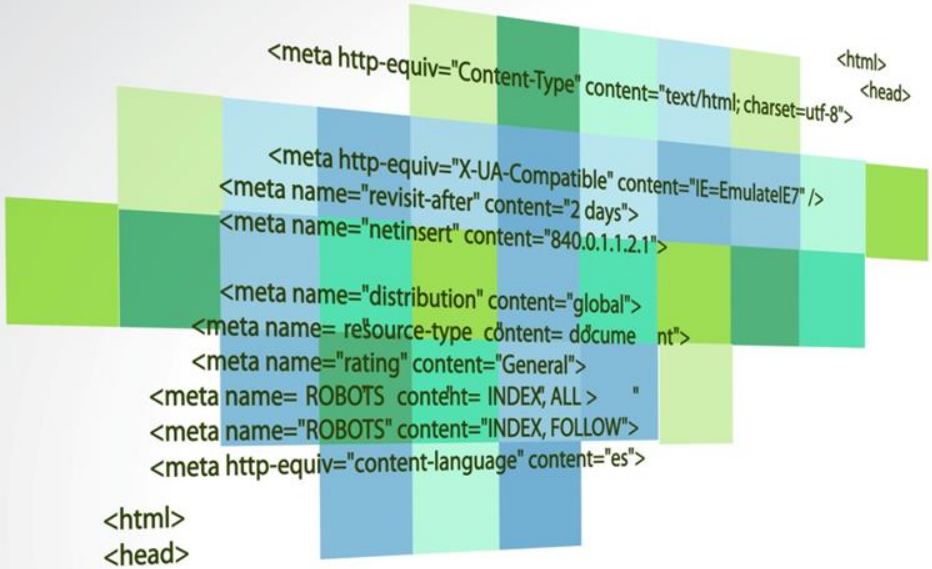


PROGRAMACIÓN ORIENTADA A OBJETOS

Material de trabajo autónomo

Colecciones/Arreglos de Objetos

UNIDAD 2

Indicaciones

Para un estudio eficaz, te invitamos a que sigas las siguientes recomendaciones:

Lee con atención

Relaciona

Contrasta y complementa

Elabora

Realiza la tarea y participa en el foro

Logros de la sesión

Al finalizar esta sesión, estarás preparado para:

- ✓ Utilizar los arreglos para implementar la colaboración entre clases.

Temario

- Definición de Colección
- Creando colecciones
- Implementando colecciones en clases colaborativas

Motivación

- El paradigma orientada a objetos se inicio en los años 70. Hoy en día la mayoría de los lenguajes son orientados a objetos.

Java Collection Framework

Java Collection Framework es un conjunto de tipos genéricos que se puede usar para crear colecciones que sirve para cargar y manejar objetos en meomoria.
Estos objetos deben pertenenecer a una misma clase, esta clase se denomina Generica.

ArrayList

Existen varios tipos de colecciones, pero los que más utilizaremos para este curso son el ArrayList y el HashMap, ambos se encuentran en el paquete **java.util** de java.

ArrayList : Permiten coleccionar una secuencia de objetos de una misma clase en memoria, es dinámico, es decir, no es necesario predefinir el tamaño de la colección al definirse, sino que el número de elementos va creciendo en forma automática sin límites más que de la memoria que dispones en tu máquina.
Ideal para reportes, muy rápido más no soporta concurrencia.

Supongamos que tenemos la clase producto:

```
public class Producto {

    private String codigo;
    private String descripcion;
    private double precio;

    public Producto(String codigo, String descripcion, double precio) {
        this.codigo = codigo;
        this.descripcion = descripcion;
        this.precio = precio;
    }

    public String getCodigo() {
        return codigo;
    }

    public void setCodigo(String codigo) {
        this.codigo = codigo;
    }

    public String getDescripcion() {
        return descripcion;
    }

    public void setDescripcion(String descripcion) {
        this.descripcion = descripcion;
    }

    public double getPrecio() {
        return precio;
    }

    public void setPrecio(double precio) {
        this.precio = precio;
    }
}
```

Y un Administrador necesita saber cuanto suma todos los precios de los productos que tiene en su almacén.
Entonces se puede deducir que el Administrador posee una colección de Productos representado por:

```
import java.util.ArrayList;
import java.util.List;

public class Administrador {
    private List<Producto> catalogo = new ArrayList<Producto>();

    public List<Producto> getCatalogo() {
        return catalogo;
    }

    public void setCatalogo(List<Producto> catalogo) {
        this.catalogo = catalogo;
    }
}
```

Y justamente como el posee toda la información de todos los productos, entonces un método podría realizar la suma de sus precios, este método sumarPrecios() pertenecería a la clase Administrador:

```
public class Administrador {
    private List<Producto> catalogo = new ArrayList<Producto>();

    public double sumarPrecios(){
        double suma = 0;
        for(Producto p:catalogo){
            suma+=p.getPrecio();
        }
        return suma;
    }
```

Como funciona esta lógica del método sumarPrecios()?

Clase a la que pertenece cada objeto de la colección

"p" puntero que referencia al primer elemento de la colección

```
pub    double sumarPrec  s(){
    do le suma = 0;
    for(Producto p:catalogo){
        suma+=p.getPrecio();
    }
    return suma;
}
```

Bucle repetitivo que desplaza elemento por elemento en la colección, cada vez que repite el bucle.

Identificador o nombre de la colección

"p" podría accede a cualquier método del objeto que apunte en ese meomento en la colección

Como probamos que este método es correcto, como hacemos para que la colección catalogo tenga objetos para realizar la prueba?
La primera respuesta es sencilla usando pruebas unitarias, pero para la segunda.
Se necesita en la clase Administrador un método que permita insertar objetos producto a la colección.
El método podría llamarse: adicionar y sería como el siguiente método.

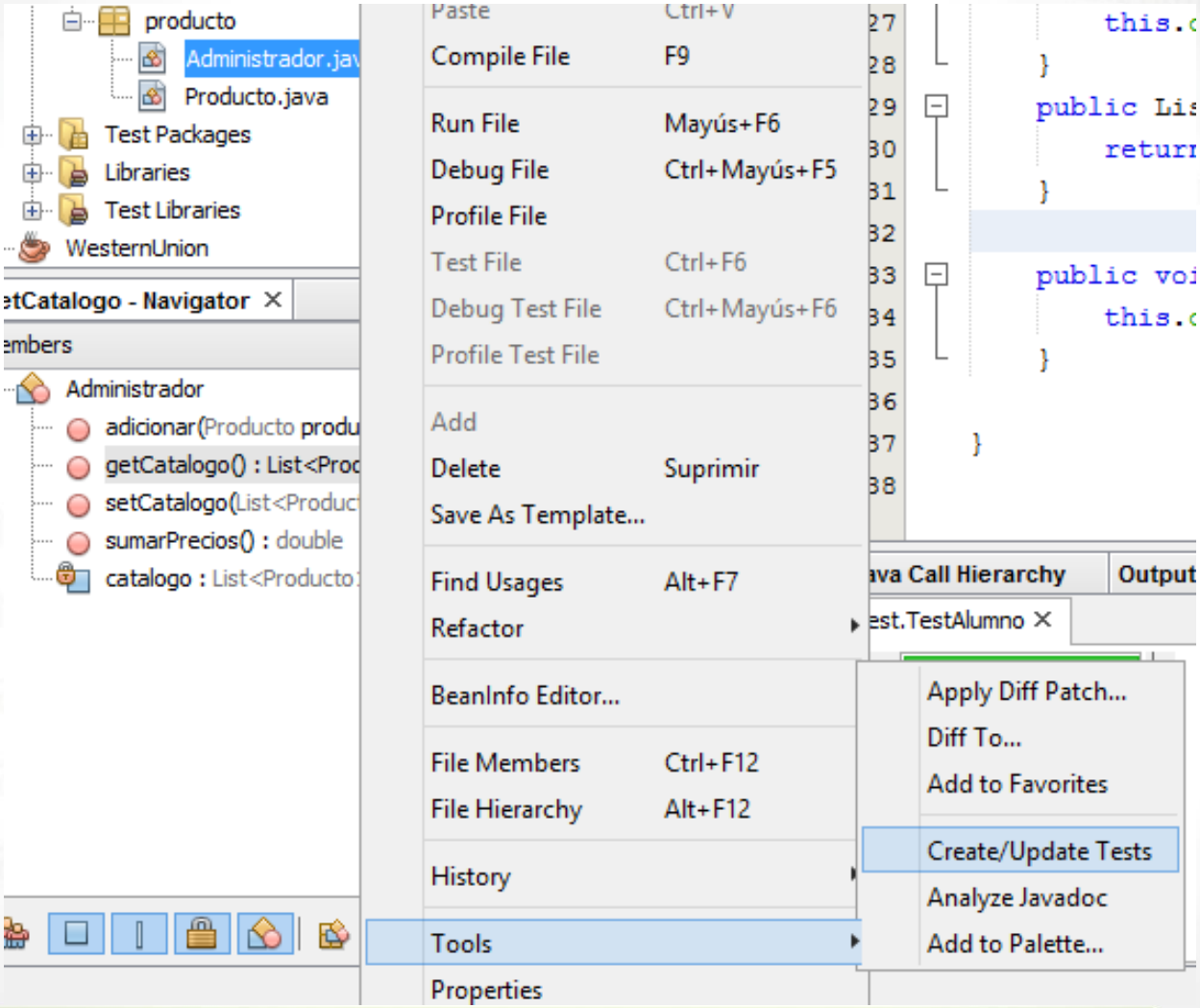
```
public void adicionar(Producto producto){
    this.catalogo.add(producto);
}
```

TIP: add es un método propio de la clase ArrayList, que permite adicionar elementos a la colección, además de este método hay otros muy útiles como size() = tamaño de la colección, remove(elemento) elimina el elemento u objeto de la colección entre otros.

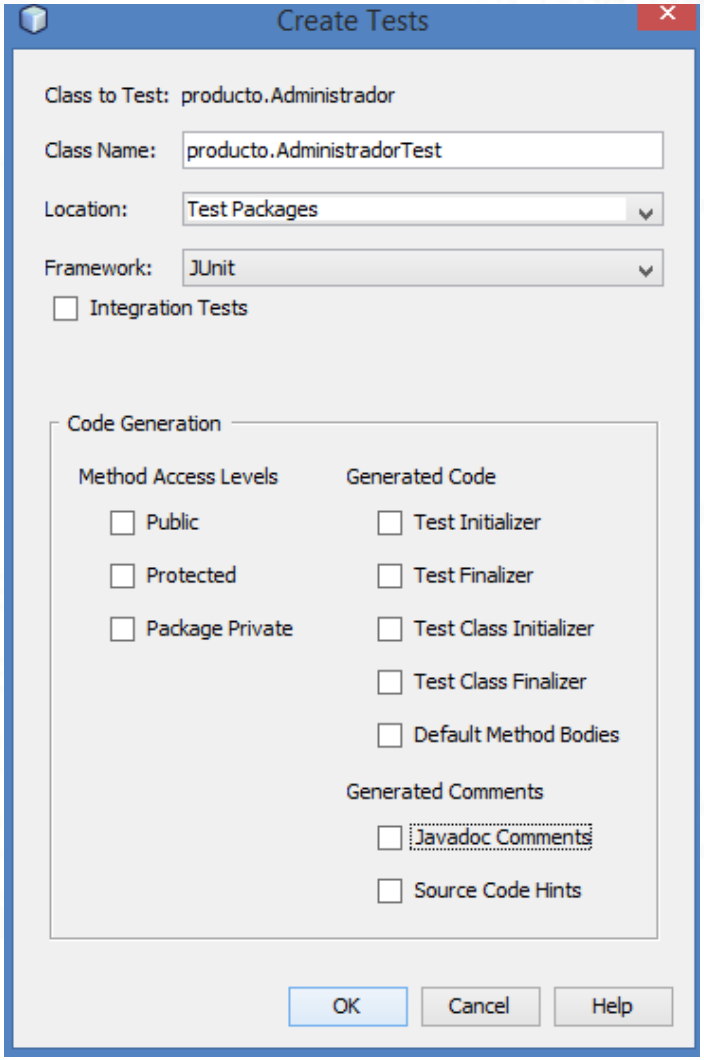
La clase quedaría como:

```
import java.util.ArrayList;
import java.util.List;
public class Administrador {
    private List<Producto> catalogo = new ArrayList<Producto>();

    public double sumarPrecios(){
        double suma = 0;
        for(Producto p:catalogo){
            suma+=p.getPrecio();
        }
        return suma;
    }
    public void adicionar(Producto producto){
        this.catalogo.add(producto);
    }
    public List<Producto> getCatalogo() {
        return catalogo;
    }

    public void setCatalogo(List<Producto> catalogo) {
        this.catalogo = catalogo;
    }
}
```

Si e stuvieramos usando Netbeans el test unitario lo podríamos crear, hacienda click Derecho sobre la clase Administrador y seleccionar / Tools / Create…:



Desactivamos todos lo checks y presionamos Ok:



El test finalmente podría ser como:

```
@Test
public void obtenerSuma()
{
    Producto producto1 = new Producto("001", "Microprocesador i5", 240.00);
    Producto producto2 = new Producto("002", "Mouse Logitech", 60.00);
    Administrador administrador = new Administrador();
    administrador.adicionar(producto1);
    administrador.adicionar(producto2);
    double resultado;
    resultado = administrador.sumarPrecios();
    assertEquals(producto1.getPrecio() + producto2.getPrecio(), resultado, 0.00);
}
```

Aquí los fuentes del proyecto:

Conclusiones

- Las relaciones entre clases nos sugieren colaboración entre ellas para lograr un resultado esperado.
- Los arreglos o colecciones de objetos nos permiten implementar las relaciones de composición con varios objetos.
- En la vida real existen muy a menudo este tipo de relaciones entre clases y la programación orientada aobjetos los representar de la manera más sencilla y objetiva.
- Las colecciones son dinámicas y sin límite no es necesario definir un número exacto del tamaño de la colección.

Preguntas...

Si, luego del estudio del material, tienes dudas sobre alguno de los temas, ingresa al Aula Virtual y participa en el **foro de dudas académicas** de la unidad

Si quieres conocer más…

- Te invito a leer más sobre estos temas de los siguientes libros