UML and Design Patterns

Object-Oriented Programming

UML과 디자인 패턴에 대해 알아보겠습니다.
이 두 가지는 객체지향 소프트웨어 설계에 있어 매우 중요한 도구입니다.

Introduction to UML and Patterns

- Software design tools applicable across programming languages
- Require object-oriented programming (OOP) features
- UML (Unified Modeling Language)
  - Graphical language for software design and documentation
  - Used within the OOP framework
- What are Design Patterns?
  - Template or outline for software tasks
  - Can be implemented as different code in similar applications
- Benefits of UML and Patterns
  - Enhance software design process
  - Improve code reusability
  - Facilitate communication among developers
  - Promote best practices in OOP

UML과 패턴은 프로그래밍 언어에 관계없이 적용할 수 있는 소프트웨어 설계 도구입니다.
단, 해당 언어가 객체지향 프로그래밍 기능을 제공해야 합니다.
먼저 이 두 주제에 대해 간단히 소개하겠습니다.
UML은 Unified Modeling language의 약자로,
객체지향 프로그래밍 프레임워크 내에서
소프트웨어를 설계하고 문서화하는 데 사용되는 그래픽 언어입니다.
디자인 패턴은 소프트웨어 작업의 템플릿 또는 개요 역할을 하며,
유사한 여러 애플리케이션에서 서로 다른 코드로 구현될 수 있습니다.
UML과 디자인 패턴은 소프트웨어 설계 과정을 개선하고,
코드 재사용성을 높이며,
개발자 간의 의사소통을 촉진합니다.
또한 객체지향 프로그래밍의 모범 사례를 보여주기도 합니다.

Human-Oriented Representations

- Needs
  - Most people don't think in programming languages
  - Computer scientists seek more intuitive ways to represent programs
  - Pseudocode: mixture of programming and natural language
- Limitations of Pseudocode
  - Standard tool for programmers
  - Linear and algebraic representation
  - Lacks graphical elements

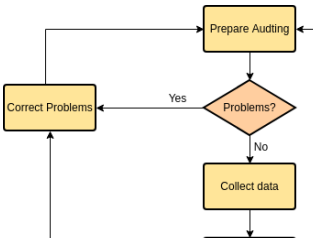
페이지 3

대부분의 사람들은 문제의 solution을 프로그래밍 언어로 생각하지 않습니다.
그래서 컴퓨터 과학자들은 프로그램을 더 직관적으로 표현할 방법을 찾아왔습니다.
그 중 하나가 프로그래밍 언어와 자연어를 혼합한 pseudocode (슈도코드) 입니다.
슈도코드는 프로그래머들의 표준 도구가 되었지만,
선형적이고 대수적인 표현에 그칩니다.
그래픽 요소가 부족하다는 한계가 있습니다.

Evolution of Graphical Representations

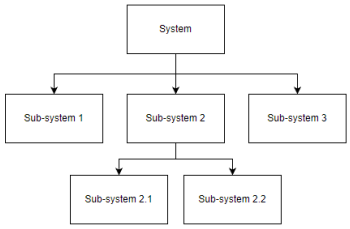
- Past attempts: flowcharts, structure diagrams
- Many graphical representations now outdated
- UML: Current candidate for graphical representation

flowcharts



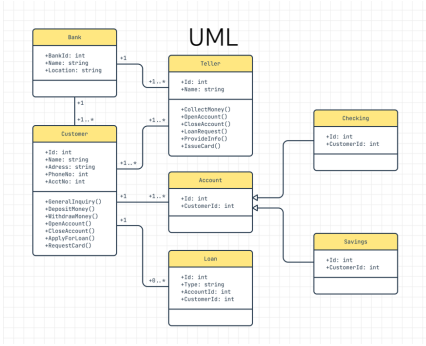
online.visual-paradigm.com

structure diagrams



https://www.savemyexams.com/

UML



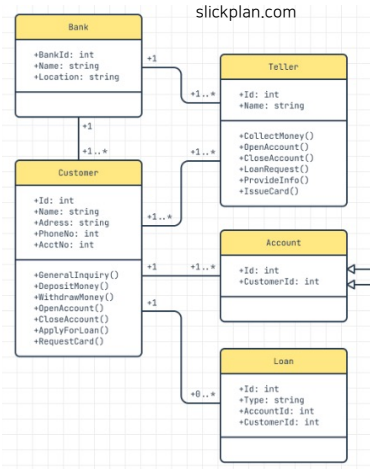
slickplan.com

페이지 4

과거에는 순서도, 구조도 등 다양한 그래픽 표현 방식이 있었지만,
대부분 지금은 구식이 되었습니다.
현재 그래픽 표현의 유력한 후보가 바로 UML입니다.

UML and Object-Oriented Programming

- UML: Designed to reflect OOP philosophy
- Gaining adoption in software design projects
- Still evolving and being tested



페이지 5

UML은 객체지향 프로그래밍 철학을 반영하도록
설계되었습니다.
많은 기업들이 소프트웨어 설계 프로젝트에 UML을 도입하고
있지만,
아직 발전 중이며 검증을 받고 있는 단계입니다.

History of UML

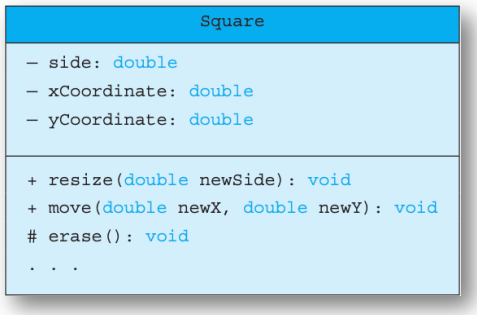
- UML and OOP
  - UML developed alongside Object-Oriented Programming (OOP)
  - Various groups created their own representations for OOP design
- Birth of UML (1996)
  - Created by Grady Booch, Ivar Jacobson, and James Rumbaugh
  - Goal: Standardize graphical representation for OO design and documentation
- UML Today
  - Maintained and certified by Object Management Group (OMG)
  - OMG: Nonprofit organization promoting object-oriented techniques

페이지 6

UML은 객체지향 프로그래밍(OOP)과 함께 발전했습니다.
OOP가 널리 사용되면서 여러 그룹들이 각자의 OOP 설계 표현 방식을 만들어냈습니다.
1996년, Grady Booch, Ivar Jacobson, James Rumbaugh가 UML의 초기 버전을 발표했습니다.
UML의 목표는 객체지향 설계와 문서화를 위한 그래픽 표현 방식을 표준화하는 것이었습니다.
현재 UML 표준은 Object Management Group(OMG)에 의해 관리되고 인증됩니다.
OMG는 객체지향 기술의 사용을 촉진하는 비영리 조직입니다.

UML Class Diagrams (1/2)

- UML Class Diagrams
  - Central to Object-Oriented Programming (OOP)
  - Easy to understand and use
  - Represents a class structure graphically
- Structure of a Class Diagram
  - Box divided into three sections
    - Class name
    - Data specification (instance variables)
    - Actions (class methods)
  - Optional color coding (not standardized)

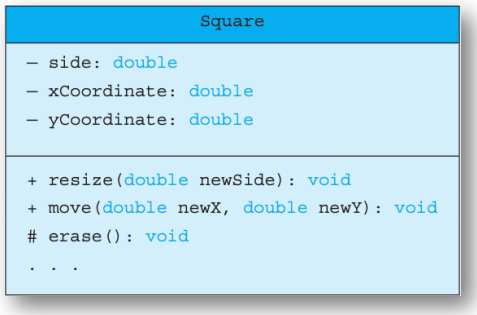


페이지 7

UML 클래스 다이어그램은 객체지향 프로그래밍의 핵심입니다.
이는 클래스 구조를 그래픽으로 표현하여 이해하고 사용하기 쉽게 만듭니다.
클래스 다이어그램은 세 부분으로 나뉜 상자로 구성됩니다.
상단에는 클래스 이름, 중간에는 데이터 스펙 (인스턴스 변수), 하단에는 액션(클래스 메서드)이 위치합니다.
색상 코딩은 선택사항이며 표준화되어 있지 않습니다.

UML Class Diagrams (2/2)

- Access Modifiers in Class Diagrams
  - Minus sign (-): private member
  - Plus sign (+): public member
  - Sharp (#): protected member
  - Tilde (~): package access
- Incomplete Class Diagrams
  - Not all members need to be listed
  - Ellipsis (...) indicates missing members
  - Useful for focused analysis



페이지 8

클래스 다이어그램에서는 접근 제어자를 기호로 표시합니다.
마이너스는 private, 플러스는 public,
샵은 protected, 물결표는 package 접근을 나타냅니다.
클래스 다이어그램은 모든 멤버를 나열할 필요가 없습니다.
생략된 멤버는 줄임표(...)로 표시합니다.
이는 특정 분석에 집중할 때 유용합니다.

Class Interactions in UML

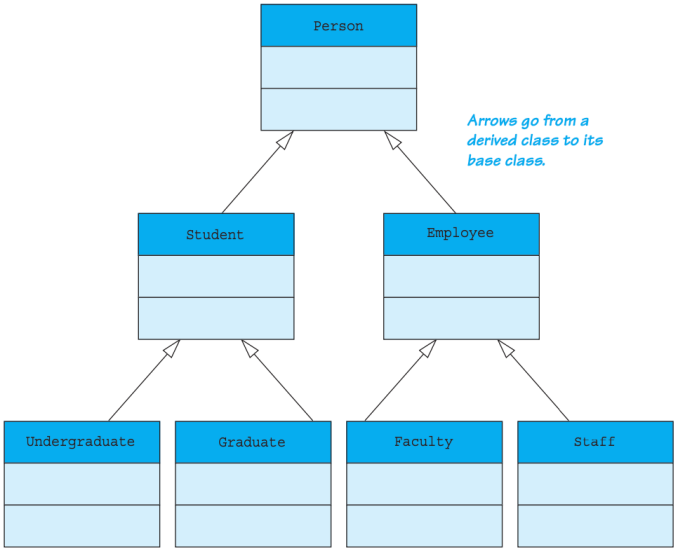
- Class diagrams alone have limited value
- UML provides ways to show class interactions:
  - Annotated arrows for information flow
  - Package groupings
  - Inheritance annotations
  - Other interaction annotations
- Extensibility of UML
  - UML can be extended for specific needs
  - Extensions follow a prescribed framework
  - Ensures understanding among different developers

페이지 9

클래스 다이어그램만으로는 가치가 제한적입니다.
UML은 클래스 간 상호작용을 보여주는 다양한 방법을 제공합니다.
예를 들어, 정보 흐름을 나타내는 주석이 달린 화살표, 패키지 그룹화, 상속 표기 등이 있습니다.
UML은 특정 요구사항에 맞게 확장할 수 있습니다.
이러한 확장은 정해진 프레임워크 내에서 이루어져, 서로 다른 개발자들이 서로의 UML을 이해할 수 있도록 합니다.

Inheritance Diagrams in UML

- Used to represent class hierarchies
- Example: University record-keeping software
- Key Features of Inheritance Diagrams
  - Arrows point from derived (child) class to base (parent) class
  - Unfilled arrowheads indicate inheritance relationship

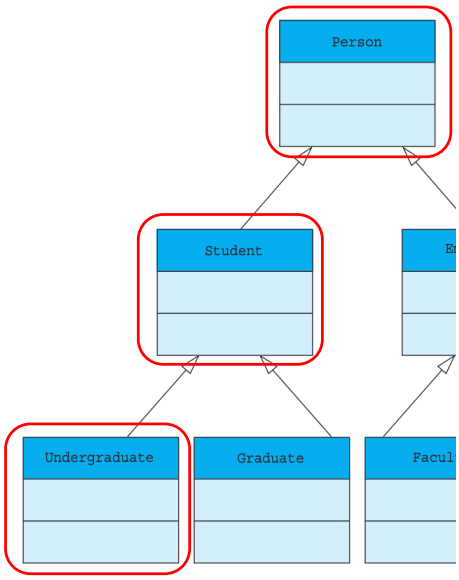


페이지 10

UML의 상속 다이어그램에 대해 알아보겠습니다.
이는 클래스 계층 구조를 표현하는 데 사용됩니다.
대학의 기록 관리 소프트웨어를 예로 들어 설명하겠습니다.
상속 다이어그램의 주요 특징은 다음과 같습니다.
화살표는 자식 클래스에서 부모 클래스로 향하며,
비어 있는 화살표 머리는 상속 관계를 나타냅니다.

Method Definition Location

- Arrows guide method definition search
- Example: Undergraduate class method
  - Look in Undergraduate class
  - If not found, look in Student class
  - If still not found, look in Person class

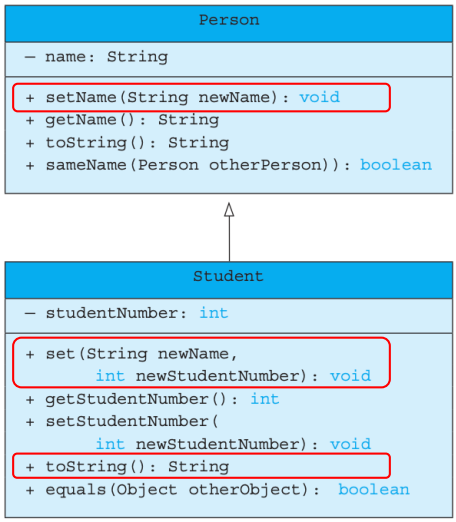


페이지 11

화살표는 메서드 정의를 찾는 데 도움을 줍니다.
예를 들어, Undergraduate 클래스의 메서드를 찾을 때는
먼저 Undergraduate 클래스에서 찾고,
없으면 Student 클래스, 그래도 없으면 Person 클래스에서
찾습니다.

Detailed Example for UML Class Hierarchy

- Student s = new Student(); 일 때
- s.toString()과 s.set("Joe", 4242) 는 class Student에서 찾을 수 있음
- s.setName("Josephine") 은 Student에 없기 때문에 자신의 parent로 가서 Person class에서 찾을 수 있음



페이지 12

Example을 한번 보겠습니다
Student s = new Student(); 로 Student object를 생성합니다
s.toString()과 s.set("Joe", 4242) 는
class Student에서 찾을 수 있습니다
s.setName("Josephine") 은 Student에 없기 때문에 자신의 parent로 올라가서 Person class에서 찾을 수 있습니다.

Design Patterns

- Design outlines applicable across various software applications
- Must be useful across different situations
- Make assumptions about application domains
- Container-Iterator Pattern
  - Container: Class holding multiple data pieces (e.g., array, vector, linked list)
  - Iterator: Construct to cycle through container items
  - Example: Array index as iterator (iterator 'i')

```
for (int i; i < a.length; i++)
    Do something with a[i]
```


페이지 13

디자인 패턴은 다양한 소프트웨어 애플리케이션에 적용할 수 있는
설계 개요입니다.
패턴은 여러 상황에서 유용해야 하며,
적용 대상 애플리케이션 도메인에 대한 가정을 포함합니다.
컨테이너-이터레이터 패턴은 잘 알려진 패턴 중 하나입니다.
컨테이너는 여러 데이터를 보유하는 클래스이고,
이터레이터는 컨테이너 항목을 순회하는 구조입니다.
예를 들어, 배열 인덱스는 배열의 이터레이터 역할을 합니다.

Adaptor Pattern

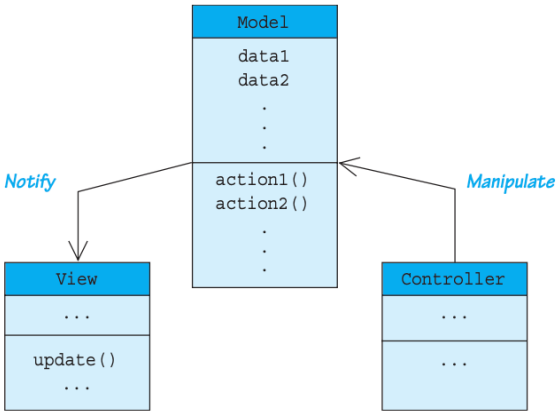
- Transforms one class into a different class
- Adds new interface without changing underlying class
- Example
  - Creating a stack from an array
  - Creating a queue from a linked list

페이지 14

어댑터 패턴은 기존 클래스를 변경하지 않고
새로운 인터페이스를 추가하여 다른 클래스로 변환합니다.
예를 들어, 배열에 스택 인터페이스를 추가하여
스택 데이터 구조를 만들 수 있습니다.
또 linked list에 queue operation을 추가하여
queue 데이터 구조를 만들 수 있습니다.

Model-View-Controller (MVC) Pattern

- Separates I/O tasks from the rest of the application
- Model: Core functionality
- View: Output display
- Controller: Input handling



페이지 15

MVC 패턴은 Model View Controller Pattern의 약어 입니다
이 패턴은 애플리케이션의 입출력 작업을 나머지 부분과 분리합니다
모델은 핵심 기능을, 뷰는 출력 표시를,
컨트롤러는 입력 처리를 담당합니다

MVC Pattern Example

- Model: Container class (e.g., array)
- View: Display of array element
- Controller: Commands to display specific index
- Suitable for GUI design projects

페이지 16

MVC 패턴의 간단한 예로,
모델은 array와 같은 컨테이너 클래스,
뷰는 배열 요소 표시,
컨트롤러는 특정 인덱스 표시 명령을 담당할 수 있습니다.
이 패턴은 특히 GUI 설계 프로젝트에 적합합니다.

Efficient Sorting Pattern

- Common pattern in efficient sorting algorithms
- Recursive approach
- Divide, sort, and recombine strategy

페이지 17

효율적인 sorting 알고리즘들은 대부분 유사한 패턴을 따릅니다.
이 패턴은 재귀적 접근법을 사용하여 리스트를 나누고, sorting한 후,
다시 합치는 전략을 사용합니다.

Divide-and-Conquer Sorting Pattern

```
/**
Precondition: Interval a[begin] through a[end] of a have elements.
Postcondition: The values in the interval have
been rearranged so that a[begin] <= a[begin+1] <= . . . <= a[end].
*/
public static void sort(Type[] a, int begin, int end) {
    if ((end - begin) >= 1) {
        int splitPoint = split(a, begin, end);
        sort(a, begin, splitPoint);
        sort(a, splitPoint+1, end);
        join(a, begin, splitPoint, end);
    }
    else {
         //else sorting one (or fewer) elements so do nothing.
    }
}
```

core of sorting pattern

페이지 18

이 슬라이드는 divide-and-conquer sorting pattern의 슈도 코드를 보여줍니다.
이 코드는 배열을 오름차순으로 정렬하는 메서드를 나타냅니다.
sorting 패턴의 핵심은 이 네 줄의 코드입니다.
split 메서드로 배열을 나누고,
재귀적으로 정렬한 후,
join 메서드로 다시 합칩니다.

Split and Join Methods

- Split: Rearranges and divides the array interval
- Join: Combines two sorted intervals
- Different implementations lead to different sorting algorithms
- Flexibility of the Pattern
  - Split method can be implemented in various ways
  - Simple division or more elaborate rearrangement
  - Adaptable to different sorting strategies

페이지 19

split 메서드는 배열 구간을 재배열하고 나누는 역할을 합니다.
join 메서드는 정렬된 두 구간을 합칩니다.
이 두 메서드의 구현 방식에 따라 다양한 정렬 알고리즘이 만들어집니다.
이 패턴의 유연성은 split 메서드의 다양한 구현 방식에 있습니다.
단순히 구간을 나누거나, 더 복잡한 재배열을 수행할 수 있어 다양한 정렬 전략에 적용할 수 있습니다.

Future of Design Patterns

- Evolving field in software engineering
- Many known patterns, more to be discovered
- Continuous development and refinement

페이지 20

디자인 패턴은 소프트웨어 공학의 발전하는 분야입니다.
현재 많은 패턴이 알려져 있지만,
앞으로 더 많은 패턴이 발견되고 정제될 것입니다.