POLITÉCNICA

Escuela Técnica Superior de Ingeniería de Sistemas Informáticos
Universidad Politécnica de Madrid

Examen de Programación Orientada a Objetos (Plan 2014) — **19 de junio de 2019**

Nº matrícula:_____ **Grupo:**_____ **Nombre:** _____

Apellidos: _____

EJERCICIO 1. (4.5 puntos)

Marcar la respuesta correcta en cada pregunta. Cada pregunta acertada suma 0.5 puntos. Cada pregunta errónea resta 0.2 puntos.

PREGUNTA 1

a) El polimorfismo requiere de la existencia de interfaces en el diseño polimórfico
b) El polimorfismo requiere de la existencia de herencia en el diseño polimórfico
c) El polimorfismo requiere de la existencia de clases genéricas en el diseño polimórfico
d) Ninguna de las anteriores

PREGUNTA 2

a) Los métodos polimórficos deben tener la misma firma
b) Los métodos polimórficos deben tener la misma firma y el mismo valor de retorno
c) Los métodos polimórficos deben tener la misma firma, el mismo valor de retorno y el mismo modificador de acceso (`public`, `private`…)
d) Ninguna de las anteriores

PREGUNTA 3

a) Los genéricos permiten que una misma clase admita diferentes tipos primitivos de datos
b) Los genéricos permiten que una misma clase admita diferentes tipos primitivos o no primitivos de datos
c) Los genéricos permiten que una misma clase admita diferentes tipos de datos no primitivos
d) Ninguna de las anteriores

PREGUNTA 4

a) Los métodos sobrecargados deben tener el mismo identificador
b) Los métodos sobrecargados deben tener el mismo identificador y el mismo número de parámetros
c) Los métodos sobrecargados deben tener el mismo identificador y distinto número de parámetros
d) Ninguna de las anteriores

PREGUNTA 5

a) La redefinición requiere de la existencia de polimorfismo
b) La redefinición requiere de la existencia de herencia
c) La redefinición requiere de la existencia de sobrecarga
d) Ninguna de las anteriores

PREGUNTA 6

a) La clase `LinkedList` pertenece al paquete `java.lang`
b) La clase `LinkedList` no admite genéricos
c) La clase `LinkedList` contiene el método `insert`, para añadir elementos
d) Ninguna de las anteriores

PREGUNTA 7

a) Para recorrer una instancia de `LinkedList` se utilizan `[]` (notación matricial)
b) Para recorrer una instancia de `LinkedList` se utiliza su método `next`
c) Para recorrer una instancia de `LinkedList` se utiliza la clase `Iterator`
d) Ninguna de las anteriores

PREGUNTA 8

a) En POO, prácticamente siempre los métodos son `private`
b) En POO, prácticamente siempre los atributos son `public`
c) En POO, prácticamente siempre las constantes son `private`
d) Ninguna de las anteriores

PREGUNTA 9

a) Reutilización, acoplamiento, y cohesión son objetivos en POO
b) Visualización, acoplamiento, y cohesión son objetivos en POO
c) Reutilización, visualización, y cohesión son objetivos en POO
d) Reutilización, cohesión y visualización son objetivos en POO

Plantilla de respuestas

Pregunta	1	2	3	4	5	6	7	8	9
Respuesta (`a, b, c` o `d`)	b	b	c	a	b	d	c	d	a

Examen de Programación Orientada a Objetos (Plan 2014) — **19 de junio de 2019**

Nº matrícula:_____ **Grupo:**_____ **Nombre:** _____

Apellidos: _____

EJERCICIO 2 (5.5 puntos)

Una empresa de depósitos industriales utilizados para almacenar líquidos nos solicita un software de control. El diseñador software de nuestra empresa nos pide que implementemos los siguientes objetos:

Apartado A (1 punto)

Interfaz *IDepositoBasico*.	
Método	Descripción
meter	Añade una cantidad de litros al depósito y devuelve la cantidad de litros que contiene el depósito tras la operación de carga. Si los litros que se pretenden introducir no caben en el depósito, se levanta la excepción *ExDepositoLleno* y no se realiza la operación
sacar	Extrae una cantidad de litros del depósito y devuelve la cantidad de litros que contiene el depósito tras la operación de descarga. Si los litros que se pretenden extraer son más que los existentes en el depósito, se levanta la excepción *ExDepositoLleno* y no se realiza la operación
getContenido	Indica el número de litros que almacena el depósito
getCapacidad	Indica la capacidad en litros del depósito

Apartado B (1.5 puntos)

Interfaz *IDepositoPeso*.	
Método	Descripción
setDensidad	Indica la densidad del líquido que contiene el depósito. Este dato se usa para hallar el peso del líquido almacenado. Solo se admiten valores de densidad mayores a cero y menores o iguales a dos; en cualquier otro caso se levanta la excepción *ExDepositoDensidad* y no se realiza la operación
getDensidad	Indica la densidad del líquido que almacena o almacenará el depósito
getPeso	Devuelve el peso del líquido contenido en el depósito. Supondremos que se calcula multiplicando el número de litros por el valor de la densidad

Apartado C (1 puntos)

Clase *DepositoBasico*.
Implementa el interfaz *IDepositoBasico*, contiene el método *toString* y proporciona dos constructores: uno al que se le pasa la capacidad del depósito y otro al que se le pasa la capacidad y el contenido inicial en el depósito

Apartado D (2 puntos)

Clase *DepositoPeso*.
Implementa el interfaz *IDepositoPeso*, contiene el método *toString* y proporciona dos constructores: uno al que se le pasa la capacidad del depósito y la densidad del líquido. Al otro constructor se le pasa la capacidad, el contenido inicial en el depósito y la densidad del líquido.

Apartado A

```
public interface IDepositoBasico {
     float meter(int cantidad) throws ExDepositoLleno;
     float sacar(int cantidad) throws ExDepositoVacio;
     float getContenido();
     float getCapacidad();
}
```

Apartado B

```
 public interface IDepositoPeso extends IDepositoBasico {
     void setDensidad(float densidad) throws ExDepositoDensidad;
     float getDensidad();
     float getPeso();
}
```

Apartado C

```
public class DepositoBasico implements IDepositoBasico {
  private float capacidad;
  private float contenido;

  public DepositoBasico(float capacidad) {
         this.capacidad = capacidad;
         this.contenido = 0f;
  }

  public DepositoBasico(float capacidad, float contenido) {
         this.capacidad = capacidad;
         this.contenido = contenido;
  }

  public float meter(int cantidad) throws ExDepositoLleno {
         if (cantidad+contenido>capacidad) throw new ExDepositoLleno();
         else {
                contenido = contenido + cantidad;
            return contenido;
         }
  }

  public float sacar(int cantidad) throws ExDepositoVacio {
         if (contenido<cantidad) throw new ExDepositoVacio();
         else {
                contenido = contenido - cantidad;
            return cantidad;
        }
  }

  public float getContenido() {
         return contenido;
  }

  public float getCapacidad() {
         return capacidad;
  }

  public String toString() {
         return "Capacidad: " + capacidad + ", Contenido: " + contenido;
  }
}
```

Apartado D

```
public class DepositoPeso extends DepositoBasico implements IDepositoPeso {

  private float densidad;

  public DepositoPeso(float capacidad, float densidad)
                                        throws ExDepositoDensidad {
        super(capacidad);
        setDensidad(densidad);
  }

  public DepositoPeso(float capacidad, float densidad) {
        super(capacidad);
        this.densidad = densidad;
  }

  public DepositoPeso(float capacidad, float contenido, float densidad) {
        super(capacidad, contenido);
        this.densidad = densidad;
  }

  public void setDensidad(float densidad) throws ExDepositoDensidad {
        if (densidad<=0 || densidad>2f) throw new ExDepositoDensidad();
        else
              this.densidad = densidad;
  }

  public float getDensidad() {
        return densidad;
  }

  public float getPeso() {
        return getContenido() * densidad;
  }

  public String toString() {
        return super.toString() + ", Densidad: " + densidad;
  }

}
```