Comenzado el	jueves, 9 de junio de 2022, 11:00
Estado	Finalizado
Finalizado en	jueves, 9 de junio de 2022, 12:21
Tiempo empleado	1 hora 20 minutos
Puntos	20,50/26,00
Calificación	**7,88** de 10,00 (**78,85**%)

Pregunta **1**

Correcta

Se puntúa 1,00 sobre 1,00

Elija la/s respuesta/s correctas. Una elección equivocada tiene penalización.
El patrón State permite:

```
Correcta vale 1
Incorrecta vale -0,5
```

Seleccione una:

- a. Permitir que un objeto cambie el comportamiento al cambiar el estado ✔
- b. Determinar un algoritmo general en la clase base y determinar los comportamientos particulares a través del mecanismo de herencia
- c. Permite añadir responsabilidades a un objeto en tiempo de ejecución

Respuesta correcta

Permitir que un objeto cambie el comportamiento al cambiar el estado

La respuesta correcta es: Permitir que un objeto cambie el comportamiento al cambiar el estado

Pregunta **2**

Correcta

Se puntúa 1,00 sobre 1,00

La serialización es una forma de hacer una clonación

- a. F ✔
- b. V

Respuesta correcta

Las respuestas correctas son:
V,

F

Pregunta **3**

Correcta

Se puntúa 1,00 sobre 1,00

Un objeto de tipo Observer sólo puede observar a un objeto de tipo Observable

```
Correcta: 1
Incorrecta: -1
```

a. F ✔

b. V

Respuesta correcta

Las respuestas correctas son:
F,

V

Pregunta **4**

Incorrecta

Se puntúa -0,50 sobre 1,00

Es correcta la implementación de los siguientes métodos?

```
Correcta: 1
Incorrecta: -0,5
```

```
public class Primero
{
    private String nombre;
    private long dni; // identificador que no se repite
    private Date fecha_nac;

    public Primero()
    {
        super();
    }
    @Override
    public boolean equals(Object object)
    {
        if (this == object)
        {
            return true;
        }
        if (!(object instanceof Primero))
        {
            return false;
        }
        final Primero other = (Primero) object;
        if (dni != other.dni)
        {
            return false;
        }
        return true;
    }
    @Override
    public int hashCode()
    {
        final int PRIME = 37;
        int result = 1;
        result = PRIME * result + ((nombre == null) ? 0 : nombre.hashCode());
        result = PRIME * result + (int) (dni ^ (dni >>> 32));
        result = PRIME * result + ((fecha_nac == null) ? 0 : fecha_nac.hashCode());
        return result;
    }
}
```

a. F

b. V ✖

Respuesta incorrecta.

La respuesta correcta es:
F

Pregunta **5**

Correcta

Se puntúa 1,00 sobre 1,00

Se tiene una clase Docente, que se extiende de Persona.
Se necesita que trabaje en forma concurrente (solamente la clase Docente). Para lograr ésto y no perder la línea de herencia ....

Elija todas las opciones correctas.

La opciones incorrectas elegidas como correctas tienen penalización.

Seleccione una o más de una:

- [ ] a. Se debe implementar la interface Runnable en la clase Persona.
- [x] b. Se debe crear un nuevo objeto de tipo Thread. ✔
- [x] c. Se debe implementar la interface Runnable en la clase Docente. ✔
- [ ] d. Se debe comenzar la ejecución del objeto de tipo Docente con la instrucción run()

Respuesta correcta

Las respuestas correctas son: Se debe implementar la interface Runnable en la clase Docente., Se debe crear un nuevo objeto de tipo Thread.

Pregunta **6**

Finalizado

Se puntúa 4,00 sobre 5,00

Determine el código del recurso compartido para el siguiente caso:

Se tiene una biblioteca con un catalogo de 20 diferentes libros. De cada libro existen varios ejemplares/copias.

El socio de la biblioteca solicita un determinado libro del catalogo. Si está disponible alguna copia del libro, se lo lleva, si no está disponible, queda en espera hasta que lo devuelvan.

Diseñe y escriba código java del recurso compartido.

Escriba una versión con wait/notify y otra con semáforos. Las dos. La de semáforos vale menos que la de wait/notify.

//El recurso compartido será el libro, que se encuentra dentro de la biblioteca

//---------- Versión con Wait and Notify----------

```
public class Biblioteca
{
   private ArrayList <libro> Libros;

   public void retirarLibro(...) //socio, libro, arraylist libros
   {
      synchronized (this.libros)
      {
         while(...) //condición para liberar o no el bloqueo recurso compartido
         {
            this.Libros.waitt(); //libero recurso compartido
         }
         this.libros.nofityAll(); // decidí elegir notifyAll para que todos los hilos en espera sean notificados
      }
   }
}
```

//---------- Versión con Semáforos----------

```
public class Biblioteca
{
   private Semaphore Semaforo;
   private ArrayList <libro> Libros;

   public Biblioteca()
   {
      Semaforo = new Semaphore(20, true);
      //semaforo con capacidad para 20 permisos
   }

   public void retirarLibro(...)
   {
      try
      {
         Semaforo.acquire();
      }
      catch(InterruptedException ie)
      {
```

```
            System.out.println("No es posible retirar el libro.");
        }
    }

    public void devolverLibro(....)
    {
        Semaforo.release();
    }
}
```

Comentario:
Falta el método que devuelve un libro, es la contraparte del métoo que retira libro.

Pregunta **7**

Correcta

Se puntúa 1,00 sobre 1,00

Elija la/s respuesta/s correctas. Una elección equivocada tiene penalización.
Según la clasificación, el patrón State es:

```
Correcta vale 1
Incorrecta vale -0,5
```

Seleccione una:

- a. Estructural
- b. de Comportamiento ✔
- c. Creacional

Respuesta correcta

de Comportamiento

La respuesta correcta es: de Comportamiento

Pregunta **8**

Finalizado

Se puntúa 5,00 sobre 5,00

Determine un ejemplo (novedoso, distinto de los mostrados en clase o los TP) de uso del patrón Observer/Observable. Establezca una breve descripción del caso y bosqueje los aspectos fundamentales de las clases participantes (herencia, implementación, atributos, constructores, invocaciones de métodos específicos, partes esenciales del método update(...), etc.

No es necesario completar todos los métodos relacionados con el Dominio del ejemplo, o sea, los cálculos propios del problema a resolver, solamente se debe aclarar lo específico del patrón.

Explique brevemente el significado de cada línea de código importante. Esta explicación es fundamental para aprobar el ejercicio.

Se me ocurrió continuar con el ejemplo de la Biblioteca y el usuario que quiere retirar algún libro. Dicho usuario observará el estado del libro que desea retirar para ver si está disponible o no.

```
public class Libros extends Observable
{
   private Map disponibles = new HashMap();

   public void devolverLibro(String nombre, String autor) throws LibroIncorrectoException
   {
      if (!libroValido(nombre, autor))
      {
         throw new LibroIncorrectoException(nombre, autor);
         //si el libro ingresado no existe lanza una excepción
      }
      EstadoLibro estado = new EstadoLibro(nombre, autor);
      //como el libro existe crea un nuevo estado del mismo
      disponibles.put(nombre, estado);
      //carga el libro a los libros disponibles
      setChanged();
      //cambia el estado del libro agregado
      nofityObservers(estado);
      //notifica a todos los observers el cambio de estado
   }

   public void retirarLibro(EstadoLibro estado)
   {
      disponibles.remove(estado.nombre());
      //al retirar un libro éste deja de estar disponible
      setChanged();
      //cambia el estado del libro agregado
      nofityObservers(estado);
      //notifica a todos los observers el cambio de estado


      //...
   }
}



public class Usuarios implements Observer
{
   Libros observado; //elemento observable

   public Usuarios(Libros libro)
   {
```

```
        observado = libro;
        observado.addObserver(this);
        //se agrega el observable
    }

    public void update(Observable libro, Object queEstado)
    {
        if (libro != observado)
        {
            throw new IllegalArgumentException();
            //lanza uina excepción si el libro a observar no se encuentra entre los observables
        }
        EstadoLibro estado = (EstadoLibro) queEstado;
        if (observado.disponibles(estado)) //actualiza el estado del objeto observado
            libroDisponible(estado);
        else
            libroNoDisponible(estado);
    }
}
```

Comentario:

Pregunta **9**

Incorrecta

Se puntúa -0,50 sobre 1,00

En el Patrón MVC.
¿De quién es la responsabilidad de validar el formato correcto de los datos que el usuario ingresa?

Seleccione una:

- a. Controlador ✖
- b. Modelo
- c. Vista

Respuesta incorrecta.

La respuesta correcta es: Vista

Pregunta **10**

Correcta

Se puntúa 1,00 sobre 1,00

Es incorrecto puede usar T (el tipo de datos parametrizado) como tipo de un campo estático o en cualquier lugar dentro de un método estático o inicializador estático.

- a. V ✔
- b. F

Respuesta correcta

Las respuestas correctas son:
V,

F

Pregunta **11**

Correcta

Se puntúa 1,00 sobre 1,00

El patrón Observer/Observable está pensado para aplicaciones WEB.

```
Correcta: 1
Incorrecta: -0,5
```

- a. V
- b. F ✔

Respuesta correcta

La respuesta correcta es:
F

Pregunta **12**

Correcta

Se puntúa 1,00 sobre 1,00

La función específica del método wait() de un recurso compartido es poner en espera al hilo que lo contiene y liberar al recurso compartido

```
Correcto: 1
Incorrecto: -1
```

a. V ✔

b. F

Respuesta correcta

La respuesta correcta es:
V

Pregunta **13**

Correcta

Se puntúa 1,00 sobre 1,00

El uso de tipos genéricos permite la comprobación de tipos correctos en tiempo de compilación.

```
Valor respuesta correcta: 1
Valor respuesta incorrecta: -0,5
```

a. V ✔

b. F

Respuesta correcta

La respuesta correcta es:
V

Pregunta **14**

Correcta

Se puntúa 1,00 sobre 1,00

En el patrón MVC, es conveniente que la clase vista cree el modelo y el controlador para poder dar inicio al programa.

```
Correcta: 1
Incorrecta: -0,5
```

a. V

b. F ✔

Respuesta correcta

La respuesta correcta es:
F

Pregunta **15**

Incorrecta

Se puntúa -0,50 sobre 1,00

La función específica de la instrucción wait() es evitar que dos o más hilos accedan en forma simultánea a un recurso compartido

```
Correcta: 1
Incorrecta: -1
```

a. F

b. V ✖

Respuesta incorrecta.

La respuesta correcta es:
F

Pregunta **16**

Correcta

Se puntúa 1,00 sobre 1,00

En el patrón MVC, la Vista es la encargada de gestionar los eventos producidos por los componentes activos (botones, enlaces, etc).

```
Correcta: 1
Incorrecta: -0,5
```

a. F ✔

b. V

Respuesta correcta

La respuesta correcta es:
F

Pregunta **17**

Correcta

Se puntúa 1,00 sobre 1,00

Para poder persistir un objeto utilizando archivos XML es necesario

Seleccione una:

a. Que todos sus métodos sean públicos y sus atributos privados

b. Ninguna de las anteriores

c. Tener un constructor que nos permita pasar por prámetro los atributos que queremos persistir

d. Tener un constructor público sin parámetros y getters y setters públicos ✔

Respuesta correcta

La respuesta correcta es: Tener un constructor público sin parámetros y getters y setters públicos

Pregunta **18**

Correcta

Se puntúa 1,00 sobre 1,00

Al momento de definir un bloque synchronized debemos:

```
Correcta: 1
Incorrecta: -0,5
```

Seleccione una:

- a. Hacerlo lo mas pequeño posible, para aprovechar la concurrencia. ✔
- b. Hacerlo lo mas extenso posible por precaución, para no poner en riesgo la zona crítica.

Respuesta correcta

La respuesta correcta es: Hacerlo lo mas pequeño posible, para aprovechar la concurrencia.

◄ Ejemplo Persistencia cuando hay patron Singleton

Ir a...