Edison Wladimir Morocho Guayanlema
Ingenieria en sistemas de información
Sexto Nivel
Paralelo **A**
Ingenieria de software II

METODO CREACIONAL BUILDER

*Celular.java TestCelular.java

```
1  package patronesCreacionales.patronBuilder;
2  public class Celular {
3      private String marca;
4      private String modelo;
5      private int ram;
6      private double almacenamiento;
7      private Celular() {}
8      public String getMarca() {
9          return marca;
10     }
11     public String getModelo() {
12         return modelo;
13     }
14     public int getRam() {
15         return ram;
16     }
17     public double getAlmacenamiento() {
18         return almacenamiento;
19     }
```

Línea 2: Aquí comienza la definición de la clase Celular.
Líneas 3-6: Estas son las variables de instancia de la clase Celular, que representan las propiedades de un celular, como la marca, el modelo, la RAM y el almacenamiento.
Línea 7: Se declara un constructor privado Celular() para evitar que se cree una instancia de Celular directamente desde fuera de la clase. En su lugar, se utiliza un patrón de diseño Builder para crear objetos Celular.
Líneas 8-19: Estos son los métodos getter que permiten acceder a las propiedades de un objeto Celular.

```
20     //Clase interna builder
21     public static class builderCelular {
22         private String marca;
23         private String modelo;
24         private int ram;
25         private double almacenamiento;
26         public builderCelular setMarca(String marca) {
27             this.marca = marca;
28             return this;
29         }
30         public builderCelular setModelo(String modelo) {
31             this.modelo = modelo;
32             return this;
33         }
34         public builderCelular setRam(int ram) {
35             this.ram = ram;
36             return this;
37         }
38         public builderCelular setAlmacenamiento(double almacenamiento) {
39             this.almacenamiento = almacenamiento;
40             return this;
41         }
42         //metodo constructor de objeto
43         public Celular buildCelular(){
44             Celular celular = new Celular();
45             celular.marca = this.marca;
46             celular.modelo = this.modelo;
47             celular.ram = this.ram;
48             celular.almacenamiento = this.almacenamiento;
49             return celular;
50         }
51     }
52 }
```

Edison Wladimir Morocho Guayanlema
Ingenieria en sistemas de información
Sexto Nivel
Paralelo **A**
Ingenieria de software II

Línea 20: Se define una clase interna llamada builderCelular, que se utiliza para construir objetos Celular de manera más flexible.

Líneas 21-36: En la clase builderCelular, se definen métodos set para configurar las propiedades del Celular. Cada uno de estos métodos devuelve una instancia de builderCelular, lo que permite encadenar llamadas a los métodos set.

Líneas 38-44: El método buildCelular() crea una instancia de Celular utilizando los valores configurados previamente en la clase builderCelular y devuelve ese objeto Celular.

Celular.java | *TestCelular.java

```
1  package patronesCreacionales.patronBuilder;
2
3  public class TestCelular {
4
5      public static void main(String[] args) {
6          Celular celular = new Celular.builderCelular().setMarca("BISON")
7                  .setModelo("PRO X10").setRam(16)
8                  .setAlmacenamiento(256).buildCelular();
9
10         System.out.println("Marca: "+celular.getMarca());
11         System.out.println("Modelo: "+celular.getModelo());
12         System.out.println("Ram: "+celular.getRam()+"GB");
13         System.out.println("Almacenamiento: "+celular.getAlmacenamiento()+"GB");
14
15     }
16
17 }
18
```

Línea 4: Se crea un objeto Celular utilizando el patrón Builder. Se encadenan llamadas a los métodos setMarca, setModelo, setRam y setAlmacenamiento para configurar las propiedades del celular.
Líneas 6-9: Se configuran las propiedades del celular utilizando los métodos set.
Línea 10: Se llama al método buildCelular para construir el objeto Celular con la configuración especificada.
Líneas 12-15: Se imprimen las propiedades del celular, como la marca, el modelo, la RAM y el almacenamiento.

Edison Wladimir Morocho Guayanlema
Ingenieria en sistemas de información
Sexto Nivel
Paralelo **A**
Ingenieria de software II

METODO ESTRUCTURAL COMPOSITE

Dioses.java | DiosesGriegos.java | TestDiosesSemiDioses.java | TitanGriego.java

```java
package patronesEstructurales.patronComposite;

public interface Dioses
{
    void mostrarDioses();
}

```

Esta es una interfaz llamada Dioses que forma parte de un patrón estructural llamado Composite. La interfaz define un único método mostrarDioses(). En el contexto del patrón Composite, esta interfaz representa tanto los componentes individuales como los compuestos en una estructura jerárquica.
mostrarDioses(): Este método es utilizado para mostrar información sobre un objeto que implementa la interfaz Dioses. La implementación específica de este método variará dependiendo de la clase que implemente la interfaz Dioses.

Dioses.java | DiosesGriegos.java | TestDiosesSemiDioses.java | TitanGriego.java

```java
package patronesEstructurales.patronComposite;

public class DiosesGriegos implements Dioses
{
    private String nombreDios;
    private String encargado;

    DiosesGriegos(String nombre,String encargado)
    {
        this.nombreDios = nombre;
        this.encargado = encargado;
    }


    @Override
    public void mostrarDioses() {
        System.out.println("Dios: "+nombreDios);
        System.out.println("Dios del : "+encargado);

    }

}

```

package patronesEstructurales.patronComposite;: Al igual que en la interfaz, esta línea especifica el paquete al que pertenece la clase DiosesGriegos.

public class DiosesGriegos implements Dioses {: Aquí se define la clase DiosesGriegos, que implementa la interfaz Dioses. Esto significa que la clase debe proporcionar una implementación del método mostrarDioses() definido en la interfaz.

private String nombreDios; y private String encargado;: Estas líneas declaran dos campos (variables de instancia) privados en la clase DiosesGriegos. Uno es nombreDios para almacenar el nombre del dios griego y el otro es encargado para almacenar la descripción del área de responsabilidad del dios.

DiosesGriegos(String nombre, String encargado) { ... }: Aquí se define el constructor de la clase DiosesGriegos que toma dos argumentos, nombre y encargado, para inicializar los campos nombreDios y encargado al crear una instancia de esta clase.

@Override: Esta anotación se utiliza para indicar que el método que sigue a continuación (mostrarDioses()) es una implementación del método definido en la interfaz Dioses.

public void mostrarDioses() { ... }: Aquí se proporciona la implementación concreta del método mostrarDioses() requerido por la interfaz. En este caso, el método imprime en la consola el nombre del dios y su área de responsabilidad.

Dioses.java | DiosesGriegos.java | TestDiosesSemiDioses.java | TitanGriego.java

```
1  package patronesEstructurales.patronComposite;
2
3  import java.util.ArrayList;
4
5  public class TitanGriego implements Dioses {
6
7      private String nombreSemiDios;
8      private String hijodeDios;
9
10     private ArrayList<Dioses> dioses = new ArrayList<>();
11
12     TitanGriego(String nombreSemiDios,String hijoDios)
13     {
14         this.nombreSemiDios = nombreSemiDios;
15         this.hijodeDios = hijoDios;
16     }
17
18     public void agregarDios(Dioses dios)
19     {
20         dioses.add(dios);
21     }
22
23     @Override
24     public void mostrarDioses() {
25         System.out.println("Titan: "+nombreSemiDios+" Titan del: "+hijodeD
26         for(Dioses elementoDioses : dioses )
27         {
28             elementoDioses.mostrarDioses();
29         }
30     }
31 }
```

package patronesEstructurales.patronComposite;: Al igual que en las clases anteriores, esta línea especifica el paquete al que pertenece la clase TitanGriego.

import java.util.ArrayList;: Importa la clase ArrayList del paquete java.util, que se utiliza para crear una lista de elementos.

public class TitanGriego implements Dioses {: Aquí se define la clase TitanGriego, que implementa la interfaz Dioses. Como resultado, la clase debe proporcionar una implementación del método mostrarDioses() definido en la interfaz.

private String nombreSemiDios; y private String hijodeDios;: Estas líneas declaran dos campos (variables de instancia) privados en la clase TitanGriego. Uno es nombreSemiDios para almacenar el nombre del titán semidiós y el otro es hijodeDios para almacenar la descripción de su origen o área de responsabilidad.

private ArrayList<Dioses> dioses = new ArrayList<>();: Aquí se declara un campo llamado dioses, que es una lista de objetos que implementan la interfaz Dioses. Se inicializa como una nueva instancia de ArrayList, lo que significa que se utiliza para almacenar otras instancias de objetos que implementan la interfaz Dioses.

TitanGriego(String nombreSemiDios, String hijoDios) { ... }: En esta parte, se define el constructor de la clase TitanGriego que toma dos argumentos, nombreSemiDios y hijoDios, para inicializar los campos nombreSemiDios y hijodeDios al crear una instancia de esta clase.

public void agregarDios(Dioses dios) { ... }: Este método permite agregar un objeto que implementa la interfaz Dioses a la lista de dioses dentro de esta clase.

public void mostrarDioses() { ... }: Aquí se proporciona la implementación concreta del método mostrarDioses() requerido por la interfaz. En este caso, el método imprime en la consola el nombre del titán semidiós, su área de responsabilidad y luego recorre la lista de dioses para mostrar sus detalles llamando al método mostrarDioses() de cada uno de ellos.

Edison Wladimir Morocho Guayanlema
Ingenieria en sistemas de información
Sexto Nivel
Paralelo **A**
Ingenieria de software II

Dioses.java | DiosesGriegos.java | TestDiosesSemiDioses.java | TitanGriego.java

```java
package patronesEstructurales.patronComposite;

public class TestDiosesSemiDioses {

    public static void main(String[] args)
    {
        Dioses dios1 = new DiosesGriegos("Zeus","Rayo");
        Dioses dios2 = new DiosesGriegos("Ades","Inframundo");
        Dioses dios3 = new DiosesGriegos("Poseidon","Mares");

        TitanGriego titan = new TitanGriego("Kronos", "Tiempo");
        titan.agregarDios(dios1);
        titan.agregarDios(dios2);
        titan.agregarDios(dios3);

        Dioses dios4 = new DiosesGriegos("Apolo", "Sol");

        TitanGriego titan2 = new TitanGriego("Gaia", "Tierra");

        titan2.agregarDios(dios4);

        titan.agregarDios(titan2);

        titan.mostrarDioses();



    }

}
```

package patronesEstructurales.patronComposite;: Esta línea especifica el paquete al que pertenece la clase TestDiosesSemiDioses.
public static void main(String[] args) { ... }: El método main es el punto de entrada del programa.
Creación de instancias de dioses griegos (DiosesGriegos) y asignación a las variables dios1, dios2 y dios3. Cada uno de estos dioses se inicializa con un nombre y su área de responsabilidad (p.ej., Zeus como dios del rayo).
Creación de un titán griego (TitanGriego) llamado "Kronos" con el área de responsabilidad "Tiempo".
Agregación de los dioses dios1, dios2 y dios3 al titán titan utilizando el método agregarDios().
Creación de un nuevo dios griego (dios4) llamado "Apolo" con el área de responsabilidad "Sol".
Creación de otro titán griego (titan2) llamado "Gaia" con el área de responsabilidad "Tierra".
Agregación del dios dios4 al titán titan2.
Agregación del titán titan2 al titán titan, lo que permite crear una estructura jerárquica de dioses y titanes.
Llamada al método mostrarDioses() del titán titan para mostrar la jerarquía de dioses y titanes.