangulo("A123", 3, 5), new Triangulo("B12 System.out.println(java.util.Arrays.toString(arrayTriangulo)); java.util.Arrays.sort(arrayTriangulo); System.out.println(java.util.Arrays.toString(arrayTriangulo));

Todas las pruebas superadas.

Question author's solution:

```
public class Triangulo implements Comparable<Triangulo> {
    private String id;
    private double base;
    private double altura;

    public Triangulo(String id, double base, double altura) {
        this.id = id;
        this.base = base;
        this.altura = altura;
    }

    public String toString() {
        return "Triangulo(" + id + "): " + base + " x " + altura;
    }

    public double getArea() {
        return base * altura;
    }

    public int compareTo(Triangulo unTriangulo) {
        double area1 = getArea();
        double area2 = unTriangulo.getArea();

        if (area1 < area2) {
            return 1;
        }
        if (area1 > area2) {
            return -1;
        }
        return 0;
    }
}
```

Correcta

Puntos para este envío: 1,00/1,00.

Pregunta **8**

Correcta

Puntúa 0,90 sobre 1,00

Marcar pregunta

Tenemos la clase Triangulo :

```
public class Triangulo {
    private String id;
    private double base;
    private double altura;

    public Triangulo(String id, double base, double altura) {
        this.id = id;
        this.base = base;
        this.altura = altura;
    }

    public String toString() {
        return "Triangulo(" + id + "): " + base + " x " + altura;
    }

    public double getArea() {
        return base * altura;
    }

    public double getBase() {
        return base;
    }
}
```

Crea la clase ComparadorArea para que implemente correctamente la interfaz Comparator y cuando se ejecute el siguiente código el array se ordene según el área de los tríangulos, de menor a mayor y si el área es igual en función de la base también de menor a mayor:

```
Triangulo[] arrayTriangulo = {new Triangulo("A321", 4, 3), new Triangulo("C321", 5, 2), new Triangulo("A123", 2, 5), new Triangulo("B123", 3, 4)};

System.out.println(Arrays.toString(arrayTriangulo));
Arrays.sort(arrayTriangulo, new ComparadorArea());
System.out.println(Arrays.toString(arrayTriangulo));
```

El resultado será:

```
[Triangulo(A321): 4.0 x 3.0, Triangulo(C321): 5.0 x 2.0, Triangulo(A123): 2.0 x 5.0, Triangulo(B123): 3.0 x 4.0]
[Triangulo(A123): 2.0 x 5.0, Triangulo(C321): 5.0 x 2.0, Triangulo(B123): 3.0 x 4.0, Triangulo(A321): 4.0 x 3.0]
```

Respuesta: (penalty regime: 10, 20, ... %)

```
import java.util.*;
public class ComparadorArea implements Comparator<Triangulo> {

    @Override
    public int compare(Triangulo unTriangulo, Triangulo otroTriangulo) {
        int devuelve = 0;
        if(unTriangulo.getArea() < otroTriangulo.getArea()){
            devuelve = -1;
        }else if(unTriangulo.getArea() > otroTriangulo.getArea()) {
            devuelve = 1;
        } else if(unTriangulo.getArea() == otroTriangulo.getArea())  {
            if(unTriangulo.getBase() < otroTriangulo.getBase()) {
                devuelve = -1;
            }else if(unTriangulo.getBase() > otroTriangulo.getBase()) {
                devuelve = 1;
            }else{
                devuelve = 0;
            }
        }
        return devuelve;
```

Test
Triangulo[] arrayTriangulo = {new Triangulo("A321", 4, 3), new Triangulo("C321", 5, 2), new Triangulo("A123", 2, 5), new Triangulo("B123", 3, 4

	Test
	System.out.println(java.util.Arrays.toString(arrayTriangulo)); java.util.Arrays.sort(arrayTriangulo, new ComparadorArea()); System.out.println(java.util.Arrays.toString(arrayTriangulo));
✓	Triangulo[] arrayTriangulo = {new Triangulo("A321", 2, 3), new Triangulo("C321", 5, 2.2), new Triangulo("A123", 2.2, 5), new Triangulo("B123", System.out.println(java.util.Arrays.toString(arrayTriangulo)); java.util.Arrays.sort(arrayTriangulo, new ComparadorArea()); System.out.println(java.util.Arrays.toString(arrayTriangulo));

Todas las pruebas superadas. ✓

Question author's solution:

```
import java.util.*;

public class ComparadorArea implements Comparator<Triangulo> {

    public int compare(Triangulo triangulo1, Triangulo triangulo2) {
        double area1 = triangulo1.getArea();
        double area2 = triangulo2.getArea();
        if (area1 == area2) {
            double base1 = triangulo1.getBase();
            double base2 = triangulo2.getBase();
            if (base1 < base2) {
                return -1;
            }
            if (base1 < base2) {
                return 1;
            }
            return 0;
        }
        if (area1 < area2) {
            return -1;
        }
        return 1;
    }
}
```

Correcta

Puntos para este envío: 1,00/1,00. Contando con los intentos anteriores, daría **0,90/1,00**.

Pregunta **9**

Sin contestar

Puntúa 0,00 sobre 1,00

Marcar pregunta

Tenemos la clase abstracta Empleado :

```
public abstract class Empleado {
    private String nombre;
    private String numSeguridadSocial;

    public Empleado(String nombre, String numero) {
        this.nombre = nombre;
        numSeguridadSocial = numero;
    }

    abstract double salario();

    public String toString() {
        return nombre + " (" + numSeguridadSocial + ")";
    }
}
```

Crea la clase EmpleadoHorasa partir de la clase Empleado para que cumpla las siguientes características:

```
1. Tendrá 2 atributos más. El primero será de tipo doble y se denomina pagaHora el segundo será un int y se denominará horas. pagaHora se deberá i
nicializar en el constructor a partir del valore indicado como parámetro. horas se inicializará a 0 también en el constructor.
2. El método salario devolverá el resultado de la siguiente operación: horas * pagaHoras siempre que las horas trabajadas sean menor o igual que 4
0. Si se trabja más de 40 horas, las horas extra se pagarán a 1.5 * pagaHora. En estos casos el resultado será: 40 * pagaHoras + (horas - 40) * pa
gaHoras * 1.5
3. El método toString devolverá el siguiente texto: nombre (numSeguridadSocial) Paga/hora: pagaHora Horas: horas. La paga por hora se mostrará con
 un decimales.
4. Se añadirá el método addHoras al que se le pasará un entero con el número de horas trabajadas y se las añadirá al atributo horas.
```

Si ejecutamos el siguiente código :

```
EmpleadoHoras iker = new EmpleadoHoras("Iker Zamora", "111111JJ", 25);
iker.addHoras(100);
System.out.println(iker);
System.out.println(iker.salario());
```

El resultado será:

```
Iker Zamora (111111JJ) Paga/hora: 25,0 Horas: 50
1375.0
```

Recuerda interactuar con el código de la superclase mediante la palabra reservada super.

Respuesta: (penalty regime: 10, 20, ... %)

```
1
```

Question author's solution:

```
public class EmpleadoHoras extends Empleado {

 private double pagaHora;
   private int horas;

 public EmpleadoHoras(String nombre, String numero, double pagaHora) {
   super(nombre, numero);
     this.pagaHora = pagaHora;
     horas = 0;
       }

   public void addHoras(int horas) {
     this.horas += horas;
   }

 public double salario() {

     if (horas < 40) {
       return pagaHora * horas;
     }
     return pagaHora * 40 + pagaHora * (horas - 40) * 1.5;
       }
```

```
 public String toString(){
  return super.toString() + String.format(" Paga/hora: %.1f Horas: %d", pagaHora, horas);
       }
}
```

Pregunta **10**

Incorrecta

Puntúa 0,00 sobre 1,00

Marcar pregunta

Tenemos la interfaz FormaCalculable :

```
public interface FormaCalculable {

   public static final double PI = 3.1416;

   public double calcularArea();
   public double calcularPerimetro();
}
```

Modifica la clase Rombo para que implemente correctamente la interfaz FormaCalculable:

```
public class Rombo {
   private double diagonal1;
   private double diagonal2;

   public Rombo(double diagonal1, double diagonal2) {
      this.diagonal1 = diagonal1;
      this.diagonal2 = diagonal2;
   }
}
```

sabiendo que:

```
área = diagonal1 * diagonal2 / 2
perímetro = 2 * raizCuadrada(diagonal1
```

2

```
 + diagonal2
```

2

```
)
```

y cuando se ejecute el siguiente código :

```
Rombo unRombo = new Rombo(10, 5);
System.out.printf("ROMBO: Area = %.2f, Perímetro = %.2f\n", unRombo.calcularArea(), unRombo.calcularPerimetro());
```

El resultado será:

```
ROMBO: Area = 25,00, Perímetro = 22,36
```

Respuesta: (penalty regime: 10, 20, ... %)

Reiniciar respuesta

```
1  // Clase Rombo
2
3  public class Rombo {
4
5     // Atributos
6     private double diagonal1;
7     private double diagonal2;
8
9     // Constructor
10    public Rombo(double diagonal1, double diagonal2) {
11       this.diagonal1 = diagonal1;
12       this.diagonal2 = diagonal2;
13    }
14 }
```

Syntax Error(s)

```
__Tester__.java:25: error: cannot find symbol
System.out.printf("ROMBO: Area = %.2f, Perimetro = %.2f\n", unRombo.calcularArea(), unRombo.calcularPerimetro()); ;
                                                                ^
  symbol:   method calcularArea()
  location: variable unRombo of type Rombo
__Tester__.java:25: error: cannot find symbol
System.out.printf("ROMBO: Area = %.2f, Perimetro = %.2f\n", unRombo.calcularArea(), unRombo.calcularPerimetro()); ;
                                                                                      ^
  symbol:   method calcularPerimetro()
  location: variable unRombo of type Rombo
2 errors
```

Question author's solution:

```
// Clase Rombo

public class Rombo implements FormaCalculable {

   // Atributos
   private double diagonal1;
   private double diagonal2;

   // Constructor
   public Rombo(double diagonal1, double diagonal2) {
      this.diagonal1 = diagonal1;
      this.diagonal2 = diagonal2;
   }

   // Devuelve el area de un rombo usando la formula de Heron
   @Override
   public double calcularArea() {
      return diagonal1 * diagonal2 / 2;
   }

   // Devuelve el perimetro de un rombo
   @Override
   public double calcularPerimetro() {
      double lado = Math.sqrt(Math.pow(diagonal1, 2) + Math.pow(diagonal2, 2));
      return 2 * lado;
   }
}
```

Incorrecta

Puntos para este envío: 0,00/1,00.

Finalizar revisión