



Lecture:


Designing classes

How to write classes in a way that they are easily understandable, maintainable and reusable



Software changes

- Software is not like a novel that is written once and then remains unchanged.
- Software is extended, corrected, maintained, ported, adapted...
- The work is done by different people over time (often decades).

- 
- Design is not a science.
 - There are no prescriptive formulas you can follow during design that will assure a optimal design solution in the end.
 - The most that can be offered are general guidelines and principles.

OOA, OOD, and OOP

Object-oriented methods may be applied to different phases in the software life-cycle:

e.g., analysis, design, implementation, etc.

- **OO analysis (OOA)** is a process of discovery
Where a development team models and understands the requirements of the system
- **OO design (OOD)** is a process of invention and adaptation

Where the development team creates the abstractions and mechanisms necessary to meet the system's behavioral requirements determined during analysis

Designing classes

A construction company would like to handle a customer's order for a new home. The customer may select one of four models of a home. Model A for 10000, Model B for 120000, Model C for 180000, or Model D for 250000. Each model can have one, two, three, or four bedrooms.

Several nouns have been underlined in this description. Identify the possible objects that may be defined by this narrative. Not all the nouns underlined make sense when used as a potential object.

Abstraction and modularization

- **Abstraction** is the ability to ignore details of parts to focus attention on a higher level of a problem.
- **Modularization** is the process of dividing a whole into well-defined parts, which can be built and examined separately, and which interact in well-defined ways.



Divide-and-conquer

The Role of Abstraction

- "We (humans) have developed an exceptionally powerful technique for dealing with complexity. We abstract from it. Unable to master the entirety of a complex object, we choose to ignore its inessential details, dealing instead with the generalized, idealized model of the object." (Wulf)

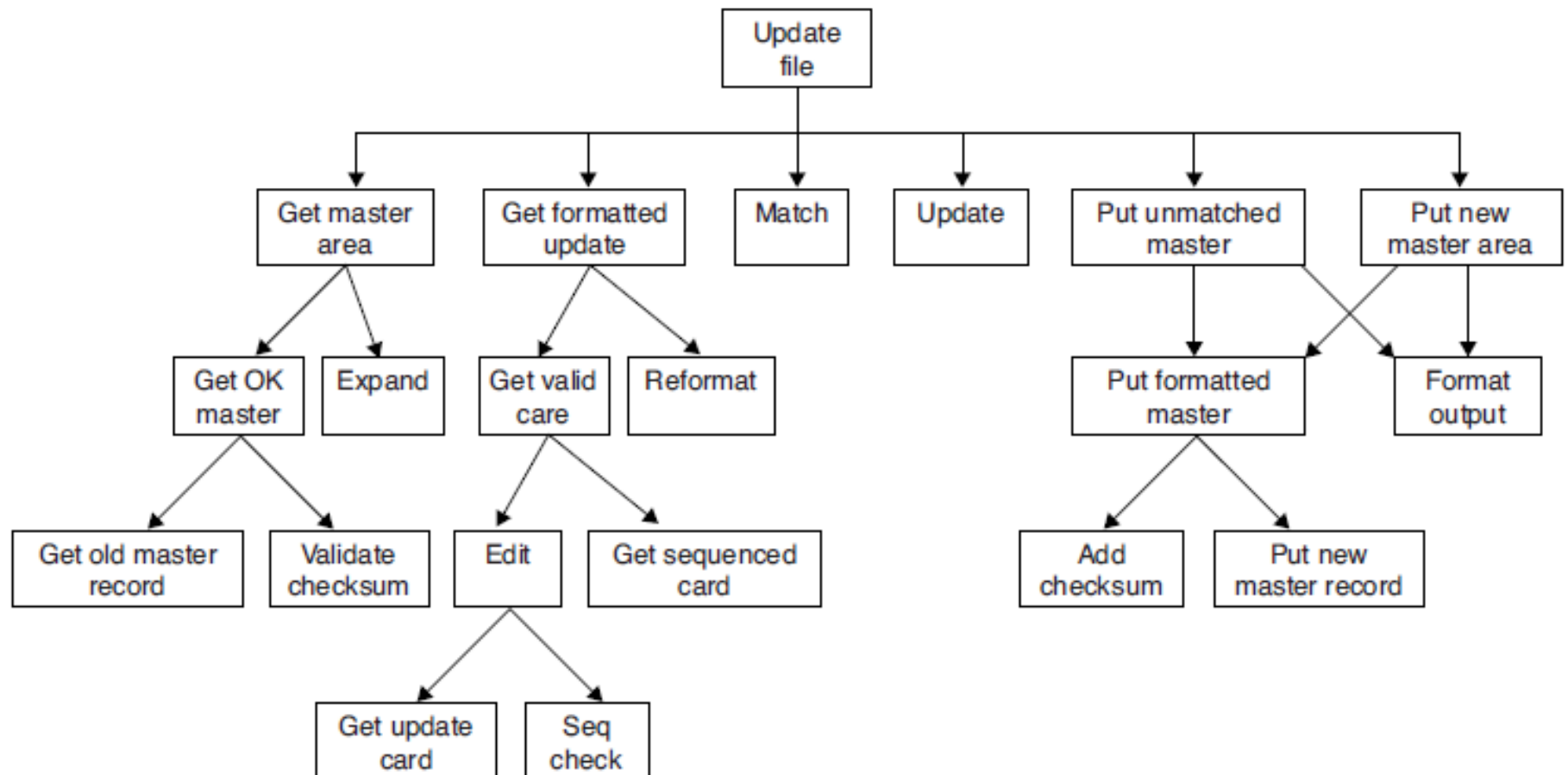
Role of Decomposition

- When designing a complex software system, it is essential to decompose it into smaller and smaller parts, each of which we may then refine independently.
- divide et impera (divide and rule)

Algorithmic Decomposition

- Decomposition as a simple matter of algorithmic decomposition, wherein each module in the system denotes a major step in some overall process.

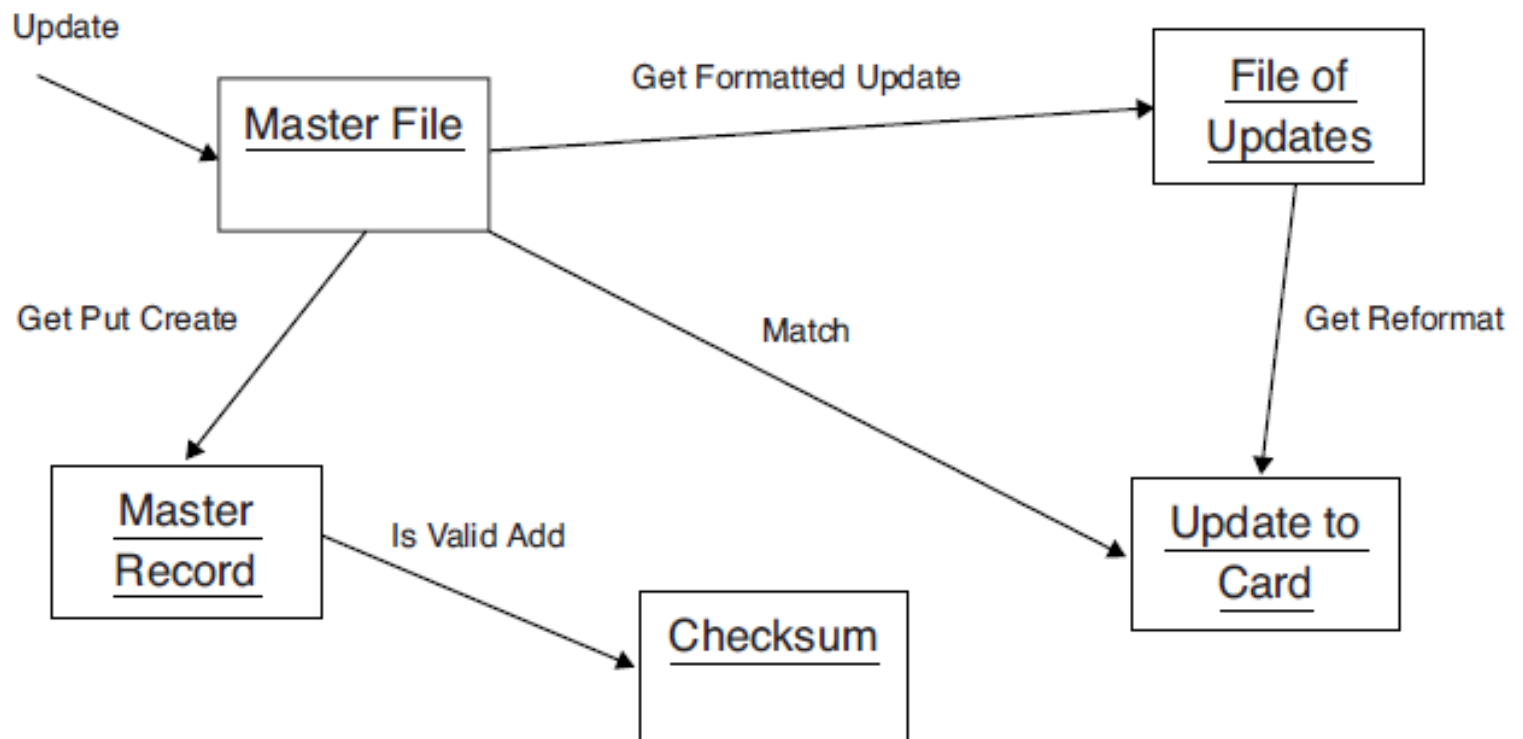
Example



Object-Oriented Decomposition

- Decomposed the system according to the key abstractions in the problem domain.
- We view the world as a set of autonomous agents that collaborate to perform some higher-level behavior.

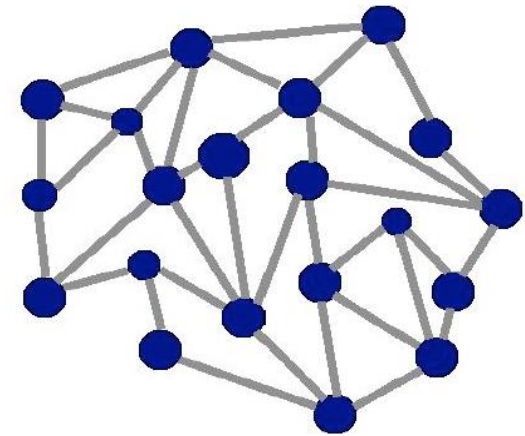
Example



Good OO design

- The most important promise of OO is improve modularization
 - that is, enhance maintainability
 - while still supporting functional decomposition (which is good for cohesion)
 - while also promoting re-use
- Good OO = high cohesion + low coupling

Coupling



Couplin_

- Coupling refers to links between separate units of a program.
- If two classes depend closely on many details of each other, we say they are *tightly coupled*.
- We aim for *loose coupling*.



Loose coupling

Loose coupling makes it possible to:

- understand one class without reading others;
- change one class without affecting others.
- Thus: improves maintainability.



Cohesion

- Cohesion refers to the the number and diversity of tasks that a single unit is responsible for.
- If each unit is responsible for one single logical task, we say it has *high cohesion*.
- Cohesion applies to classes and methods.
- We aim for high cohesion.



High cohesion

High cohesion makes it easier to:

- understand what a class or method does;
- use descriptive names;
- reuse classes or methods.



Cohesion

- Of methods: a method should be responsible for one and only one well defined task.
- Of Classes: a class should represent one single, well defined entity.



Code duplication


Code duplication

- is an indicator of bad design,
- makes maintenance harder,
- can lead to introduction of errors during maintenance.



Localizing change

- One aim of reducing coupling and responsibility-driven design is to localize change.
- When a change is needed, as few classes as possible should be affected.



Основни въпроси при проектирането

- Колко голям трябва да е един метод?
- Колко голям трябва да е един клас?
- Можете ли да отговорите в смисъла на кохезия и свързаност ???

Кохезия

- За метод: Всеки един метод трябва да е отговорен за една и само една добре дефинирана задача.
- За класове: Всеки клас трябва да описва един добре дефиниран обект в проблемната област.

Принципи на проектирането

- Един метод е прекалено голям, когато изпълнява повече от една задача;
- Един клас е прекалено сложен, ако описва повече от една логическа единици.

Code: To Reuse or Not to Reuse?

- One of the major advantages touted by OO proponents is that if you write code properly the first time, you can reuse it to your heart's content.
 - One way to create reusable code is to create frameworks.
 - Using interfaces and abstract classes to create frameworks and encourage reusable code.

What Is a Framework?

- Hand-in-hand with the concept of code reuse is the concept of *standardization*, which is sometimes called *plug-and-play*.
- The idea of a framework revolves around these plug-and play and reuse principles.

model-view-controller (MVC)

- In object-oriented programming development, model-view-controller (MVC) is the name of a methodology or design pattern for successfully and efficiently relating the user interface to underlying data models.

model-view-controller (MVC)

- A *Model* , which represents the underlying, logical structure of data in a software application and the high-level class associated with it. This object model does not contain any information about the user interface.
- A *View* , which is a collection of classes representing the elements in the user interface (all of the things the user can see and respond to on the screen, such as buttons, display boxes, and so forth)
- A *Controller* , which represents the classes connecting the model and the view, and is used to communicate between classes in the model and view.

Object-oriented concepts

A brief review of underlying themes...

Abstraction: Process of identifying essential aspects of an entity and ignoring unimportant properties.

- Concentrate on what an object is and what it does, before deciding how to implement it.

Encapsulation: Object contains both data structure and set of operations used to manipulate it.

Information Hiding: Separate external aspects of an object from its internal details, which are hidden from outside.

- Allows internal details of object to be changed without affecting apps that use it, provided external details remain same.
- Provides data independence.

General design guidelines and principles

- Abstraction - abstraction is the principle means of managing complexity.
- Modularization - a complex system can be simplified by dividing it into smaller, easier-to-understand modules.
- Coupling and Cohesion - coupling between modules should be low and cohesion within modules should be high. This reduces dependencies that make the system harder to understand and maintain.
- Information Hiding - implementation details and likely-to-change design decisions should be hidden behind simple well-defined interfaces. This reduces coupling and provides for better abstractions.

Resource

- <http://sce2.umkc.edu/BIT/burris/pl/oop/>