Faculty of Computers, Informatics and Microelectronics

Technical University of Moldova

IPP

Lab#1

*Author:*

Mereuta Alex

*Supervisor:*

Anastasia SERSUN

Chisinau 2017

1. Objective

Creational Patterns

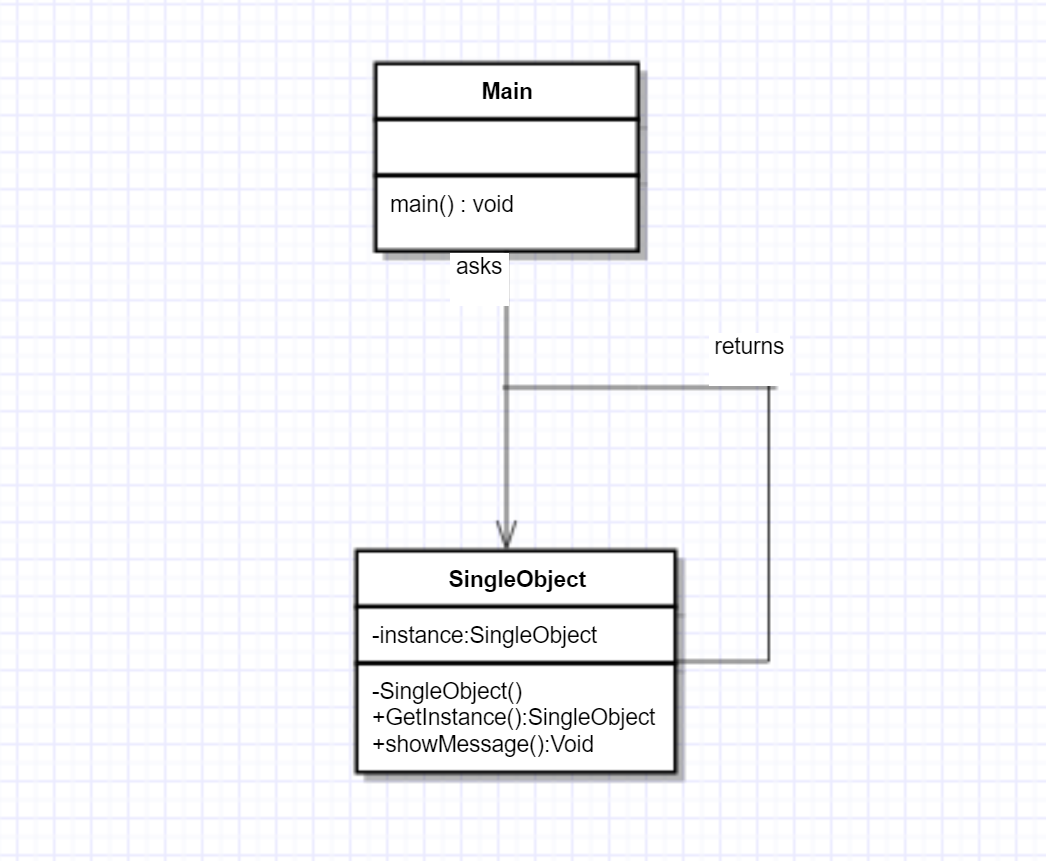
Implementing Creational Patterns:

Builder, Factory, Abstract Factory, Singleton, Prototype

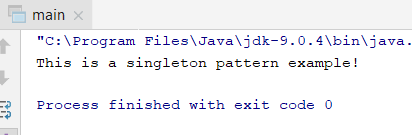
Singleton Pattern

* Ensure a class has only one instance, and provide a global point of access to it.
* Encapsulated "just-in-time initialization" or "initialization on first use".

Make the class of the single instance responsible for access and "initialization on first use". The single instance is a private static attribute. The accessor function is a public static method



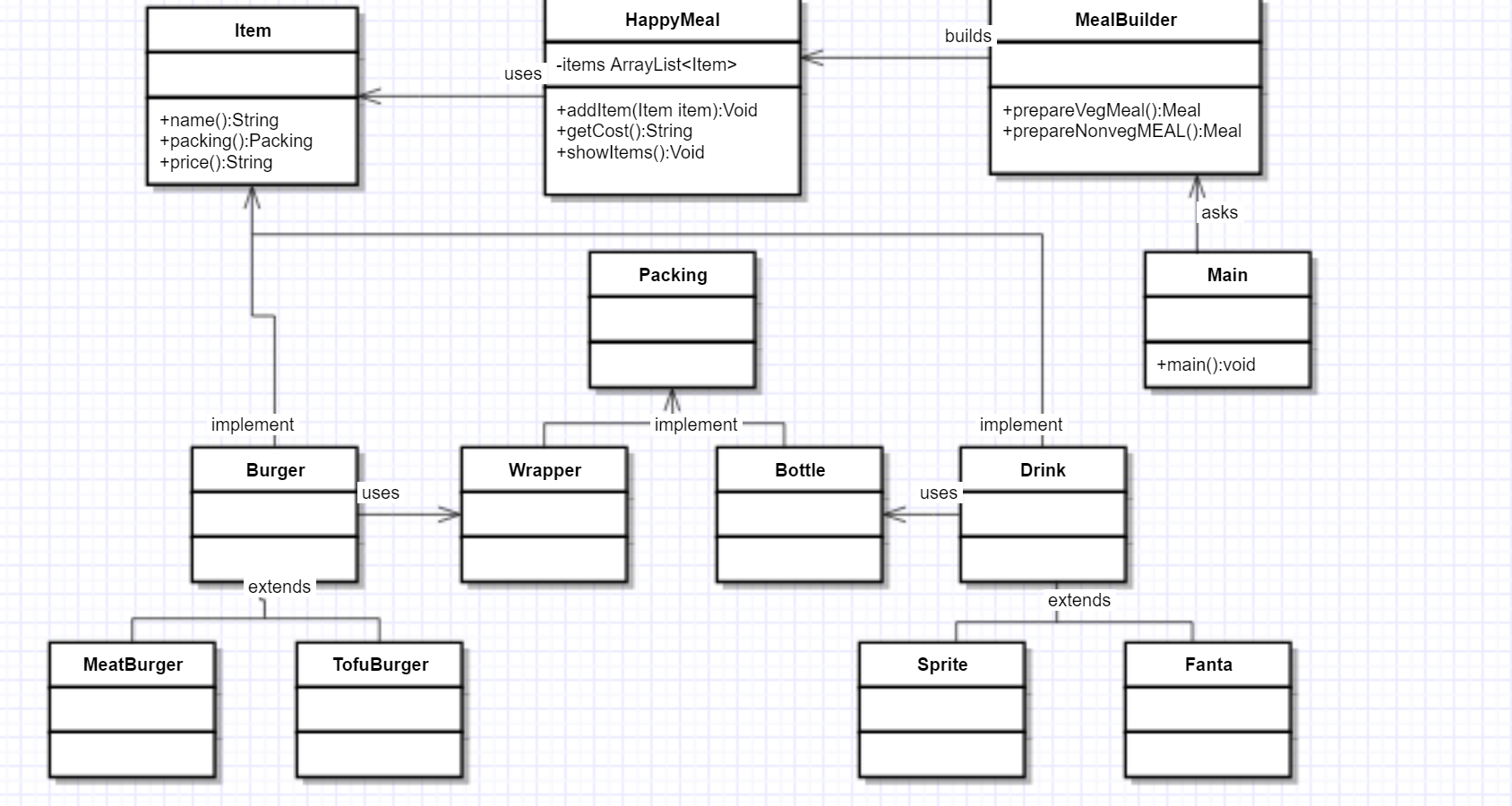
In my case I have a SingleObject class which has its constructor as private and a static instance of itself. Main class uses SingleOject to get a SingleObject.



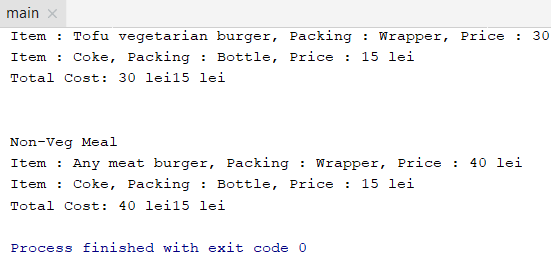
Builder Pattern

Builder pattern builds a complex object using simple objects and using a step by step approach. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

A Builder class builds the final object step by step. This builder is independent of other objects.



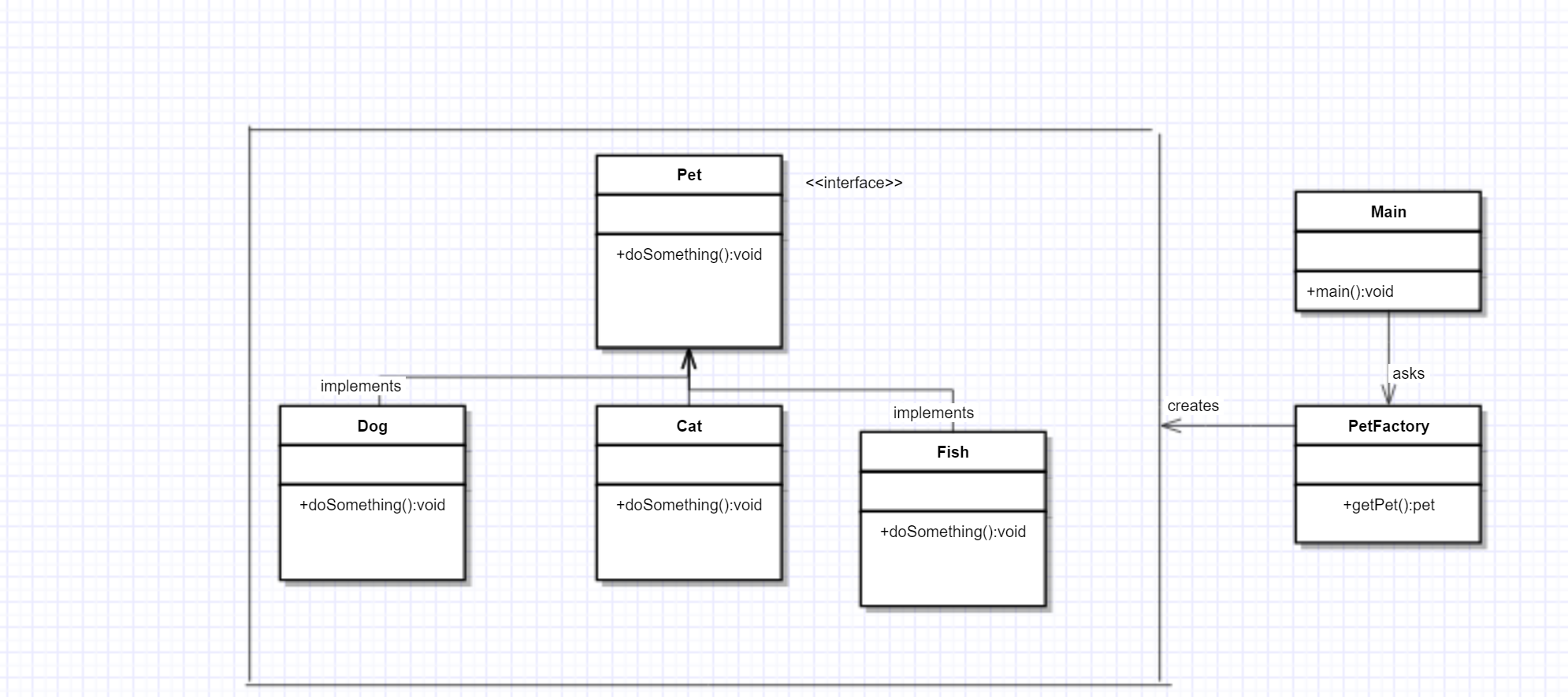
In my case I had a fastfood place with the possibility to order a burger and a drink. Burger could be either a meaty one or a tofu one. Same with the drink, there’s a sprite and a fanta option. Food is packed in a wrapper and drink is packed in a bottle. The Item interface represents food items (respectively concrete classes implementing item regarding food) and theres also a packing interface representing the type of package. Then there’s the HappyMeal class which creates an ArrayList of Items and the MealBuilder actually builds different types of HappyMeal depending on Item. Main class will use MealBuilder in order to build a Meal.



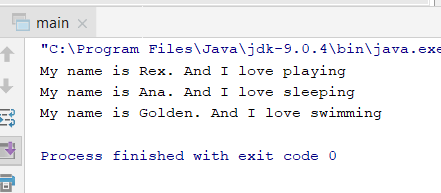
Factory Pattern

Factory pattern is one of the most used design patterns in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

In Factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface.



Here I have a Pet interface and concrete classes like dog, cat and fish to implement its method. The Main class will use PetFactory class to get a Pet object. So it will pass the info(cat,dog,fish) to petfactory regarding to the requested pet.



Prototype Pattern

Prototype pattern refers to creating duplicate object while keeping performance in mind. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

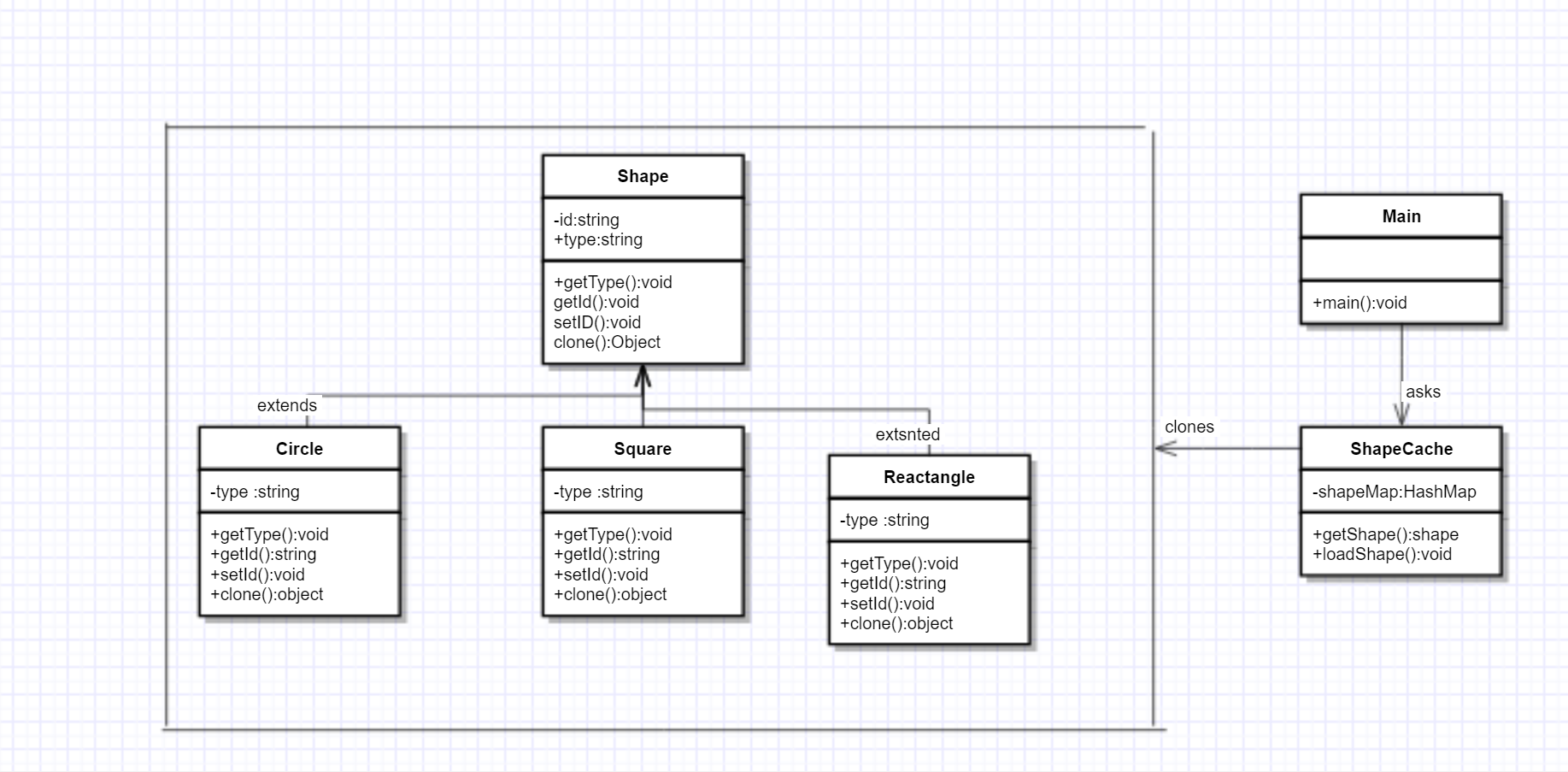
This pattern involves implementing a prototype interface which tells to create a clone of the current object. This pattern is used when creation of object directly is costly. For example, an object is to be created after a costly database operation. We can cache the object, returns its clone on next request and update the database as and when needed thus reducing database calls.

Here I used a hashtable and since I’m new to java I had to read more about it.

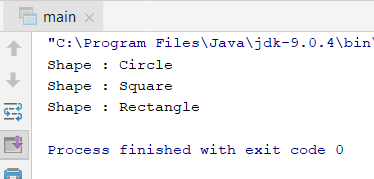
This class implements a hash table, which maps keys to values. Any non-null object can be used as a key or as a value.

To successfully store and retrieve objects from a hashtable, the objects used as keys must implement the hashCode method and the equals method.

*shapeMap.put(shapeKey, shape);*



Here I have an abstract class Shape and also concrete classes extending shape such as circle, rectangle and square. The class ShapeCache stores shape objects in a Hashtable and returns their clone when requested. Main class will use the ShapeCache class to get a shape object.



Conclusion:

Doing this practical laboratory work I got the chance to learn more about Design patterns, where to use them, when and how to use them. It’s different to read theory and to apply it in practice. By studying the class diagrams it was easier to understand how they are constructed. The first 5, creational patterns were easy to understand. Especially Builder, Singleton and Factory. Prototype and Abstract Factory are a bit more complex. In the end, these patterns are there to simplify our daily coding life. Code has to be reusable and easy to understand.