Java
Lecture 9 – Design Patterns

IT Learning &
Outsourcing Center

Lector: Peter Manolov
Skype: nsghost1
E-mail: p.manolov@gmail.com

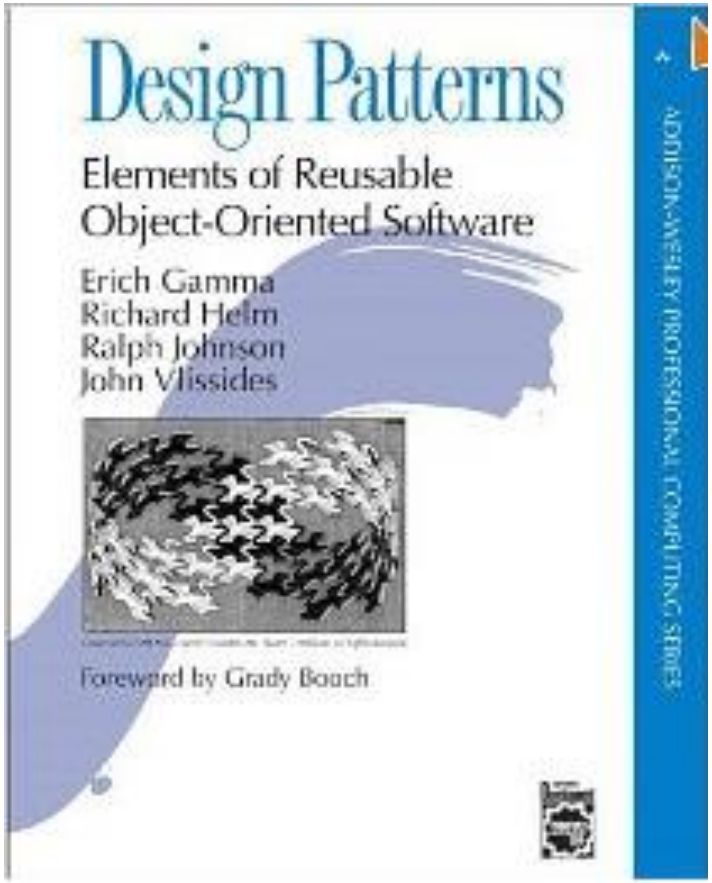
www.pragmatic.bg

2013 – 2014

Summary

- What are design patterns
- MVC pattern
- Classification of design patterns
  - Creational
  - Structural
  - Behavioral

Book to read

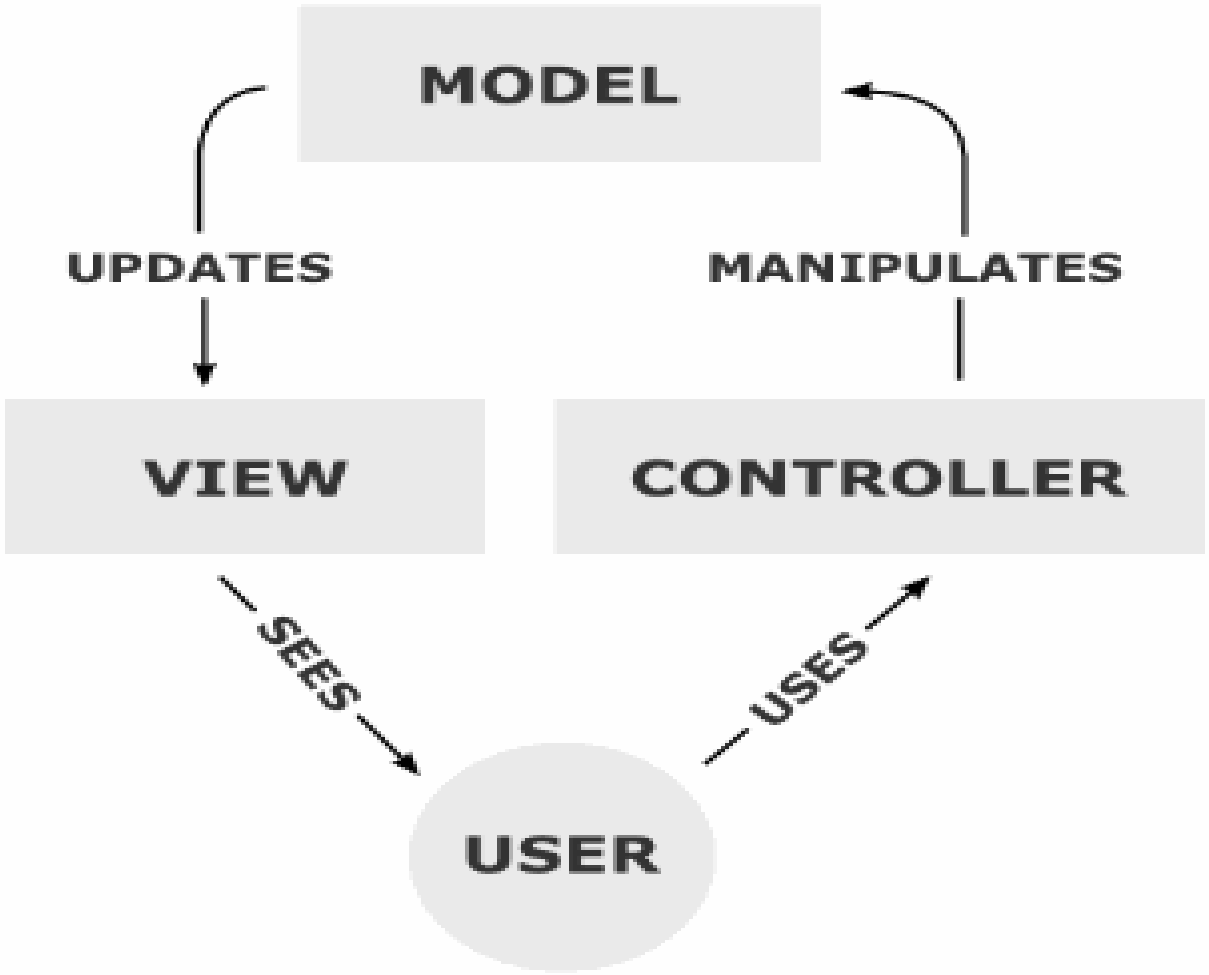


Design patterns elements of reusable
Object–Oriented Software
- Gang of Four

What are design patterns?

- A design patterns is a problem which occurs in our day to day lives and a solution to that problem. The solution is done in such a way that it will allow the beneficiary of that solution to implement it in way that it best suits the situation at hand.
- Design patterns are a good way to help team members quickly communicate a solutions to one another
- Design patterns are not “panacea”. Each one has pros and cons.

MVC Pattern



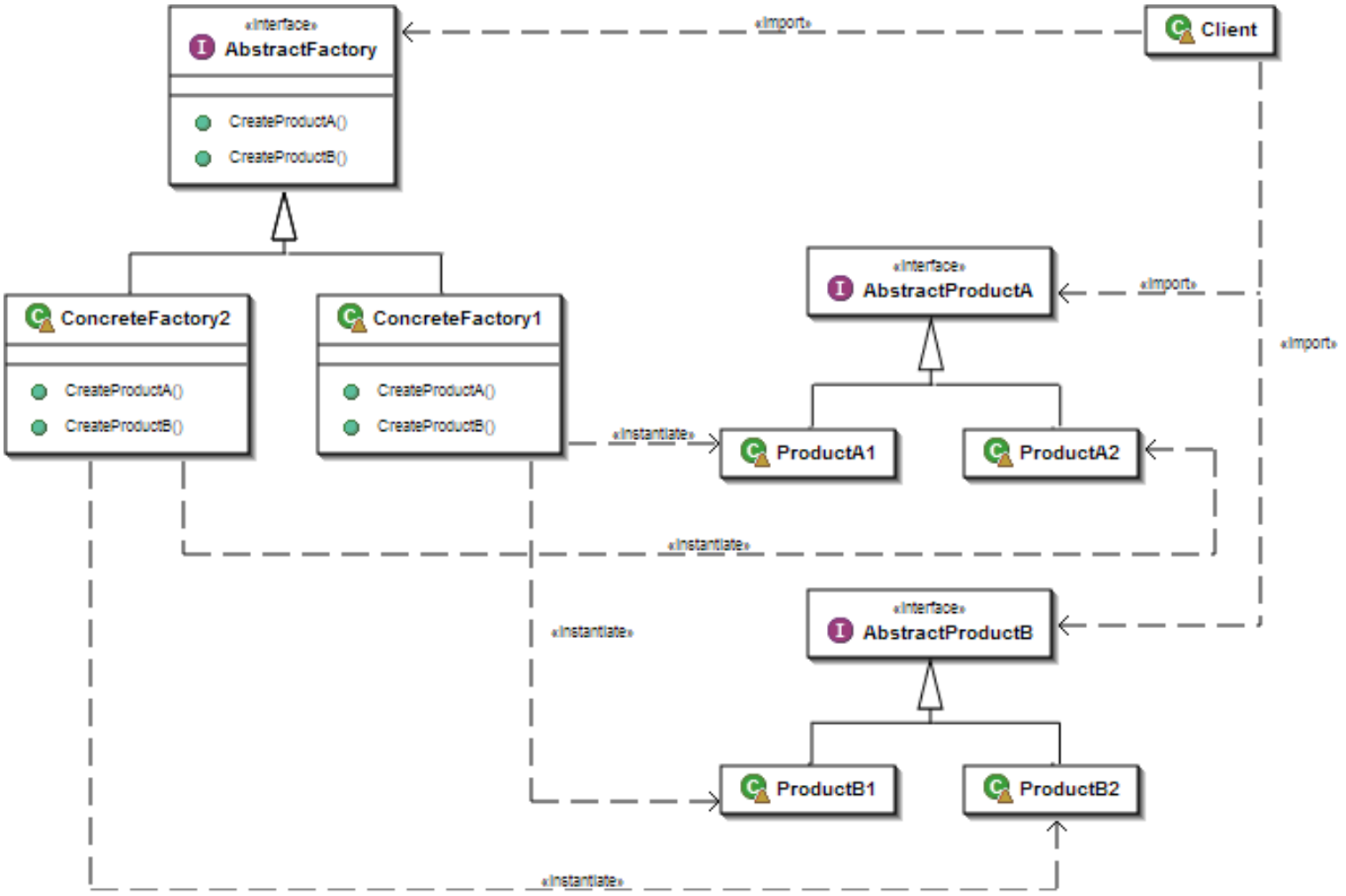
MVC Pattern

- A design pattern widely used to create user interface
- A pattern that separates the an application in three layers
- The application data (model) is separated from the way the that data is presented to the user (view) and it is manipulated or accepted by the user (controller)
- Separating the different layers allows application to be more flexible with changes that takes place within a live system. (new requirements from the clients)

Creational Patterns

- Creational design patterns abstract the instantiation process
- They hide away the manner in which objects are created, composed and represented.
- Creational patterns allow better flexibility of the system by hiding away object creation as the system grows and requirements change they allow easier introduction of behavior by composition rather than inheritance.
- Two important features of those classes
  - Hides away the actual classes that a system uses
  - Hides away how the system creates those classes and puts them together

Abstract Factory



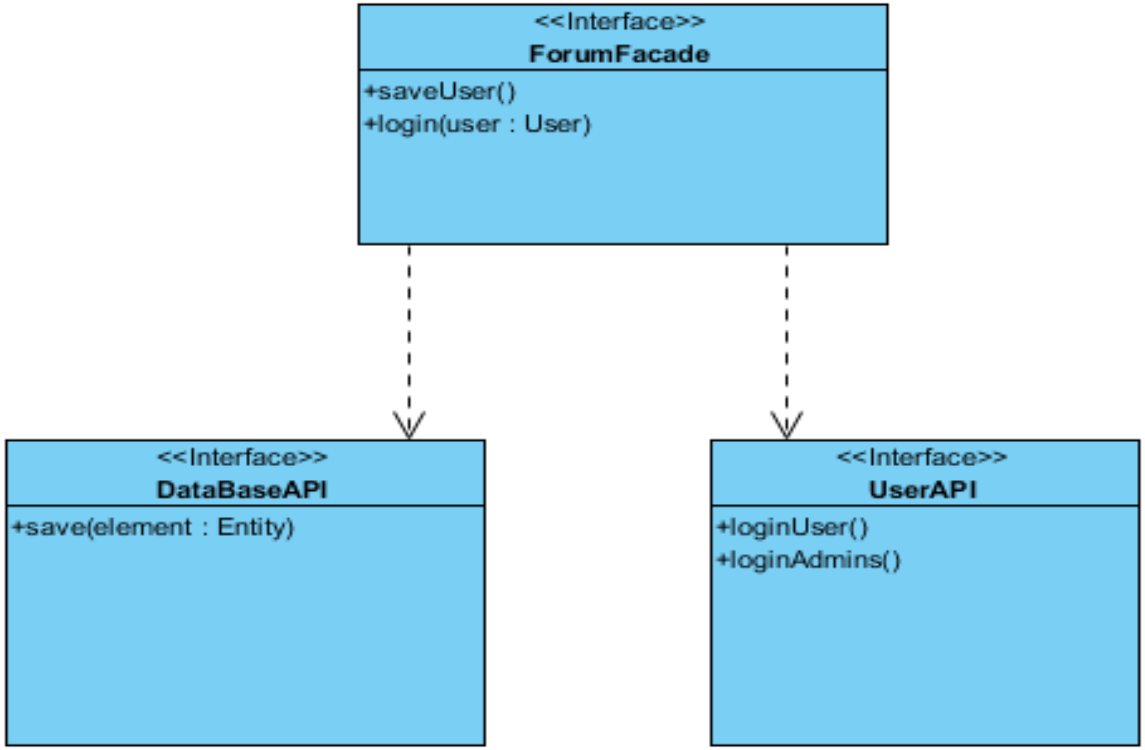
Abstract factory

- Provides an interface for creating related classes without specifying their concrete implementation
- The clients (those who use the factory) need only have access to the abstract factory itself without any actual knowledge over the concrete factory implementation.
- The intent of this pattern is to create a barrier, an insolation between the creation of objects and their usage.

Structural

- These patterns handle the creation and composition of a complex structure of objects thereby providing new behavior.
- Inheritance is used to create interfaces or concrete implementations, usually both.
  - Inheritance allows us to introduce new behavior to an already existent family of classes (the child can add or override existing functionality). In java there is no multiple inheritance, therefore we need a way to manage the introduction of multiple behavior from different sources, with structural patterns we can introduce just that.

Facade


<<Interface>>
ForumFacade
+saveUser()
+login(user : User)
<<Interface>>
DataBaseAPI
+save(element : Entity)
<<Interface>>
UserAPI
+loginUser()
+loginAdmins()

Facade

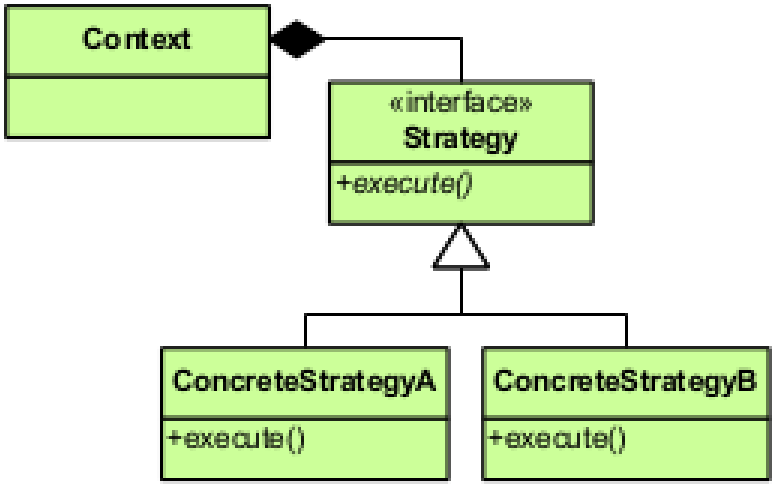
- Very similar to a (real) building's façade
  - Building facades hide the "ugly" walls and bricks of that building
- Software facades add a common interface , an easy way to manipulate a big "chunk of code". It hides away the ugly internals and presents us with a nice and clean interface.
- What we gain by using a façade
  - Provide methods for common functionality which needn't necessarily exist in the code the façade is hiding
  - We hide away the internals of the code base we are using, allowing us to easily switch away from it if need be.

Behavioral

- These patterns concern themselves with algorithms (how a particular task is accomplished) and how objects interact with one another
- Behavioral patterns distribute functionality across classes , that is achieved either by Inheritance (ex. Template method pattern) or composition (ex. Observer pattern)

Strategy

- Strategy patterns allows a concrete algorithm to be selected at runtime



- Think of strategy as a switch statement for behavior

Strategy

- Use strategy by creating several algorithms, encapsulating each one , and making them interchangeable.
- Strategy lets the algorithm vary independently regardless of the who is using them.

Conclusion

- Design patterns gives a common well established, well tested manner in which we can solve common problems
- It allows us better communication between team members. It's easy to say : "… ahh let's use an Obeserver here" or "… wouldn't a strategy be better in this case" and everyone in the room will instantly understand what you mean.
- It's a good idea to learn some of the most common design patterns, good knowledge of (at least the common ones) a number of design patterns is the sign of a good software engineer.

Q and A ?

Problems ?!

- What's the purpose of Design Patterns?
- When to use
  - Structural Design Patterns ?
  - Behavioral Design Patterns ?
  - Creational Design Patterns ?
- What are the main components of MVC ?
- Is it a good idea to separate the different application layers? Why ?