Trabajo Práctico – Clases Abstractas vs. Interfaces

MENÚ

TRABAJO PRÁCTICO – CLASES ABSTRACTAS VS. INTERFACES

Trabajo Práctico – Clases Abstractas vs. Interfaces

Objetivos

Comprender las diferencias, ventajas y desventajas del uso de clases abstractas e interfaces en Java; tanto a nivel de diseño como de implementación.

Tips:

- Recuerde que para validar su implementación debe crear **tests de unidad**.
- Lea la documentación de las clases del sistema que va a manipular como así también los métodos que implementan. De esta forma va a poder saber cómo utilizar sus instancias o métodos de clase. Si no se visualiza el código fuente en su ambiente de programación recuerde los videos tutoriales que están publicados en la página de la materia.
- Recuerde que además de las clases presenciales puede hacer uso de la lista de la materia para evacuar cualquier duda o comentario que tenga sobre los trabajos prácticos. La lista para realizar consultas tpi-est-obj2@listas.unq.edu.ar

Interfaces en Java

En Java, una interface se define de la siguiente manera:

```
package nombrePaquete;
   {imports}
[public] interface NombreInterfaz [extends ListaDeSuperInterfaces] {
   [Constantes]
   [Signaturas de métodos]
}
```

Las clases que necesiten implementar interfaces deberán hacer de la siguiente manera:

```
class NombreDeLaClase implements InterfaceUno, InterfaceDos…{}
```

Actividad de lectura: clases abstractas e interfaces

Lea el siguiente material online:

- What is inheritance? https://docs.oracle.com/javase/tutorial/java/concepts/inheritance.html
- Multiple Inheritance of State, Implementation, and Type https://docs.oracle.com/javase/tutorial/java/IandI/multipleinheritance.html
- Interfaces: https://docs.oracle.com/javase/tutorial/java/concepts/interface.html

Luego, debata y responda las siguientes preguntas:

1. ¿Qué ventajas, en cuanto a polimorfismo, brindan las interfaces en Java?

2. ¿Por qué no siempre puedo utilizar clases abstractas para agrupar clases polimórficas?
3. ¿Qué ventajas tienen las clases abstractas sobre las interfaces?
4. ¿Se puede instanciar una interface?
5. ¿Por qué no es recomendable incrementar o modificar las firmas definidas en un interface?
6. ¿Por qué, en lenguajes como Smalltalk, no es necesaria la implementación de interfaces?

La caja del mercado central

El mercado central necesita un sistema para el manejo de su caja y ud. es el encargado de diseñarlo e implementarlo. La especificación obtenida a través del analista funcional plantea que el funcionamiento de las cajas es sencillo: se registran los productos que desea adquirir el cliente y se obtiene el monto total a pagar que luego se informa al cliente. En particular, registrar un producto significa consultarle su precio para acumularlo en el monto a pagar y decrementar el stock existente de ese producto. El mercado diferencia a los productos de cooperativas de los productos de empresas tradicionales. La diferencia principal es que el producto de cooperativa aplica a su precio un descuento del 10%, en concepto de IVA, de su precio base.

1. Diseñe la solución con un diagrama de clases UML.
2. Implemente su diseño en Java.
3. Cree tests que verifiquen el correcto funcionamiento de su implementación.

La caja del mercado central - Parte 2

Su sistema es un éxito, tal es así que el mercado ha decidido volver a contratarlo para actualizar el sistema y poder incorporar el pago de facturas de servicios e impuestos en las cajas de venta. Los servicios tienen un costo por unidad consumida y una cantidad de unidades consumidas en el período facturado, y para calcular el monto a pagar deben multiplicarse ambas cantidades. En el caso de los impuestos, el monto a pagar está determinado por la tasa del servicio que es un valor fijo en pesos. Además, es necesario registrar el pago del servicio/impuesto, que en ambos casos consiste en notificar a la agencia recaudadora. Para ello, se dispone de una Interface como se describe a continuación.

```
public interface Agencia{
    public void registrarPago(Factura factura);
}
```

Su sistema funciona correctamente, y usted no quiere modificar algo que funciona (si no está roto, no lo arregles). ¿Es posible diseñar una nueva solución al problema de cobrar en las cajas tanto productos como servicios alterando lo menos posible la solución existente? ¿Cómo?

1. Implemente su diseño en Java.
2. Cree tests con el fin de verificar el correcto funcionamiento de su implementación.

Interfaces, Colecciones y otras yerbas

Para resolver este ejercicio, será necesario que estudie el funcionamiento y protocolo de Collection, List, ArrayList y LinkedList.
Para ello, puede utilizar como bibliografía los sitios web de la documentación de clases e interfaces de Oracle:

- Collection: http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html
- List: http://docs.oracle.com/javase/7/docs/api/java/util/List.html
- ArrayList: http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html
- LinkedList: http://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html

Considere la siguiente clase:

```
public class ColeccionadorDeObjetos {

    public Object getFirstFrom(XXX collection){

        return collection.get(0);

    }
```

```
        public void addLast (Object element, YYY collection){

            collection.add(element);

        }

        public Collection getSubCollection(ZZZ collection, int from, int to){

            return collection.subList(x,y);

        }

        public Boolean containsElement(WWW collection, Object element){

            return collection.contains(element);

        }

    }
```

Como notará, los tipos de las colecciones no están definidos. ¿Qué ocurriría si utiliza en lugar de XXX, YYY, WWW y ZZZ las clases e interfaces recientemente estudiadas?

De Smalltalk/Wollok a Java: Personas y Mascotas

1. Defina la clase `Persona` y modélela en Smalltalk/Wollok. Una persona tiene un nombre, un DNI y una fecha de nacimiento, por lo que debe ser posible pedirle su nombre, DNI, fecha de nacimiento, edad y debe ser posible compararla en edad con otra persona mediante el mensaje “<”.
2. Defina la clase `Mascota` y modélela en Smalltalk/Wollok. Una mascota tiene un nombre y una raza (String), por lo que debe ser posible pedirle tales datos.
3. Instancie dos `Persona`s, dos `Mascota`s e inserte los cuatro objetos en una colección. Itere sobre la colección e imprima un listado con los nombres de todos los objetos de la misma.
4. Durante la iteración, ¿fue necesario distinguir si el objeto era una `Persona` o una `Mascota` para poder imprimir su nombre? ¿Cómo se llama el mecanismo de abstracción que permite esto?
5. ¿Podría asegurar que tanto mascotas como personas respondan al mensaje `getName()` mediante herencia? ¿Cómo? ¿Qué ocurre si `Persona` es subclase de otra clase que no puede ser modificada por ud.? ¿De qué otra manera puede asegurar el polimorfismo, teniendo en cuenta que está implementando en Java?
6. Implemente, ahora en Java, los puntos 1, 2 y 3 según las consideraciones de 4 y 5.