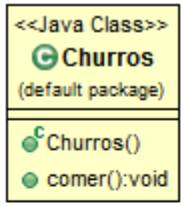
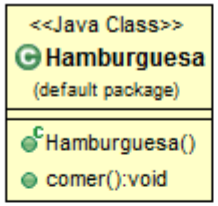
Patrones: Factory Method Abstract Factory

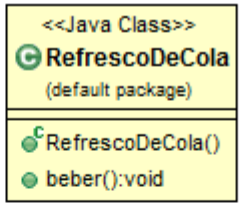
Ejemplo de uso

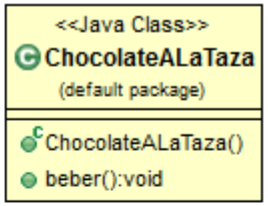
Merienda: comida + bebida

- Escenarios: Chocolatería y Hamburguesería



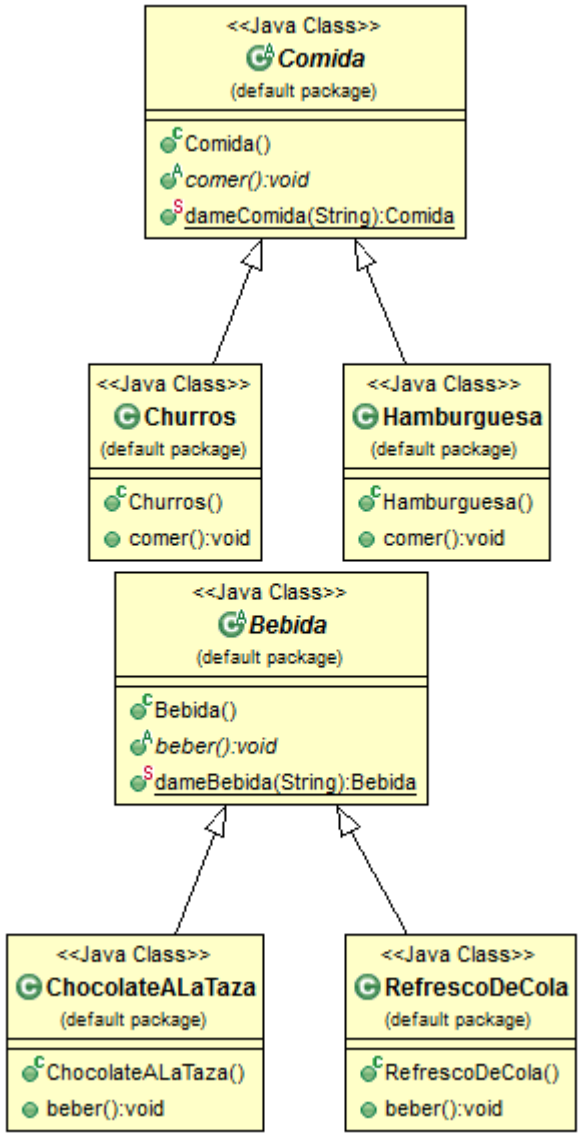







```java
public void crearMerienda(String tipoMerienda){
    /* Escogemos una comida y una bebida...."compatibles" */
    if (tipoMerienda=="Burguer"){
        // creamos los productos a consumir
        Hamburguesa c = new Hamburguesa();
        RefrescoDeCola b = new RefrescoDeCola();

        // los utilizamos
        c.comer();
        b.beber();
    } else
        if (tipoMerienda == "Cholateria"){
            // creamos los productos a consumir
            Churros c = new Churros();
            ChocolateALaTaza b = new ChocolateALaTaza();

            // los utilizamos
            c.comer();
            b.beber();
        }
}
```

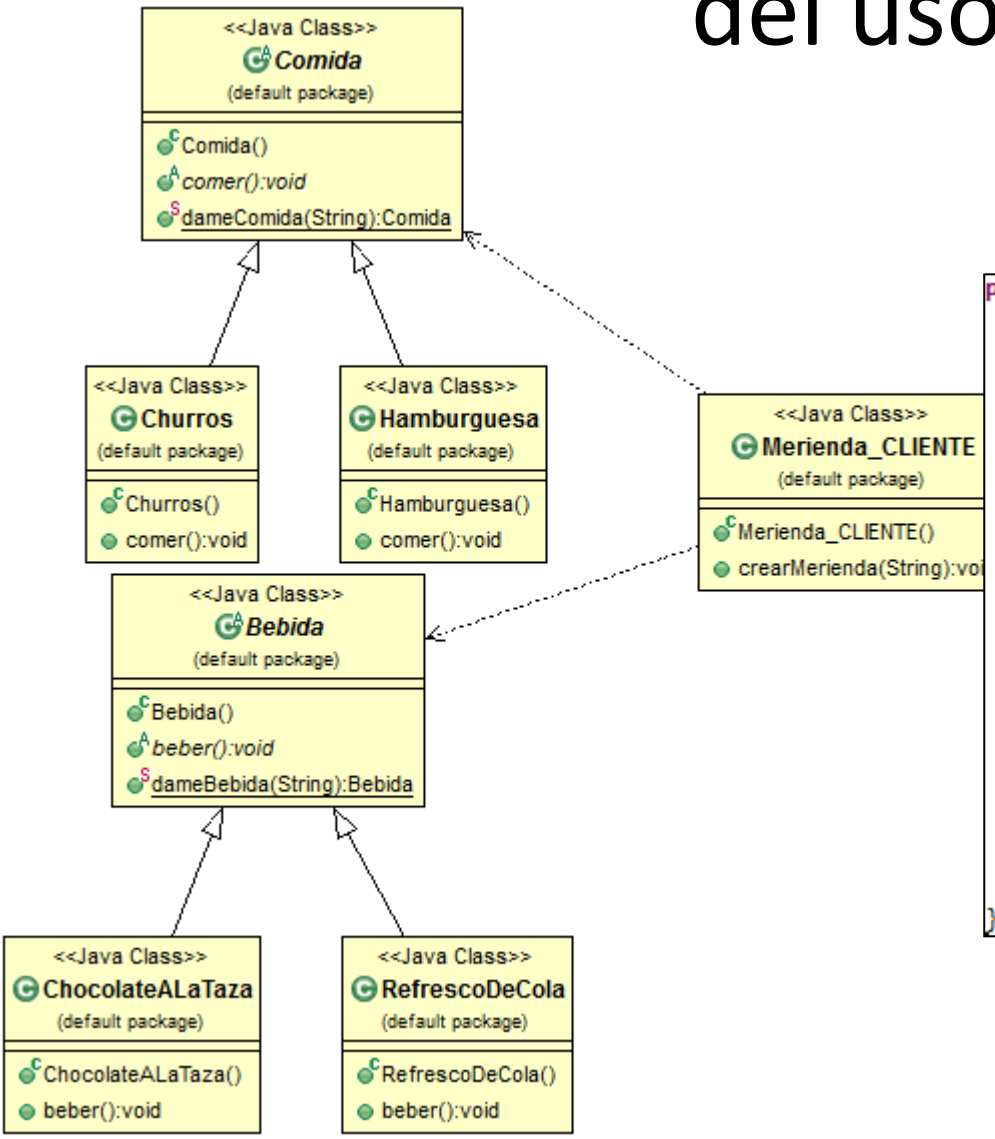
Factory Method: Separar la creación del uso de los “productos”




```
public abstract class Comida {
    public abstract void comer();

    public static Comida dameComida(String tipo){
        switch(tipo){
        case "Hamburgueseria":
            return new Hamburguesa();

        case "Chocolateria":
            return new Churros();

            /* NUEVAS COMIDAS AQUÍ*/
        default:
            return null;
        }
    }
}
```

```
public abstract class Bebida {
    public abstract void beber();
    public static Bebida dameBebida(String tipo){
        switch(tipo){
        case "Hamburgueseria":
            return new RefrescoDeCola();

        case "Chocolateria":
            return new ChocolateALaTaza();

            /* NUEVAS BEBIDAS AQUÍ*/
        default:
            return null;
        }
    }
}
```

Factory Method: Separar la creación del uso de los “productos”




```java
public class Merienda_CLIENTE {

    public void crearMerienda(String tipoMerienda){
        /* Escogemos una comida y una bebida...."compatibles" */

        //-------------------------------------------------------
        // creamos los productos a consumir

        Comida c = Comida.dameComida(tipoMerienda);
        //Antes: new Hamburguesa(); o new Churros()

        Bebida b = Bebida.dameBebida(tipoMerienda);
        //Antes: new RefrescoDeCola(); o new ChocolateALaTaza()


        //-------------------------------------------------------
        // los utilizamos
        c.comer();
        b.beber();

    }
}
```

Factory Method: Separar la creación del uso de los “productos”

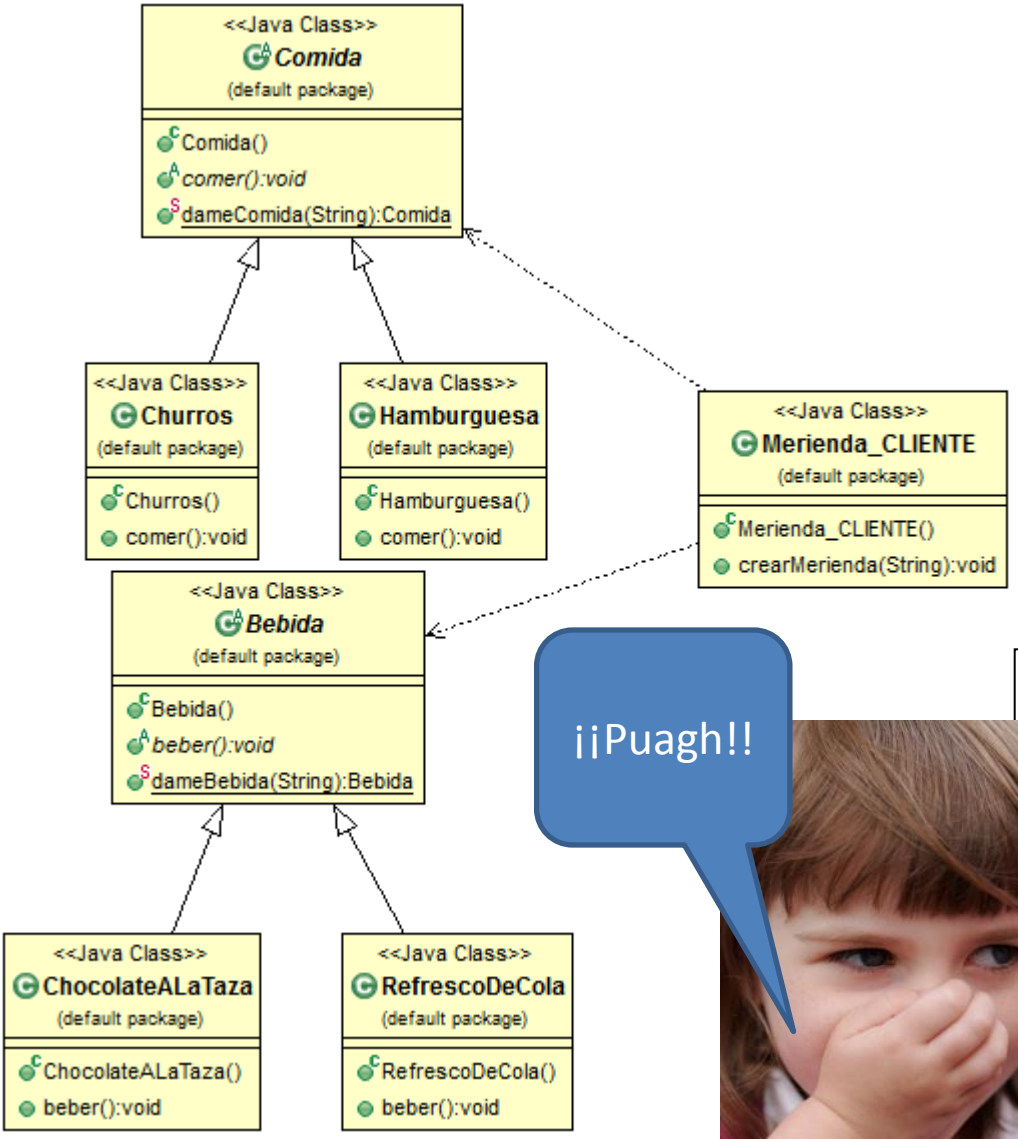
Sin Patrones

```
public void crearMerienda(String tipoMerienda){
    /* Escogemos una comida y una bebida...."compatibles"
    if (tipoMerienda=="Burguer"){
        // creamos los productos a consumir
        Hamburguesa c = new Hamburguesa();
        RefrescoDeCola b = new RefrescoDeCola();

        // los utilizamos
        c.comer();
        b.beber();
    } else
        if (tipoMerienda == "Cholateria"){
            // creamos los productos a consumir
            Churros c = new Churros();
            ChocolateALaTaza b = new ChocolateALaTaza();

            // los utilizamos
            c.comer();
            b.beber();
        }
}
```

Factory Method(s)

```
public class Merienda_CLIENTE {

    public void crearMerienda(String tipoMerienda){
        /* Escogemos una comida y una bebida...."compatibles" */

        //-------------------------------------------------------
        // creamos los productos a consumir

        Comida c = Comida.dameComida(tipoMerienda);
        //Antes: new Hamburguesa(); o new Churros()

        Bebida b = Bebida.dameBebida(tipoMerienda);
        //Antes: new RefrescoDeCola(); o new ChocolateALaTaza()


        //-------------------------------------------------------
        // los utilizamos
        c.comer();
        b.beber();

    }
}
```

Problema: ¿"productos" compatibles?



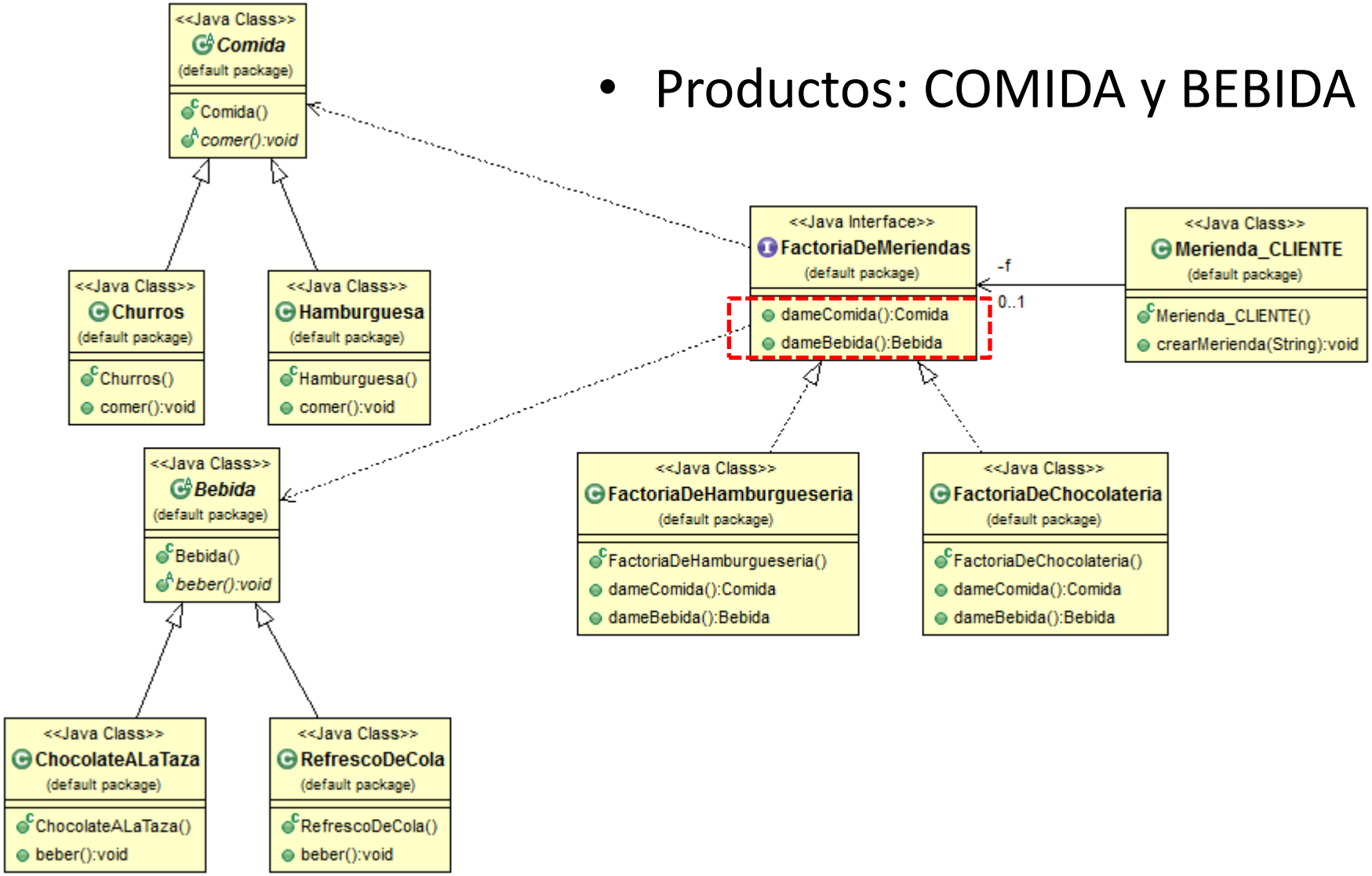
Sintácticamente ☑

Semánticamente ☒

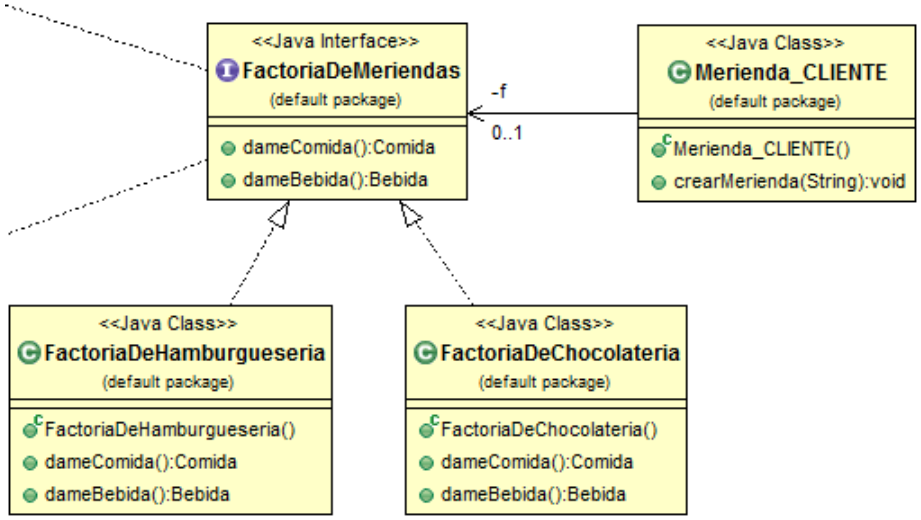

```
public void crearMeriendaConflictiva(){
    /* Escogemos una comida y una bebida....¿compatibles? */

    //-------------------------------------------------------
    // creamos los productos a consumir

    Comida c = Comida.dameComida("Hamburgueseria");

    Bebida b = Bebida.dameBebida("Chocolateria");


    //-------------------------------------------------------
    // los utilizamos
    c.comer();
    b.beber();
```

Solución: Abstract Factory

- Productos: COMIDA y BEBIDA



Solución: Abstract Factory




```java
public class FactoriaDeHamburgueseria implements FactoriaDeMeriendas {

    @Override
    public Comida dameComida() {

        return new Hamburguesa();
    }

    @Override
    public Bebida dameBebida() {

        return new RefrescoDeCola();
    }

}
```

```java
public class Merienda_CLIENTE {

    private FactoriaDeMeriendas f = null;

    Merienda_CLIENTE(String tipoMerienda){
        /* Escogemos una comida y una bebida...."compatibles" */

        //-------------------------------------------------------
        // SELECCIONAMOS la FAMILIA de productos a consumir

        if (tipoMerienda=="Hamburgueseria")
            f = new FactoriaDeHamburgueseria();

        if (tipoMerienda=="Chocolateria")
            f = new FactoriaDeChocolateria();
    }

    public void crearMerienda(){
        //-------------------------------------------------------
        // Creamos los productos a consumir COMPATIBLES

        Comida c = f.dameComida();
        Bebida b = f.dameBebida();

        //-------------------------------------------------------
        // los utilizamos
        c.comer();
        b.beber();

    }
}
```