**OOP: Encapsulation in Java**



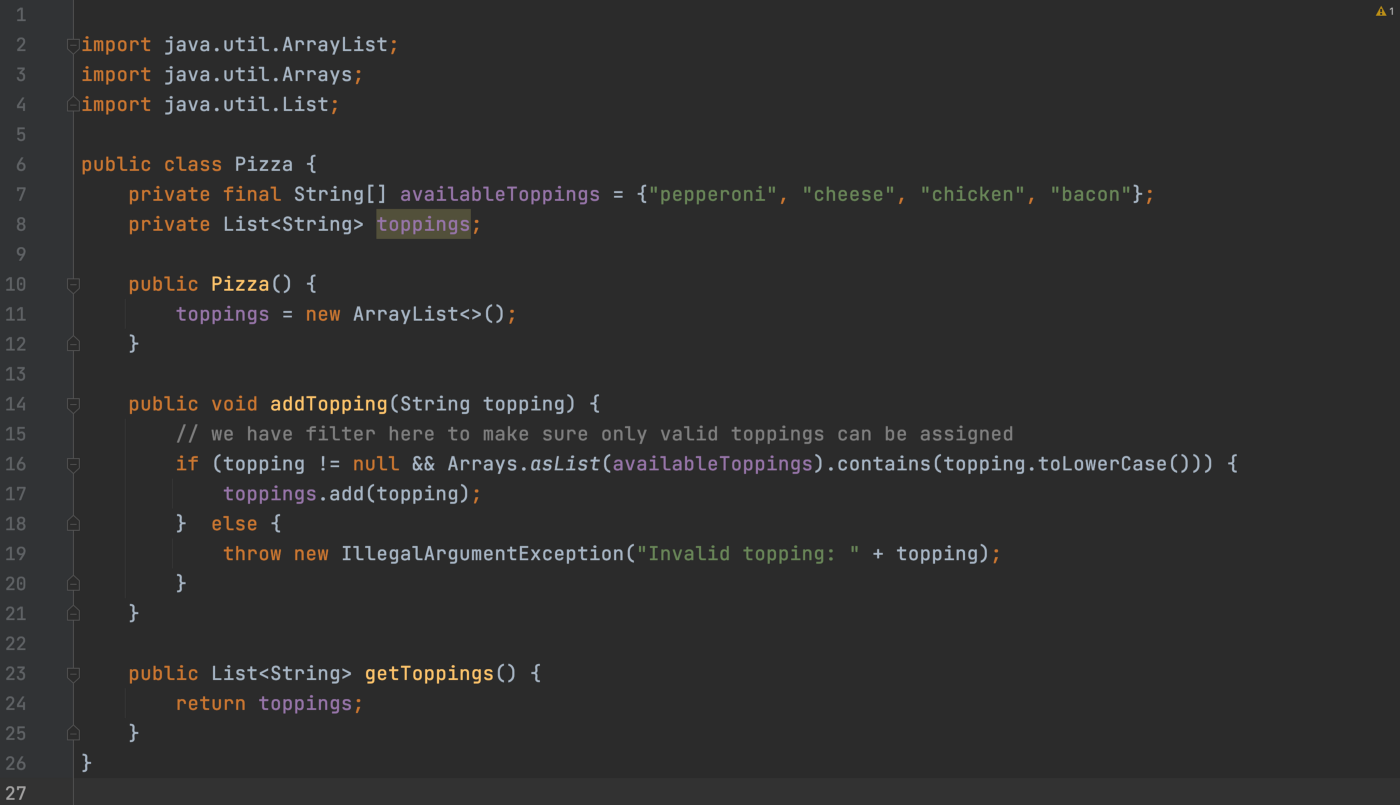
in Person.java file

Encapsulation is a data hiding mechanism. We achieve encapsulation by making our variables private and providing public getters and setters.

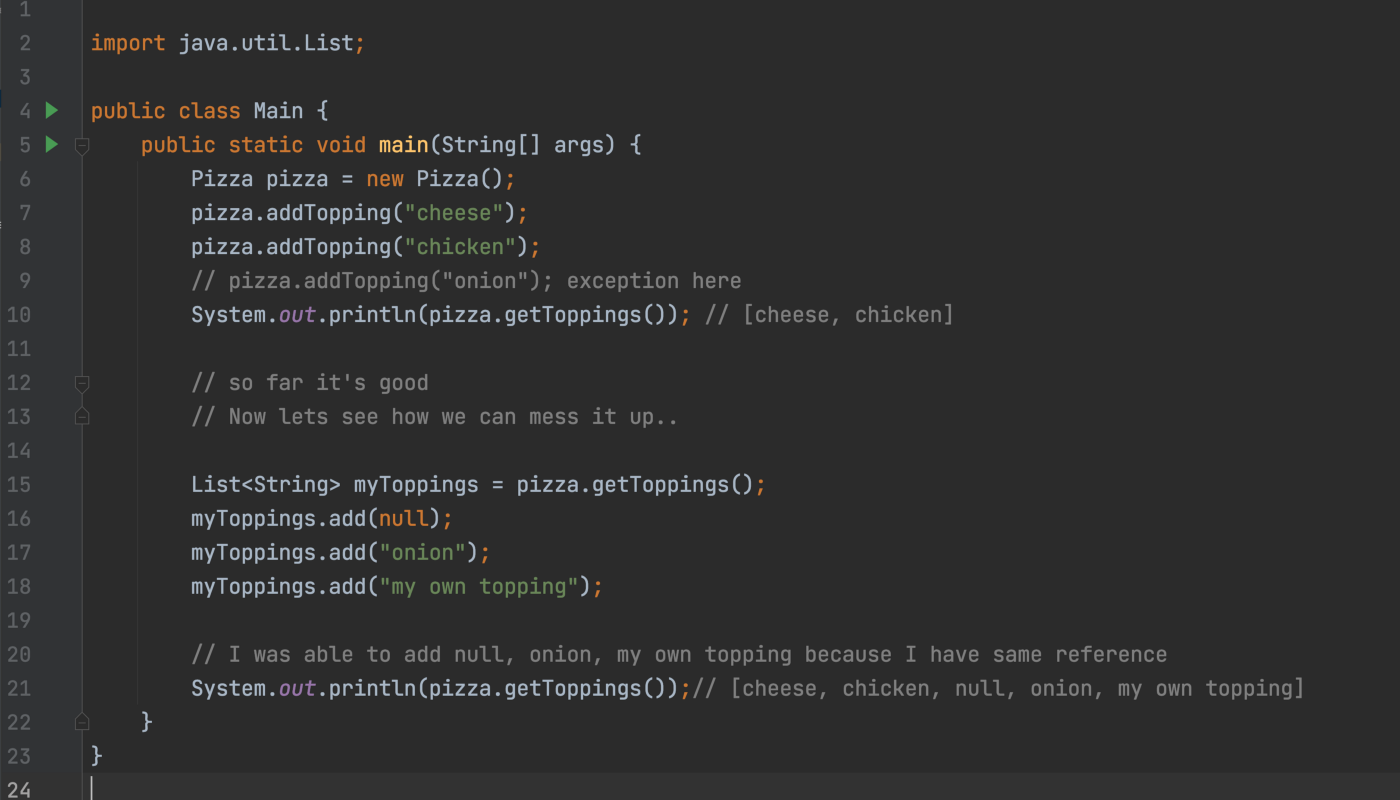
**Why do we need to encapsulate?** If the property has public access, the client code can have direct access and can assign any value. By encapsulating we have one layer where we can control what comes to our property in our setter method. In the above example, we can see how we are restricting negative age in the setAge method. Another example — let’s say we are creating a custom List data structure based on an array. The underlining array data structure should be private if we will make it public, it will be accessible to client code. There are possibilities that client code will create a big mess by manipulating directly with the array.

**From whom we are protecting our data?** Seems a very simple question however understanding this question is the main part to start developing using the encapsulation concept. We are removing direct access from object properties so we are protecting it from client code that will use this class(object). We are hiding properties from ourselves if we will use this class in other parts of the project.

If the property type of object is a mutable object. We cannot return the original address because the client will direct access using the returned reference. We always need to take a copy and return a reference to it.



The main idea of the above encapsulation to make sure only available toppings can be assigned to our toppings list. But there is an issue with this code. It returns toppings directly. It’s a reference so using this reference client code could change toppings directly.



We were able to add toppings that are not part of the available toppings. Even we could add null. So we want to achieve incapsulation correctly we never return references directly instead we return their copies.

For example, we can fix our Pizza class by changing getToppings method

public List<String> getToppings() {  
 return new ArrayList<>(toppings); // return copy  
}

Even though you can create your setters and getters methods with any name or if the requirement is not required to have them, it is totally fine to avoid them. But keep in mind that if you want to use your objects with external libraries, they might assume you have all setters and getters with correct names.