**Prototype Pattern**

**Overview**

What would you do when you need lot of objects of same type to be created at runtime ? The usual problem with object creation is it gets instantiated and has to get some data back and forth from database. To avoid this problem, what you could do is, create object at runtime. Such as, if you got Animal arraylist with bunch of animals such as cat, dog etc. you could create of same type during runtime without having to instantiate again and again and going to database can be tedious. This is what prototype pattern is. It copies object of same type when you need without much intensive operation through cloning of existing object.

**Benefits**

* Object is created at runtime so there is no need to have communication to database or instantiation.
* Object can be created and disposed during runtime
* Makes program run faster and less objects are created so code can be made simplified and clean.

**Problem/Scenario**

As an example scenario, suppose a Bottleshop sells 100s of beers. One day bottleshop opens it’s sister shopfront in another location and wants to make copy of beers from one to another location maybe online. It is going to display exact same beers with same prices in different location. Would you bring those beer from database or clone from existing Bottleshop store in original location? Answer is prototype design pattern can be used to create list of beers with prices in different location. In this example, I have only used Corona beer as an example, but it can be of all beers using Arraylist where you can create beers.

With normal way of creating objects usually demands for inheritance, new object creation in runtime which needs data fetched from backend to display, beer name, price, and alcohol level. Instead of old way of subclassing objects and telling user prices, names, we want to implement prototype design pattern which uses object factory to clone objects from existing beer objects on one store to another. This is entire concept of prototype pattern which helps save memory in runtime and avoids having to read from database.

**Before Refactor**

Initially, beers would need to be created using new keyword and would inheritance from one beer to another beer. This, creates problem that beers’ data will need transfer to database and fetched back when needed. We do not want that extensive operation. It is a tedius task of creating beer and storing prices of those, and maybe names, alcohol percents etc.

**After Refactor**

After refactor all beers data can be brought from parent website where all beers listed are also listed by child Bottleshop by cloning objects listed. It avoids inheritances, and uses ObjectFactory instead to use a method that has been used from interface through original beer creator method which is in a class called CoronaBeer. If multiple beers are needed we could just use ArrayList, List and create a bunch of them using this pattern.

Before/after link

https://github.com/sppanday/S120-PRT583-Group-A/tree/feat/design-pattern/s260598-PandaySurendra/Sprint-4-Deliverables

**UML Diagram**

Please find it inside relevant project UML in pattern folder

/PrototypePattern.png

**Reference**

Freeman Eric., Freeman Elisa. 2004. 1st ed. O’Reily Media Inc. 1995, Sebasttopo, CA