COLECCIÓN

Es un conjunto de elementos que están definidos por un mismo tipo.

Un concepto similar, lo escuchamos nosotros en LPE, es el de los arreglos, pero a diferencia de estos, las colecciones son dinámicas, mientras que los arreglos son estáticos.

Tipos de colecciones: Existen 3 tipos y son los siguientes:

- Diccionarios.- En otros lenguajes diferentes al Java se les conoce como "Dictionary", en java se le conoce como "Map".

  Este tipo de colecciones se les invoca con un nombre o también llamado llave. Estos, teóricamente, van a responder con un solo objeto.

  Tiene asociado 4 clases a las que implementa:

  -HashMap
  -Hashtable
  -LinkedHashMap
  -TreeMap

- Lista.- En java es conocido como "List".

  Este tipo de colecciones se invoca mediante un índice, además este tipo de colecciones puede aceptar elementos duplicados.

  Tiene asociado 3 clases a las que implementa:

  -ArrayList
  -Vector
  -LinkedList

- Conjunto.- En java se le conoce como "Set".

  Este tipo de colecciones solo aceptan elementos únicos. Por ello, su utilidad es que cuando agrego un item(que ya existe) entonces no lo toma en cuenta.

  Tiene asociado 3 clases a las que implementa:

  -HashSet
  -LinkedHashSet
  -TreeSet

INTERFACES

Idea de interfaz.- Es aquel que da una respuesta. Es como si fuese un contrato que establece el qué vas a exponer mas no cómo lo vas a implementar.

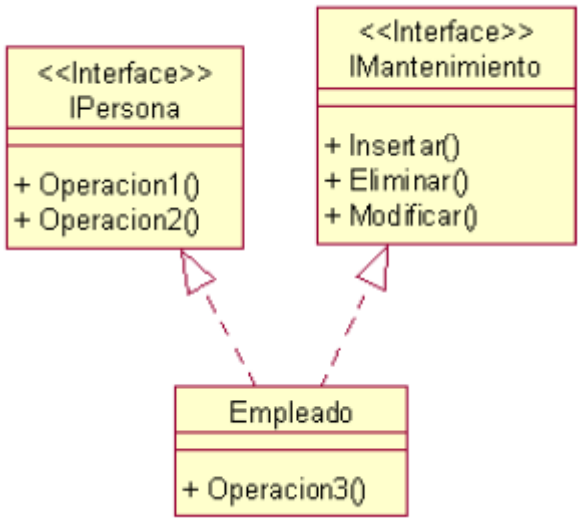
Definición de interfaz.- Es un tipo, que permite la especificación sobre qué hacer. No implementa la especificación.

Para entender mejor esto, diferenciaremos lo que significa implementar de especificar: Implementación.-cómo lo debe de hacer.
Especificación.- es lo que debe de hacer.

Para Java los atributos de una interface son constantes.

Representación de una interfaz en diagramas UML

La representación de una interface es similar a la de herencia, la diferencia es que la línea es discontinua y la punta de la flecha está vacía.



Ahora bien la clase que implementa todos los métodos de la interfaz se llamará clase concreta. En otras palabras implementa todas las especificaciones de la interfaz.

Suponiendo que en el gráfico mostrado tengamos dentro de la clase Empleado, los métodos "Operacion1()" y "Operacion2()", entonces podríamos decir que Empleado es la clase concreta de la interfaz Persona, o en otras palabras la clase Empleado implementa la interfaz Persona.

Sintaxis de una Interfaz en Java

A diferencia de las clases la interfaz se declara con la palabra reservada "interface" y dentro de ella se declaran sus especificaciones utilizando la palabra reservada "throws". Por ejemplo:

```
public interface AccesoDAO{

        public Acceso agregarAcceso(Acceso acceso)
                        throws DAOException;

}
```

Si nos damos cuenta la especificación "agregarAcceso" no lleva llaves, y para este caso, las excepciones ya no se colocan dentro de un try y luego se capturaban mediante un catch. Ahora las excepciones son lanzadas mediante el "throws".

Implementación de interfaces y su Sintaxis

Dentro de una implementación se tienen 3 elementos una interfaz, una clase abstracta que puede implementar ciertas o todas las especificaciones de la interfaz y por último está la clase concreta.

-Clase abstracta.- puede o no tener métodos abstractos. No puedo crear objetos ni métodos de objetos, porque es una clase abstracta. Pero sí puedo crear objetos.

-Un método abstracto no tiene implementación, sólo va a tener punto y coma “ ; ”.

En el ejemplo de las diapositivas presentadas en clase, teníamos una interfaz llamada AccesoDAO, una clase que contenía las transacciones básicas llamada BaseDaoHibernate y por último la clase concreta AccesoDaoHibernate.

De este ejemplo la clase concreta AccesoDaoHibernate implementa la interfaz y hereda de la clase "BaseDaoHibernate" los métodos básicos.

La sintáxis sería la siguiente:

```
public class AccesoDaoHibernate extends BaseDaoHibernate implements AccesoDao{

        public Acceso agregarAcceso(Acceso acceso) throws DAOException {
        try{  save((Object) acceso);  }
        catch (HibernateException e){  throw new DAOException ("Error hibernate");  }
        catch (Exception e){  throw new DAOException ("Error interno SQL");  }
        return acceso;
        }

}
```

En este ejemplo se observa nuevamente el try y el catch, pero por el hecho de que estoy implementando una interfaz, para capturar una excepción tengo, necesariamente, que utilizar el "throw" para lanzar la excepción, sino no podré capturar ninguna excepción, ya que no lo podrá reconocer.