NATIONAL RESEARCH UNIVERSITY

Software Architecture Design Patterns

Lecture 03

NATIONAL RESEARCH UNIVERSITY

Today

- **Game architecture (remind)**
- **Design patterns (referenced lecture)**

NATIONAL RESEARCH UNIVERSITY

Software Architecture

- *Architecture is the fundamental* ***organization*** *of a* ***system*** *embodied in its* ***components****, their* ***relationships*** *to each other, and to the* ***environment****, and the principles guiding its design and evolution. [IEEE 1471]*
- *The* ***software architecture*** *of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. Architecture is concerned with the public side of interfaces; private details of elements—details having to do solely with internal implementation—are not architectural* [*Software Architecture in Practice (2nd edition)*, Bass, Clements, and Kazman]
  - What is a software architecture? (https://www.ibm.com/developerworks/rational/library/feb06/eeles/)
  - Software architecture and design. Chapter 1. What is software architecture? (https://msdn.microsoft.com/ru-ru/library/ee658098.aspx)

Architectural Style

- *An architectural style defines a* family of systems in terms of a pattern of structural organization. More specifically, an architectural style determines the vocabulary of components and connectors that can be used in instances of that style, together with a set of constraints on how they can be combined. These can include topological constraints on architectural descriptions (e.g., no cycles). Other constraints—say, having to do with execution semantics—might also be part of the style definition.
  - [David Garlan and Mary Shaw, January 1994, CMU-CS-94-166, see "An Introduction to Software Architecture" at http://www.cs.cmu.edu/afs/cs/project/able/ftp/intro_softarch/intro_softarch.pdf]

NATIONAL RESEARCH UNIVERSITY

Key Architecture Principles

- **Build to change instead of building to last**. Consider how the application may need to change over time to address new requirements and challenges, and build in the flexibility to support this.
- **Model to analyze and reduce risk**. Use design tools, modeling systems such as Unified Modeling Language (UML), and visualizations where appropriate to help you capture requirements and architectural and design decisions, and to analyze their impact. However, do not formalize the model to the extent that it suppresses the capability to iterate and adapt the design easily.
- **Use models and visualizations as a communication and collaboration tool**. Efficient communication of the design, the decisions you make, and ongoing changes to the design, is critical to good architecture. Use models, views, and other visualizations of the architecture to communicate and share your design efficiently with all the stakeholders, and to enable rapid communication of changes to the design.
- **Identify key engineering decisions**. Use the information in this guide to understand the key engineering decisions and the areas where mistakes are most often made. Invest in getting these key decisions right the first time so that the design is more flexible and less likely to be broken by changes.
    - Chapter 1: What is Software Architecture? (https://msdn.microsoft.com/ru-ru/library/ee658098.aspx)

Key Architectural Styles

- **Client/Server** segregates the system into two applications, where the client makes requests to the server. In many cases, the server is a database with application logic represented as stored procedures.
- **Component-Based Architecture** decomposes application design into reusable functional or logical components that expose well-defined communication interfaces.
- **Domain Driven Design** an object-oriented architectural style focused on modeling a business domain and defining business objects based on entities within the business domain.
- **Layered Architecture** partitions the concerns of the application into stacked groups (layers).
- **N-Tier / 3-Tier** segregates functionality into separate segments in much the same way as the layered style, but with each segment being a tier located on a physically separate computer.
- **Object-Oriented** a design paradigm based on division of responsibilities for an application or system into individual reusable and self-sufficient objects, each containing the data and the behavior relevant to the object.
- **Service-Oriented Architecture (SOA)** refers to applications that expose and consume functionality as a service using contracts and messages.

NATIONAL RESEARCH UNIVERSITY

Games Architecture (Refs)

- Introduction to Component Based Architecture in Games (https://www.raywenderlich.com/24878/introduction-to-component-based-architecture-in-games)
- Remember of Software Architecture AntiPatterns (https://sourcemaking.com/antipatterns/software-architecture-antipatterns)

NATIONAL RESEARCH UNIVERSITY

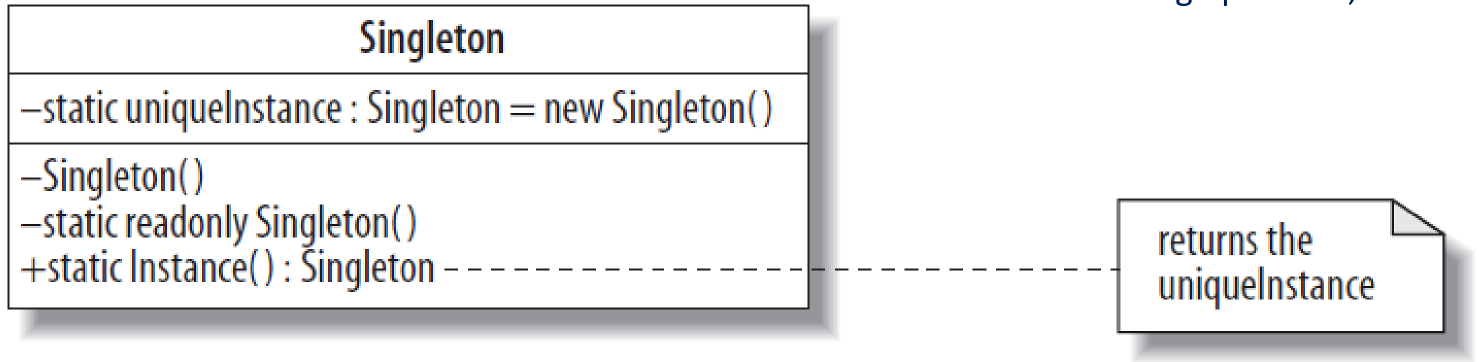
Design patters

- A pattern is that it is a solution to a problem in a context
  - M. Fauler, What is a pattern (https://martinfowler.com/articles/writingPatterns.html)
- Design patterns revisited (http://gameprogrammingpatterns.com/design-patterns-revisited.html)

Singleton

- The pattern ensures that the class is instantiated only once and that all requests are directed to that one and only object.

C# 3.0 Design patterns, J. Bishop


Singleton
–static uniqueInstance : Singleton = new Singleton( )
–Singleton( )
–static readonly Singleton( )
+static Instance( ) : Singleton
returns the uniqueInstance

NATIONAL RESEARCH UNIVERSITY

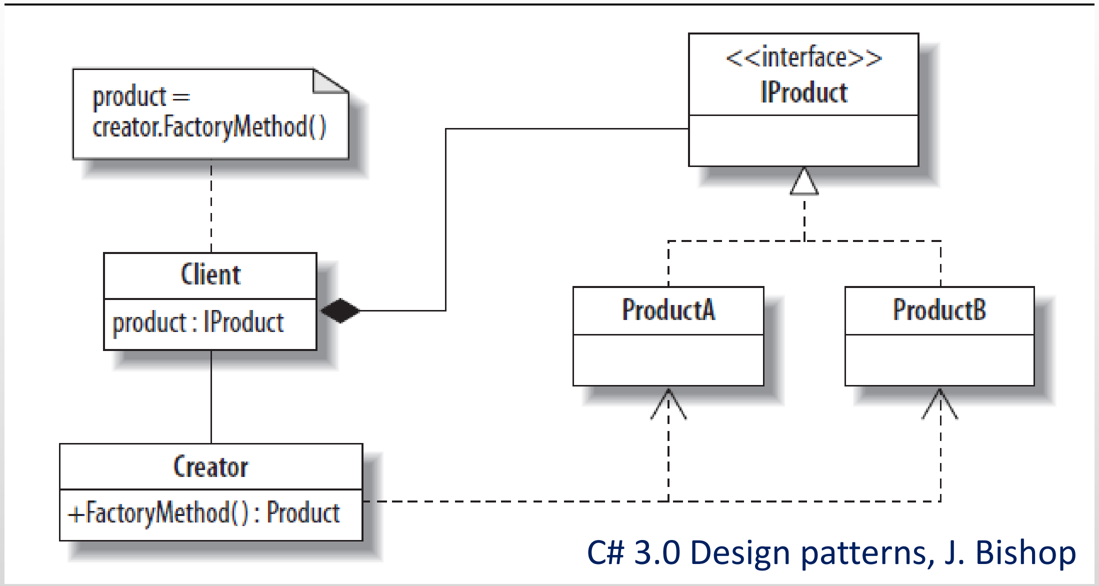
Singleton in Games

- **Why to avoid singletons, see**
  - (http://gameprogrammingpatterns.com/singleton.html)
  - Krakendev. Defeating the anti-pattern bully. Part 1 – Singletons (https://krakendev.io/blog/antipatterns-singletons)
  - D. Buchanan The Singleton Anti-Pattern (http://dillonbuchanan.com/programming/singleton-anti-pattern/)
- More about software architecture antipatterns AntiPatterns (https://sourcemaking.com/antipatterns/software-development-antipatterns)

NATIONAL RESEARCH UNIVERSITY

Factory Method

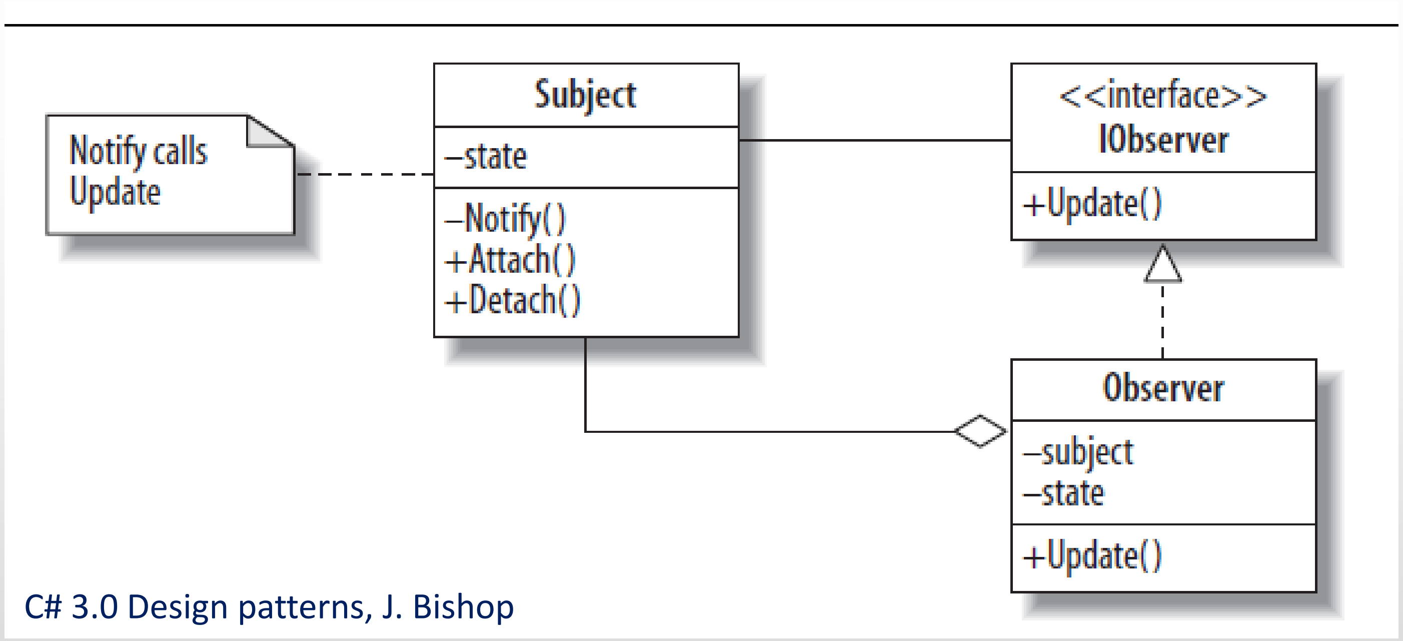
- The pattern is a way of creating objects, but letting subclasses decide exactly which class to instantiate. Various subclasses might implement the interface; the Factory Method instantiates the appropriate subclass based on information supplied by the client or extracted from the current state.



C# 3.0 Design patterns, J. Bishop

Observer

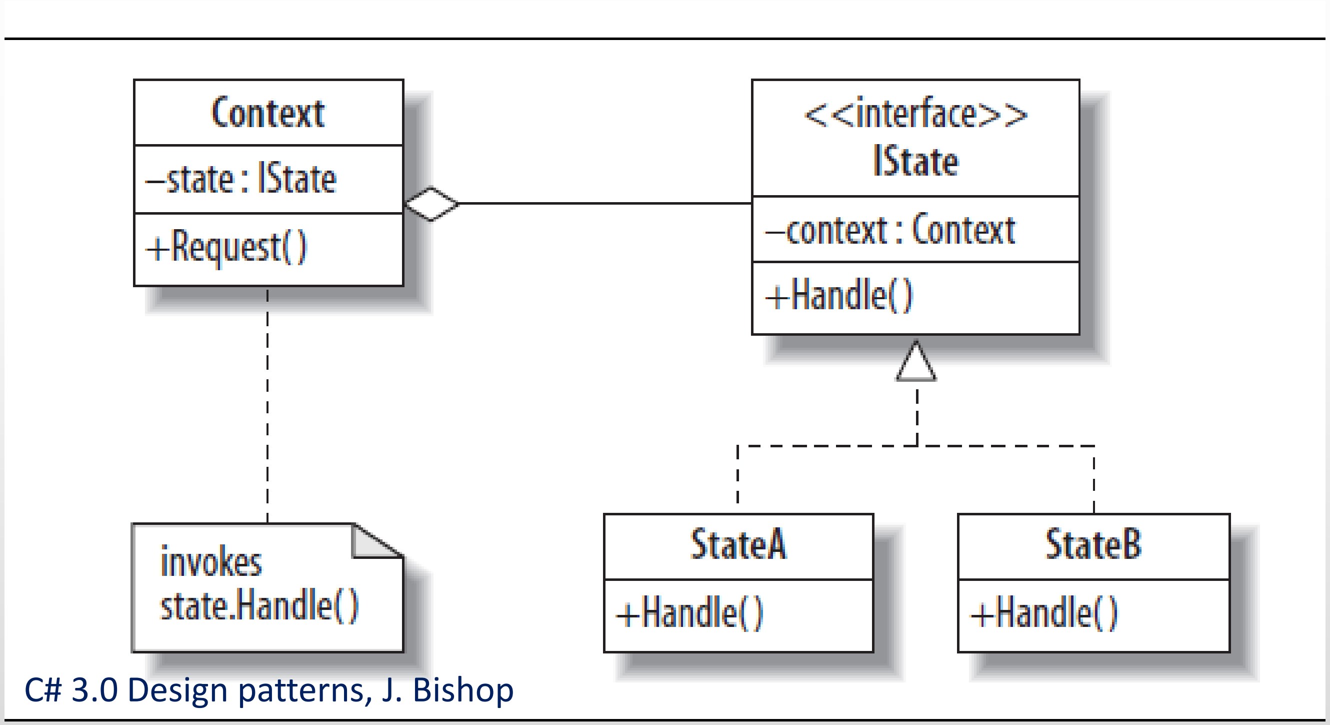
- The pattern defines a relationship between objects so that when one changes its state, all the others are notified accordingly.



C# 3.0 Design patterns, J. Bishop

State

- When the state inside an object changes, it can change its behavior by switching to a set of different operations.



C# 3.0 Design patterns, J. Bishop

NATIONAL RESEARCH UNIVERSITY

Assignment

- In groups of 2
- Revise games patterns introduced in “Game programming patterns” (http://gameprogrammingpatterns.com/contents.html)
  - Sequencing patterns
  - Behavioral patterns
  - Decoupling patterns
  - Optimization patterns
- For more information use Fabien Sanglard’s website (http://fabiensanglard.net) and Michael Haney’s post “Design patterns in game programming (https://www.gamasutra.com/blogs/MichaelHaney/20110920/90250/Design_Patterns_in_Game_Programming.php)
- Prepare short (15 min) introduction of studied group of patterns
  - Name of a group, list of patterns
  - Explain the connection between patterns and games scenarios, features, etc.
  - Give illustrations and examples
  - Utilize extra links on your topic in OneNote in Collaboration Space
  - The thesis for your introduction should be also presented in OneNote

NATIONAL RESEARCH UNIVERSITY

Links

- Alsphaugh T.A. Software architecture (http://www.thomasalspaugh.org/pub/fnd/architecture.html)
- Meyer, B., Pedroni, M. Lecture 15: Architectural styles (http://se.inf.ethz.ch/old/teaching/2010-S/0050/slides/15_softarch_styles.pdf)
- *Game Development. Stack exchange* (http://gamedev.stackexchange.com/)
- *Designing for Mobile, Part 1: Information Architecture* (http://www.uxbooth.com/articles/designing-for-mobile-part-1-information-architecture/)
- *Designing Elegant Mobile Games* (https://www.objc.io/issues/18-games/designing-elegant-mobile-games/)
- M. Rohs *Mobile Input & Output Technologies* (https://www.medien.ifi.lmu.de/lehre/ws1011/mmi2/mmi2-3d-Mobile-IO.pdf)

Links

- M. Haney *Design Patterns in Game Programming* (http://www.gamasutra.com/blogs/MichaelHaney/20110920/90250/Design_Patterns_in_Game_Programming.php)
- R. Nystrom *Game Programming Patterns* (http://gameprogrammingpatterns.com/contents.html)
- M. Luedke *Common Design Patterns for Android* (https://www.raywenderlich.com/109843/common-design-patterns-for-android)
- *Mobile Application Architecture Guide* (http://robtiffany.com/wp-content/uploads/2012/08/Mobile_Architecture_Guide_v1.1.pdf)
- I. Astahovs *Use of Design Patterns for Mobile Game Development* (https://www.diva-portal.org/smash/get/diva2:546698/FULLTEXT01.pdf)