Faculty of Computers, Informatics and Microelectronics

Technical University of Moldova

APPOO

Laboratory work#3

Class diagrams. Programming patterns.

Author:
Denis POSTICA

Supervisor:
Anastasia SERSUN

May 26, 2017

Laboratory work #3

1 Purpose of the laboratory work

During this laboratory work you will need to pick 2 different patterns (not from the same category) from the next list and implement them in your code. In order to show how they work, please create a short program.

For all classes that you create you must provide a class diagram (using UML notation).

- Creational Patterns - provide a way to create objects while hiding the creation logic, rather than instantiating objects directly using new operator. This gives program more flexibility in deciding which objects need to be created for a given use case.
  - Abstract Factory -Creates an instance of several families of classes.
  - Builder - Separates object construction from its representation.
  - Factory Method -Creates an instance of several derived classes.
  - Object Pool - Avoid expensive acquisition and release of resources by recycling objects that are no longer in use.
  - Prototype - A fully initialized instance to be copied or cloned.
- Structural Patterns - by using inheritance those patterns are used to compose interfaces and define ways to compose objects in order to obtain new functionalities.
  - Adapter - Match interfaces of different classes.
  - Bridge - Separates an object's interface from its implementation.
  - Composite - A tree structure of simple and composite objects.
  - Decorator - Add responsibilities to objects dynamically.
  - Facade - A single class that represents an entire subsystem.
  - Flyweight - A fine-grained instance used for efficient sharing.
  - Private Class Data - Restricts accessor/mutator access.
  - Proxy - An object representing another object.
- Behavioral Patterns - specifically concerned with communication between objects Chain of responsibility - A way of passing a request between a chain of objects.
  - Chain of responsibility - A way of passing a request between a chain of objects
  - Command - Encapsulate a command request as an object
  - Iterator - Sequentially access the elements of a collection
  - Memento - Capture and restore an object's internal state
  - Observer - A way of notifying change to a number of classes
  - State - Alter an object's behavior when its state changes

– Template method - Defer the exact steps of an algorithm to a subclass
– Visitor - Defines a new operation to a class without change

2 Laboratory work implementation

I chose to exemplify through implementation the Builder and Chain of Responsibility patterns . The code of the program can be found at Git repository `https://github.com/PosticaDenis/APPOO`.

Laboratory work analysis

CHAIN OF RESPONSIBILITY

As the name suggests, the chain of responsibility pattern creates a chain of receiver objects for a request. This pattern decouples sender and receiver of a request based on type of request. This pattern comes under behavioral patterns.

In this pattern, normally each receiver contains reference to another receiver. If one object cannot handle the request then it passes the same to the next receiver and so on.

I have considered the case when is necessary to deal with some weight and in case the weight is greater than the max weight a class can handle, it is transmitted to the successor.

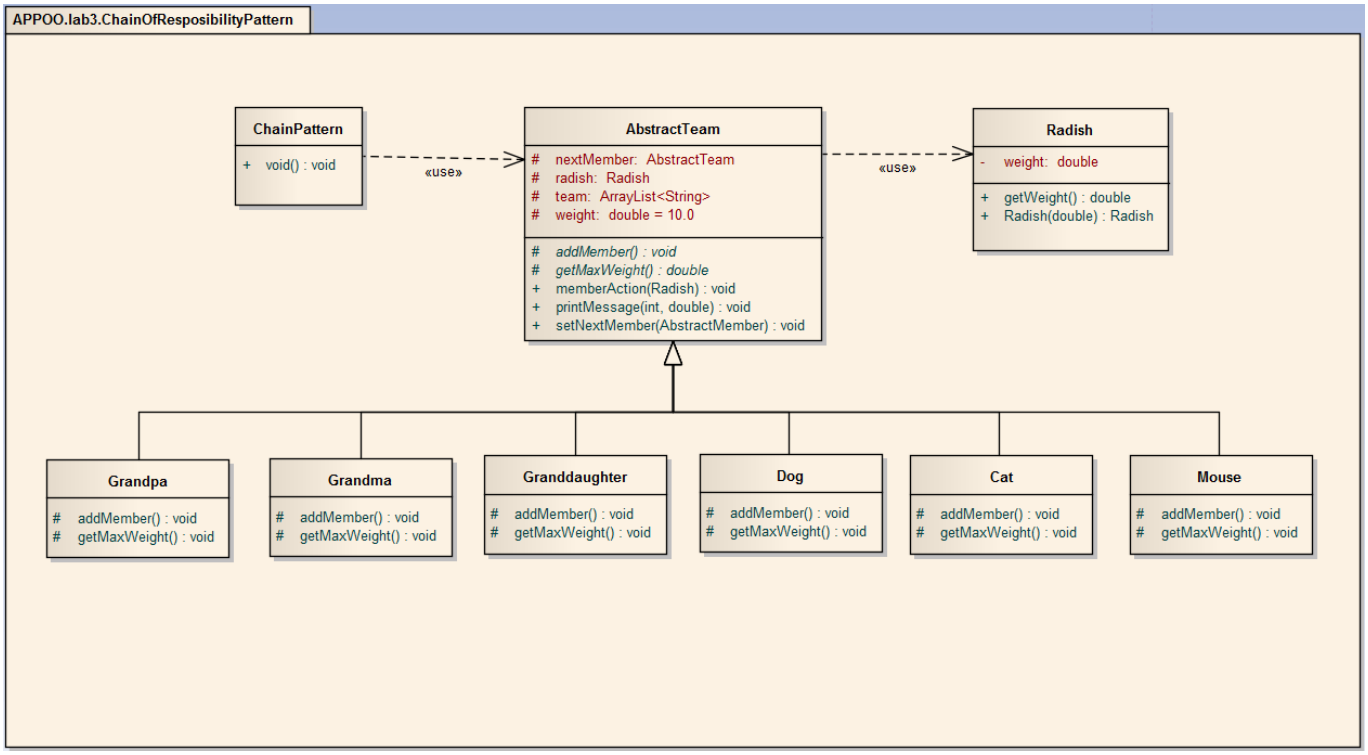

Figure 2.1– Class Diagram in the case of Chain of Responsibility pattern.

BUILDER

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.

I have considered a case of a clothes shop, where a piece of clothing might be a pair of pants or a shirt. Pants could be either classical or casual looking and having blue color. Shirts could be

either classical or casual looking and having black color.

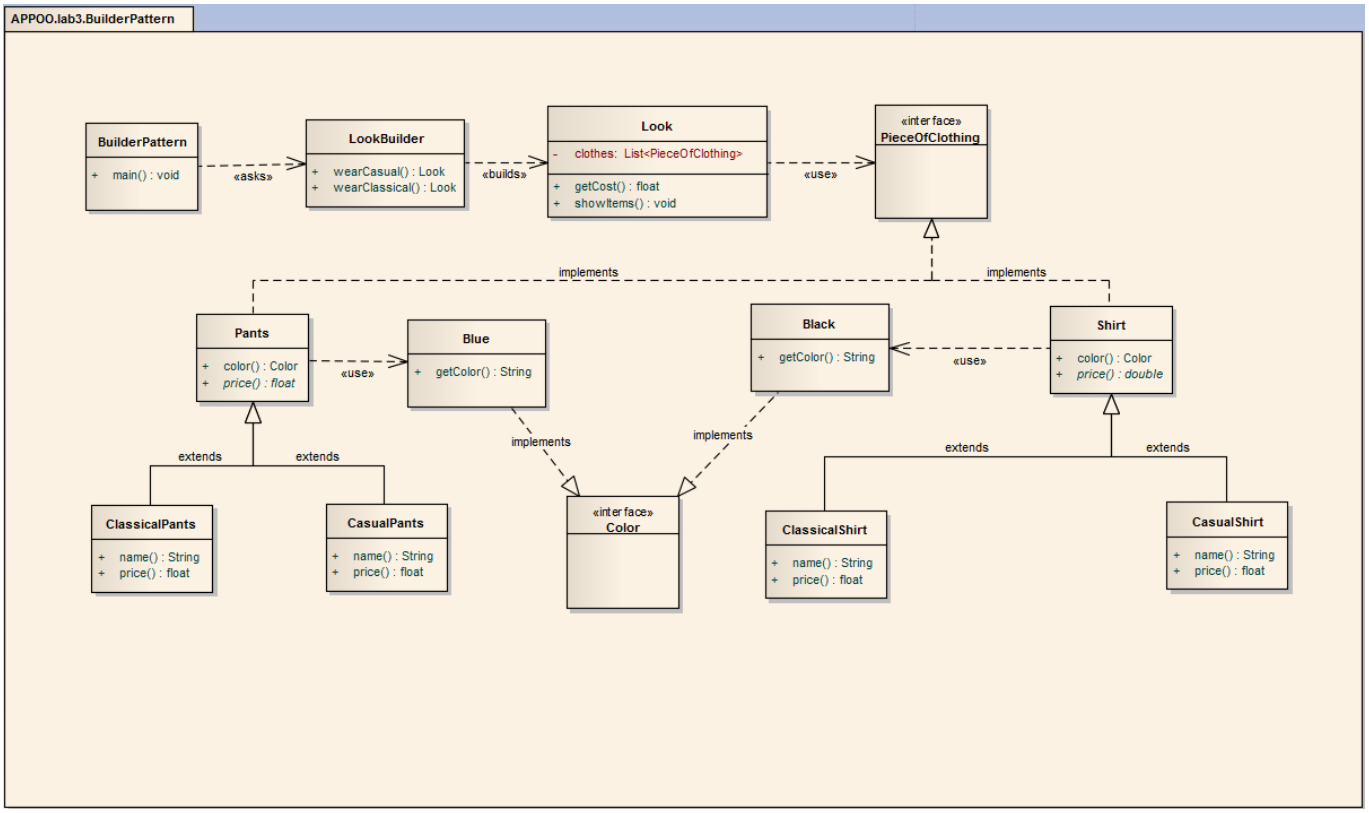

Figure 2.2– Class Diagram in the case of Builder pattern.

Conclusions

Design patterns represent the best practices used by experienced object-oriented software developers. Design patterns are solutions to general problems that software developers faced during software development. These solutions were obtained by trial and error by numerous software developers over quite a substantial period of time.

References

1 Chain of Responsibility Pattern, https://www.tutorialspoint.com/design_pattern/chain_of_responsibility_pattern.htm

2 Design Patterns - Builder Pattern, https://www.tutorialspoint.com/design_pattern/builder_pattern.htm

3 Guide for laboratory work 3.