OOP

Oblikovni obrasci
kratki pregled

U ovom poglavlju naučit ćete

- Što je ideja oblikovnih obrazaca
- SOLID principi
- Adapter, Factory, Singleton primjer

Oblikovni obrasci

- a general repeatable solution to a commonly occurring problem
- drafts of solutions, not their implementation
- tested, proven approach paradigms
- speed up application development
- increase the readability of the code
- reduce the need for changes
- must not be purpose in itself

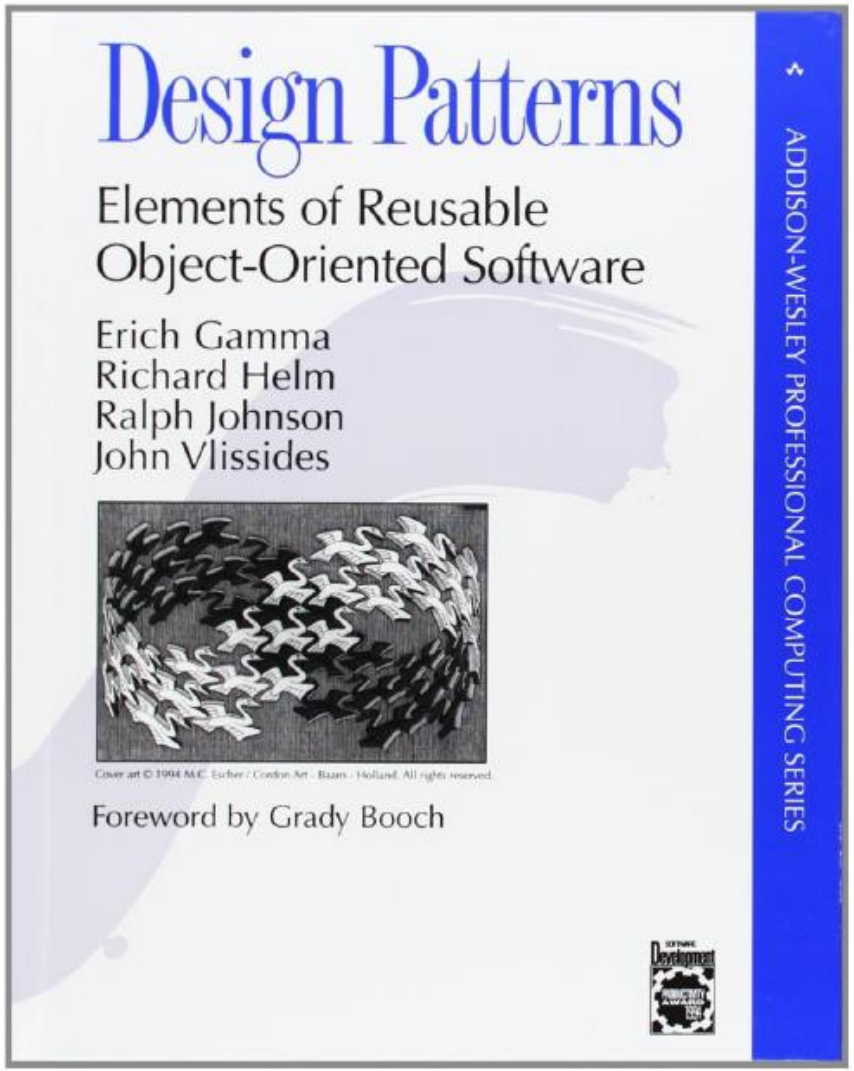
Intro – design patterns

- John Lennon: "There are no problems, only solutions"
- We solve problems that have had to solve countless time before
- Knowledge of patterns => breaking down complex solutions, develop applications in uniformed way with TRIED and TRUSTED solutions

Intro – design patterns (2)

- Design pattern (DP) = high-level abstract solution templates
- Language agnostic
- Origin: C. Alexander, 1970, architecture
- Origin in software development: Design Patterns: Elements of Reusable Object-Oriented Software (GoF)
  - 23 design patterns, 3 groups

GoF


Design Patterns
Elements of Reusable
Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Common design principles

- Keep It Simple Stupid (KISS)
- Don’t Repeat Yourself (DRY)
- You Ain’t Gonna Need It (YAGNI)
- Separation of Concerns (SoC)
- Liskov Substitution Principle (LSP)
- Interface Segregation Principle (ISP)
- Dependency Inversion Principle (DIP)
- Dependency Injection (DI) and Inversion of Control (IoC)

S.O.L.I.D design principles

- **S**ingle Responsibility Principle (SRP)
- **O**pen-Closed Principle (OCP)
- **L**iskov Substitution Principle (LSP)
- **I**nterface Segregation Principle (ISP)
- **D**ependency Inversion Principle (DIP)

Single Responsibility Principle (SRP)

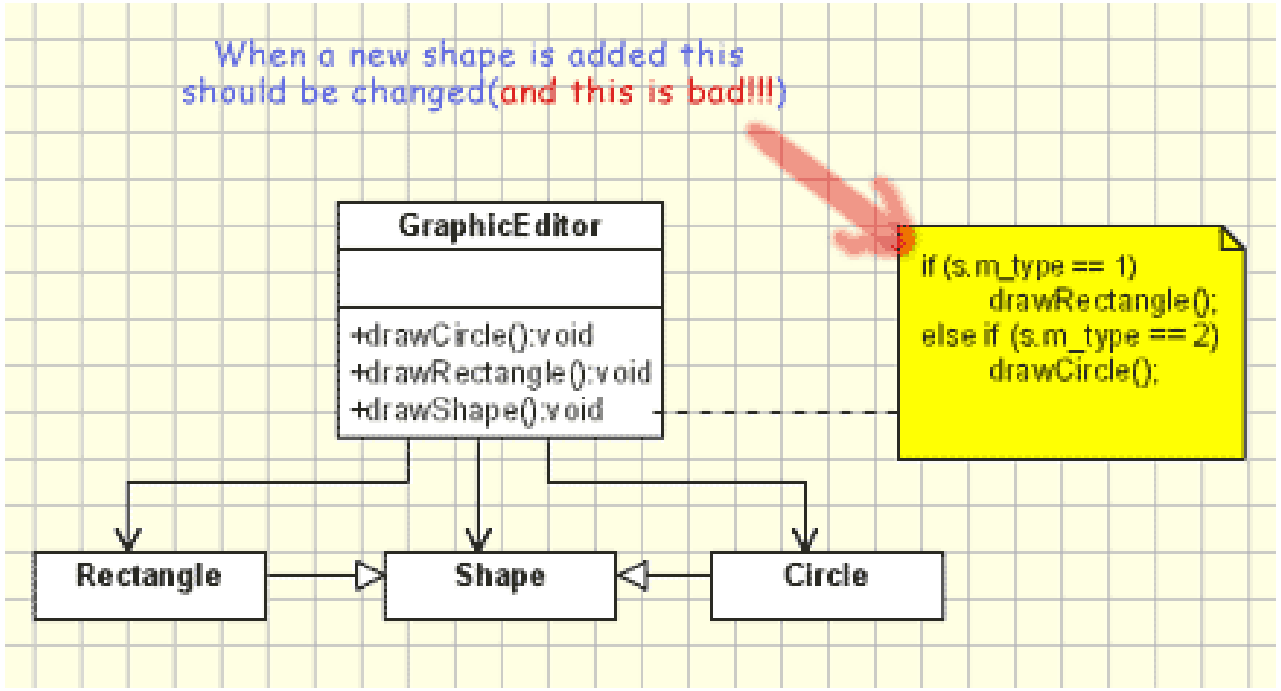
- Responsibility is considered to be one reason to change
- If we have 2 reasons to change for a class, we have to split the functionality in two classes

- A class should have only one reason to change.

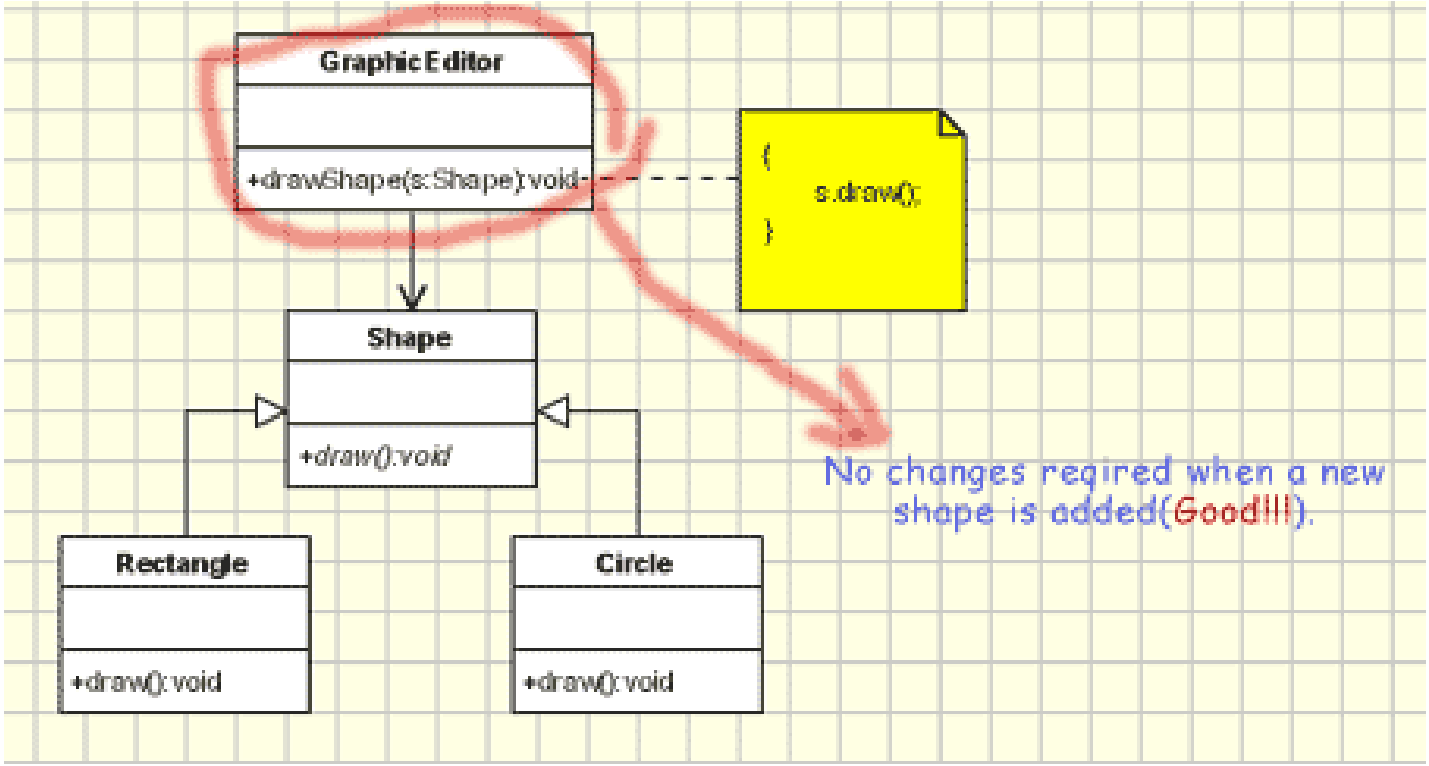
Open Closed Principle (OCP)

- Design and writing of the code should be done in a way that new functionality should be added with minimum changes in the existing code.
- The design should be done in a way to allow the adding of new functionality as new classes, keeping as much as possible existing code unchanged.
- Software entities like classes, modules and functions should be open for extension but closed for modifications.

Open Closed Principle(2) – bad example



Open Closed Principle(3) – good example



Liskov's Substitution Principle (LSP)

- All the time we design a program module and we create some class hierarchies. Then we extend some classes creating some derived classes
- We must make sure that the new derived classes just extend without replacing the functionality of old classes. Otherwise the new classes can produce undesired effects when they are used in existing program modules.

- Derived types must be completely substitutable for their base types.

Interface Segregation Principle (ISP)

- Clients should not be forced to implement interfaces they don't use.
- Instead of one fat interface many small interfaces are preferred based on groups of methods, each one serving one submodule.

- Clients should not be forced to depend upon interfaces that they don't use.

Dependency Inversion Principle (DIP)

- High-level modules should not depend on low-level modules. Both should depend on abstractions.
- Abstractions should not depend on details. Details should depend on abstractions.

- Repeat that ☺

Podjela dizajnerskih predložaka

- **Kreacijski predlošci**
  - Singleton
  - Factory
  - Abstract Factory
  - Factory Nethod
  - Builder
  - Prototype
  - Object Pool

Podjela dizajnerskih predložaka

- **Strukturalni predlošci**
  - Facade
  - Adapter
  - Bridge
  - Composite
  - Decorator
  - Flyweight
  - Proxy

Podjela dizajnerskih predložaka

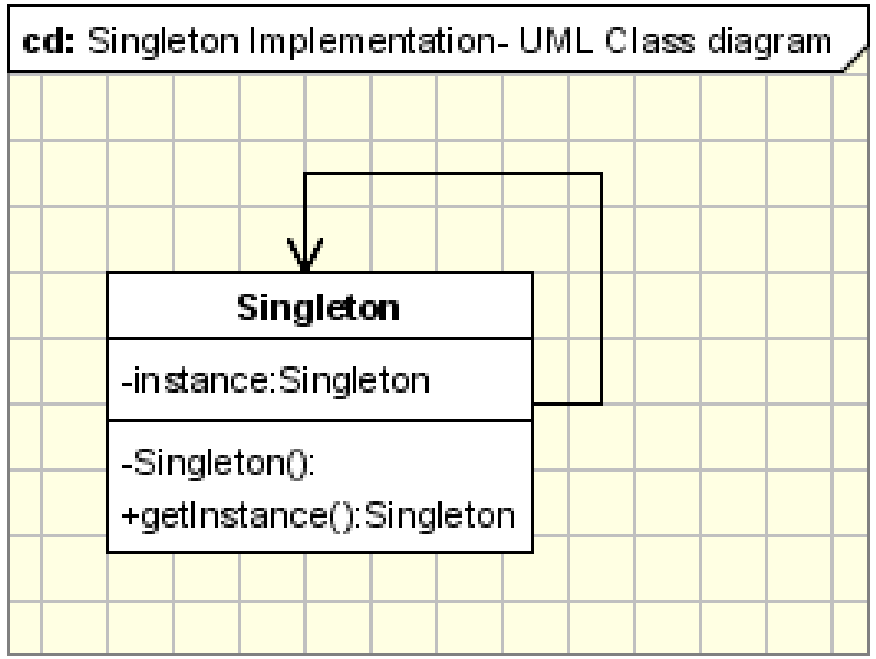
- **Predlošci ponašanja**
  - Chain of Responsibility
  - Command
  - Interpreter
  - Iterator
  - Mediator
  - Memento
  - Observer
  - Strategy

Podjela dizajnerskih predložaka(2)

- **Predlošci ponašanja**
  - Template Method
  - Visitor
  - Null Object

Singleton

- Cilj: Osigurati da će se kreirati samo jedna (N) instanca neke klase

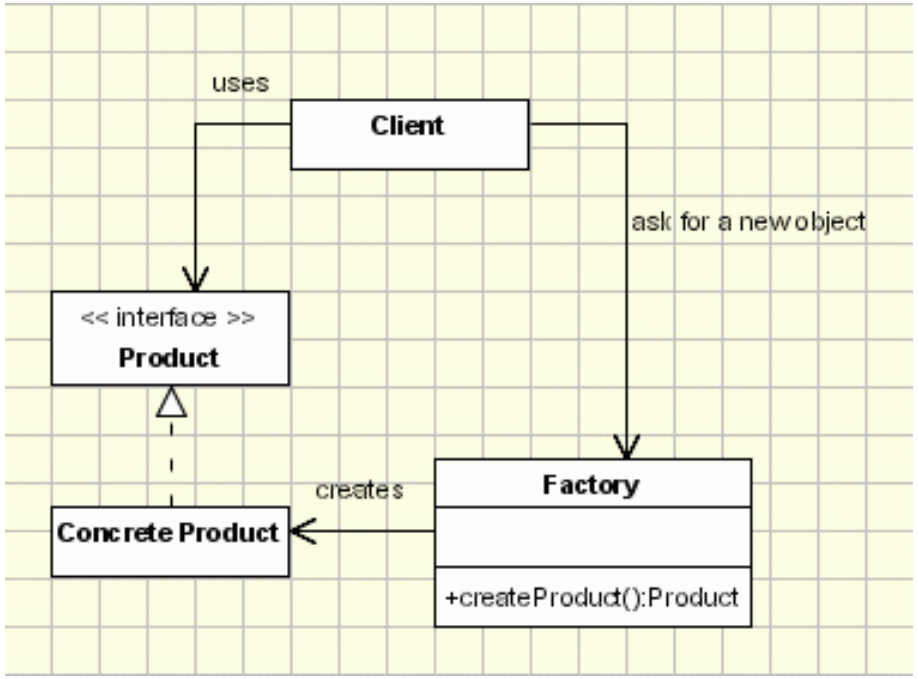


Singleton(2)

- **Gdje ga koristiti?**
  - Logiranje
  - Konfiguracijske klase
  - „Factories”

Factory

- Cilj: Kreirati objekte bez da se logika kreiranja prikazuje klijentu



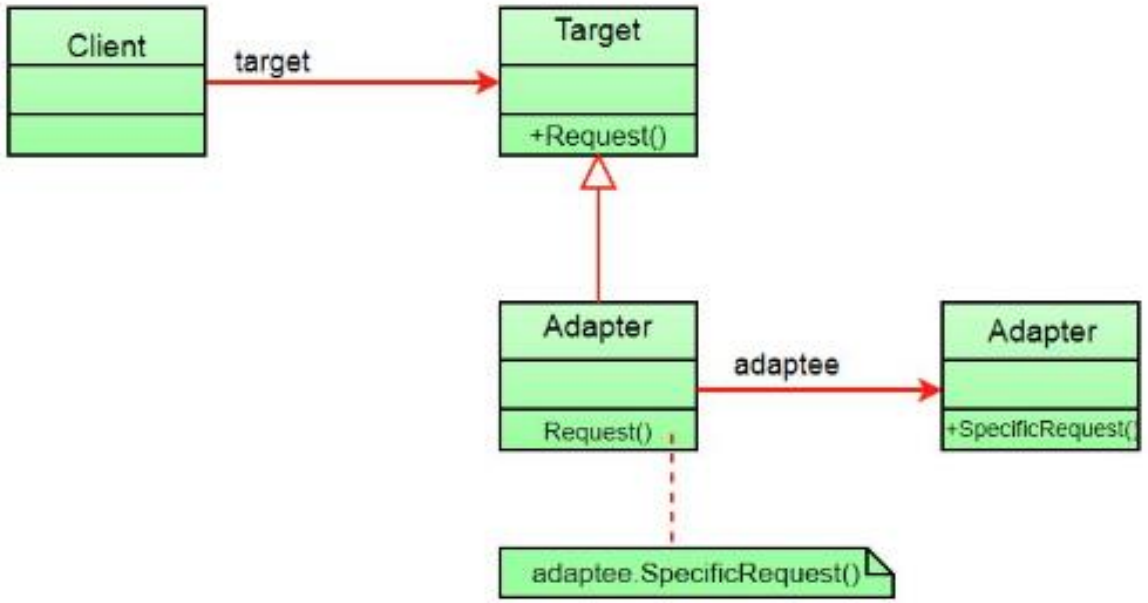
Factory(2)

- **Gdje ga koristiti?**
  - Gdje ne ☺
  - Vjerojatno najčešće korišteni predložak u OOP rješenjima
- **Tipični problemi**
  - Parametrizirani „Factory”

Adapter

- **Gdje ga koristiti?**
  - Suradnja između klasa koje inače ne bi mogle raditi zajedno zbog inkompatibilnosti njihovih sučelja

Adapter


Client
target
Target
+Request()
Adapter
Request()
adaptee
Adapter
+SpecificRequest()
adaptee.SpecificRequest()

Hvala na pažnji!