Examen de Programación Orientada a Objetos (Plan 2014)

19 de diciembre de 2020

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

Apellidos: _____

Parte 1. TEST (4 puntos)

Cada pregunta puntúa con un punto, las respuestas erróneas tienen una penalización de 0.25 puntos.

Pregunta 1. Marcar la respuesta correcta. En las excepciones de Java:

a) En la cabecera de los métodos se puede indicar que es posible que se lance (o levante) una excepción utilizando la palabra `throws` y el tipo de excepción; en el interior de los métodos se puede lanzar una excepción utilizando las palabras `throw new` y el constructor de la excepción.
b) En el interior de todos los métodos se debe poner un bloque try/catch para lanzar (o levantar) una excepción, a condición de que al menos uno de los catch recoja esa excepción.
c) Un bloque try puede completarse sin necesidad de poner ningún catch, siempre que en la cabecera del método que lo contiene no se indique la posibilidad de que se puede levantar una excepción.
d) Cada bloque try debe finalizar con al menos un bloque catch en el que se indique, con la palabra throws, la excepción que recoge y definiendo entre paréntesis el constructor de la excepción.

Pregunta 2. Marcar la respuesta correcta. En las excepciones de Java:

a) Cuando existe más de un `catch` en el mismo bloque `try` no existe preferencia u orden en su evaluación
b) Cada `catch` de un bloque `try` admite uno o varios parámetros de tipo `Exception` o derivado de `Exception`
c) En el bloque de ejecución asociado a cada `catch` se puede introducir otro u otros bloques `try`
d) Ninguna de las anteriores.

Pregunta 3. Tenemos el interfaz:

```
public interface IAgenda <E extends IPersona>
```

Y la clase `Persona` que implementa el interfaz `IPersona`. Indicar cuál de las siguientes definiciones de clase es correcta:

a) `public class AgendaHash<E>  implements IAgenda<E>`
b) `public class AgendaHash<E>  extends IAgenda<E extends Persona>`
c) `public class AgendaHash<E extends Persona>  implements IAgenda<E extends Persona>`
d) `public class AgendaHash<E extends Persona>  implements IAgenda<E>`

Pregunta 4. Disponemos de una clase genérica `Pila` que admite instancias de tipo `Examen`, y que contiene el constructor vacío. Indicar cuál de las siguientes instanciaciones es correcta:

a) `Pila<E extends Examen> examenes = new Pila<Examen>;`
b) `Pila<E> examenes = new Pila <Examen>();`
c) `Pila<Examen> examenes = new Pila <Examen>();`
d) `Pila<E extends Examen> examenes = new Pila<Examen>();`

Pregunta 5. En el paquete `java.util`:

a) `ArrayList` y `Vector` son clases que implementan el interfaz `HashMap`, mientras que `TreeSet` es una clase que implementa el interfaz `Set`
b) `ArrayList` y `Vector` son clases que implementan el interfaz `Set`, mientras que `TreeSet` es una clase que implementa el interfaz `HashMap`
c) `ArrayList` y `Vector` son clases que implementan el interfaz `List`, mientras que `HashMap` es una clase que implementa el interfaz `TreeSet`
d) Ninguna de las anteriores

Pregunta 6. Dado el método `public void setPersona(String nombre, String dni)`, indicar cuál de los siguientes métodos NO lo sobrecarga:

a) `public void setPersona(String dni, String email)`
b) `public void setPersona(String dni, int edad)`
c) `public void setPersona(String dni)`
d) `public void setPersona(String nombre, String dni, String email)`

Pregunta 7. Una clase `Divisas` contiene el método:

```
static double cotizacionEuro();
```

y existe la instancia `instDivisas` de la clase `Divisas`. Indicar cuál de las siguientes instrucciones es la más adecuada en POO:

a) `double valorEuro = instDivisas.cotizacionEuro();`
b) `double valorEuro = Divisas.cotizacionEuro();`
c) `double valorEuro = ((Divisas) instDivisas).cotizacionEuro();`
d) `Ninguna de las anteriores`

Pregunta 8. Marca la respuesta correcta. Los métodos redefinidos (*sobreescritos, override*):

a) Están contenidos en una misma clase y presentan la misma firma
b) Están contenidos cada uno en una clase diferente sin relación entre sí, y presentan la misma firma
c) Están contenidos, cada uno, en una superclase y en su clase derivada, y presentan firmas diferentes
d) Ninguna de las anteriores

Pregunta 9. Marca la respuesta correcta:

a) Los interfaces pueden contener definiciones abstractas de constructores
b) Las clases abstractas pueden tener todos sus métodos abstractos
c) Los interfaces pueden contener propiedades no constantes, siempre que sean privadas
d) Ninguna de las anteriores

Pregunta 10. Tenemos un interfaz `ISemaforo` y su interfaz especializado `ISemaforoDerecha`. También tenemos una clase `Semaforo` y su clase derivada `SemaforoDerecha`, que 'siguen las directrices' de ambos interfaces. Indicar cuál es la definición correcta de la clase `SemaforoDerecha`:

a) `public class SemaforoDerecha implements Semaforo extends ISemaforoDerecha`
b) `public class SemaforoDerecha extends Semaforo, ISemaforoDerecha`
c) `public class SemaforoDerecha extends Semaforo implements ISemaforoDerecha`
d) `public class ISemaforoDerecha implements SemaforoDerecha`

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

Apellidos: _____

Parte 2. CODIFICACIÓN (6 puntos)

Teniendo en cuenta la siguiente interface en la que se define el funcionamiento que ha de tener una cola FIFO (*first-in, first-out*) genérica:

```
// FIFO (first-in, first-out)
public interface IColaGenerica<T> {

    // introduce un elemento en la cola
    void meter(T elemento) throws ExcepcionColaLlena;

    // saca un elemento de la cola
    T sacar() throws ExcepcionColaVacia;

    // comprueba si la cola está vacía
    boolean vacia();

    // comprueba si la cola está llena
    boolean llena();
}
```

Problema 1. (1 *punto*) Definir la cabecera de la clase `ColaGenerica` que implementa esta interface y que permite la creación de colas genéricas.

```
public class ColaGenerica<E> implements IColaGenerica<E>
```

Definir la cabecera de la clase `ColaGenericaI` que implementa esta interface y que permite la creación de colas que sólo manejan enteros.

```
public class ColaGenericaI implements IColaGenerica<Integer>
```

Problema 2. (*1 punto*). Suponiendo que empleamos un objeto de tipo `List` para almacenar los elementos de la cola, definir el atributo de la clase `ColaGenerica` necesario para dicho almacenamiento y el constructor vacío de la clase `ColaGenerica` que lo instancia.

```
private List<E> cola;

public ColaGenerica(){
  cola = new ArrayList();
}
```

Problema 3. (*1 punto*). Implementar los métodos `meter` y `sacar` de la clase `ColaGenerica`.

```
public void meter(T elemento) throws ExcepcionColaLlena {
    cola.add(elemento);
}

public T sacar() throws ExcepcionColaVacia {
    if(vacia()){
       throw new ExcepcionColaVacia();
    }
    return cola.remove(0);
}
```

Problema 4. (*1 punto*). Implementar la clase `ExcepcionColaVacia`.

```
public class ExcepcionColaVacia extends Exception {
    public ExcepcionColaVacia () {
        super("ExcepcionColaVacia");
    }
}
```

Problema 5. (*1 punto*). Instanciar un objeto de la clase `ColaGenerica` para manejar objetos de la clase `Integer`.

```
ColaGenerica<Integer> cola = new ColaGenerica<>();
```

Problema 6. (*1 punto*). Teniendo en cuenta que empleamos un `ArrayList` para almacenar los elementos de la cola ¿Sería necesario emplear la excepción `ExcepcionColaLlena` en el método meter de la clase `ColaGenerica`? ¿y si empleamos un `array` en lugar de un `ArrayList`? Razonar la respuesta.

Como los *ArrayList* son estructuras dináminas a diferencia de los *arrays*, nunca se van a llenar y como consecuencia la excepción *ExcepcionColaLlena* no se generará.

Como los *array* son estructuras estáticas necesitan crearse que un tamaño predeterminado, con lo que cuando llegue al máximo de su capacidad se lanzar/controlar la excepción *ExcepcionColaLlena* para que no se agregue ningún nuevo elemento.