GRASP: Designing objects with responsibilities

INTRODUCTION

- One way to describe object design "After identifying your requirements and creating a domain model, then add methods to the software classes, and define the messaging between the objects to fulfill the requirements"
- Not really- Answer the following questions:
  - What methods belong to where?
  - How the objects should interact?
- GRASP as Methodical Approach to Learning Basic Object Design

UML versus Design Principles

- The UML is simply a standard visual modeling language, knowing its details doesn't

  teach you how to think in objects – that is the theme of this course
- The critical design tool for software development is a mind well educated in design principles. It is not the UML or any other technology

Object Design

After the requirements identification, add the methods to the classes and define the message between the objects. The designing of object starts with

- **Inputs**
- **Activities**
- **Outputs**

Inputs to object design

- Use case model
- Domain Model
- System Sequence Diagrams
- Operation Contracts

- Supplementary Specification

Activities of object design

- Dynamic and static modeling (draw both interaction and complementary class diagrams)
- Applying various OOD principles
    - GRASP- General Responsibility Assignment Software Patterns
    - GoF (Gang of Four)design patterns
    - Responsibility-Driven Design (RDD)

Outputs of object design

- Modeling for the difficult part of the design that we wished to explore before coding
- Specifically for object design,
    - UML Interaction diagrams
    - Class diagrams
    - Package diagrams
- UI sketches and prototypes
- Database models Report sketches and prototypes

RDD

- RDD is a general metaphor for thinking about OO software design.
- Thinking of software objects as having responsibilities an abstraction of what they do. Responsibility means a contract or obligation of a classifier**.**
- RDD is a general metaphor for thinking about object oriented design**.**

  Responsibilities are related to the obligation of an object in terms of its behavior
    - what an object should know?
    - what an object should do?

Responsibilities is of two types : Doing , Knowing

1. **Knowing responsibilities**:
   - knowing about private encapsulated data
   - knowing about related objects
   - knowing about things it can derive or calculate
2. **Doing responsibilities**:

   a. doing something itself, such as creating an object or doing a calculation b. initiating action in other objects

   c. controlling and coordinating activities in other objects.

Connection Between Responsibilities, GRASP, and UML Diagrams

Assigning responsibilities to objects while coding or while modeling. Within the UML, drawing interaction diagrams becomes the occasion for considering these responsibilities. **Responsibilities are implemented using methods**

<u>What are Patterns?</u>

A pattern is a named description of a problem and solution that can be applied to new contexts; ideally, a pattern advises us on how to apply its solution in varying circumstances and considers the forces and trade-offs.

Many patterns, given a specific category of problem, guide the assignment of responsibilities to objects.

<u>Example</u> **:** The format is

Pattern Name **:** Information Expert

Problem **:** What is a basic principle by which to assign responsibilities to objects?

Solution **:** Assign a responsibility to the class that has the information needed to fulfill it.

APPLYING GRASP TO OBJECT DESIGN

GRASP stands for "**G**eneral **R**esponsibility **A**ssignment **S**oftware **P**atterns"

- It is a Learning Aid for OO Design with Responsibilities. This approach to understanding and using design principles is based on patterns of assigning responsibilities.
- We can apply the GRASP principles while drawing UML interaction diagrams, and also while coding where we deciding on responsibly assignments.
- **GRASP** defines **nine basic OO design principles or basic building blocks** in design.
- They are

1. Information Expert
2. Creator
3. Controller
4. Low Coupling
5. High Cohesion
6. Polymorphism
7. Pure Fabrication
8. Indirection
9. Protected Variations.

All these patterns answer some software problem, and in almost every case these problems are common to almost every software development project

PATTERN/ PRINCIPLE	DESCRIPTION
Information Expert	A general principle of object design and responsibility assignment? Assign a responsibility to the information expert – the class that has the information necessary to fulfill the responsibility.
Creator	Who creates? (Note that Factory is a common alternate solution.) Assign class B the responsibility to create an instance of class A if one of these is true: 1. B contains A 2. B aggregates A 3. B has the initializing data for A
Controller	What first object beyond the UI layer receives and coordinates ("controls") a system operation?Assign the responsibility to an object representing one of these choices: 1. Represents the overall "system," a "root object," a device that the software is running within, or a major subsystem (these are all variations of a façade controller).
Low coupling (evaluative)	How to reduce the impact of change?Assign responsibilities so that (unnecessary) coupling remains low. Use this principle to evaluate alternatives.
High Cohesion (evaluative)	How to keep objects focused, understandable, and manageable, and as a side-effect, support low Coupling? Assign responsibilities so that cohesion remains high. Use this to evaluate
Polymorphism	Who is responsible when behavior varies by type?When related alternatives or behaviors vary by type (class), assign responsibility for the behavior – using polymorphic operations – to the types for which the behavior varies.

Pure Fabrication	Who is responsible when you are desperate, and do not want to violate high cohesion and low coupling?Assign a highly cohesive set of responsibilities to an artificial or convenience "behavior" class that does not represent a problem domain concept – something made up, in order to support high cohesion, low coupling, and reuse.
Indirection	How to assign responsibilities to avoid direct coupling? Assign the responsibilities to an intermediate object to mediate between other components or services, so that they are not directly coupled.