Ministerio de Producción
Presidencia de la Nación

Ministerio de Educación y Deportes

Subsecretaría de Servicios Tecnológicos y Productivos

Programa 111 mil
VOS PODÉS SER UNO.

Especificaciones de acceso

Objetivo

- Evitar que los desarrolladores cometan errores llamando métodos que no deben.
- En caso de que generemos un error involuntario, podemos reducir el lugar de búsqueda de los errores.
- Regular el acceso desde ciertas partes de nuestro proyecto (clases en una herencia, clases del mismo paquete, etc.).
- Por ejemplo, queremos limitar el acceso a un método utilitario que modifica atributos internos.

Pizzeria: Todo público

```
public class Pizzeria {
    public int pizzasVendidas = 0;
    public ArrayList<Pedido> pedidos = new ArrayList<>();

    public void comprarPizza(Cliente cliente,
                             String tipoPizza){
        contarPizzaVendida();
        pedidos.add(new Pedido(cliente, saborPizza));
    }

    public void contarPizzaVendida(){
        pizzasVendidas = pizzasVendidas + 1;
    }
}
```

¿Es un problema?

- **Es un posible foco de errores de código.** El problema se agrava si trabajamos con un grupo de personas.
- Siguiendo el ejemplo, ¿qué sucede si alguien (incluso nosotros, los creadores de la clase Pizzería) utiliza el método `contarPizzaVendida` o modifica la lista de pedidos directamente?
- ¡Puede que el contador de pizzas vendidas tenga un valor que no sea acorde a los pedidos procesados!
- **Se crean inconsistencias.**

Ejemplo

```
Pizzeria pizzeria = new Pizzeria();

pizzeria.contarPizzaVendida(); // Vendidas => 1

pizzeria.comprarPizza(cliente, “muzzarella”); // Vendidas => 2,
                                  // está mal! en realidad vendimos 1.

pizzeria.pedidos.removeAt(0);
```

- En este caso, no existen errores de compilación ni de ejecución, simplemente, alguien utiliza de forma errónea el código.

Posible Solución: Modificadores de acceso

- Permiten restringir el acceso a cierta funcionalidad y atributos.
- Según la palabra reservada que utilicemos, el acceso se restringe en diferentes niveles. **public** es el modificador más permisivo.

Modificador de Acceso	Misma Clase	Mismo Paquete	Subclase	Otros Paquetes
public	Sí	Sí	Sí	Sí
protected	Sí	Sí	Sí	No
no access modifier	Sí	Sí	No	No
private	Sí	No	No	No

Mejorando la Pizzería

```
public class Pizzeria {
    private int pizzasVendidas = 0;
    private ArrayList<Pedido> pedidos =
                                new ArrayList<>();

    public void comprarPizza(Cliente cliente,
                             String tipoPizza){
        contarPizzaVendida();
        pedidos.add(new Pedido(cliente, saborPizza));
    }

    private void contarPizzaVendida(){
        pizzasVendidas = pizzasVendidas + 1;
    }
}
```

Ejemplo con nuevos modificadores

```
Pizzeria pizzeria = new Pizzeria();

pizzeria.contarPizzaVendida(); // Error de compilación!

pizzeria.comprarPizza(cliente, “muzzarella”);

pizzeria.pedidos.removeAt(0); // Error de compilación!
```

- El IDE nos indica que nuestro código tiene errores y no nos deja compilar y, por lo tanto, tampoco ejecutar la aplicación.
- Tanto la lista de pedidos (un atributo) y contarPizzaVendida (un método), solo se pueden utilizar dentro de la clase Pizzeria y no desde afuera.
- Si este código se hallara dentro de la clase Pizzería, no abría errores.

Su uso en la declaración de una clase

- En la declaración de la clase se puede utilizar:
  a. Nada: sin modificador de acceso se limita el uso de clase al paquete actual.
  b. public: La clase se puede usar desde cualquier lugar del proyecto.
- Al igual que los métodos, podemos “ocultar” clases que no queremos que sean utilizadas en otras partes de nuestro proyecto Java.

protected: un caso especial

- **protected** es similar a public, pero permite ocultar un método o atributo hacia el exterior de un paquete.
- El objetivo es permitir que un “hijo” de nuestra clase pueda llamar o pisar métodos para extender cierta funcionalidad.

Ejemplo

```
public class Pizzeria {
    protected Pizza cocinarPizza(Masa m, Ingredientes i){
        Prepizza prepizza = m.amasar();
        prepizza.colocarIngredientes(i);
        return prepizza.hornear();
    }

    public Pedido getPedido(){
        Pizza pizza = cocinarPizza(this.masa, this.ingredientes);
        return new Pedido(pizza);
    }
}
```

Ejemplo

```
public class PizzeriaEspecial extends Pizzeria{
    protected Pizza cocinarPizza(Masa m, Ingredientes i) {
        Prepizza prepizza = m.agregarIngredientes(i);
        prepizza.agregarIngredientes(ingredientesEspeciales);
        return prepizza.hornear();
    }
}
```