서버 프로그램 구현

개발 환경 구축

- 개발 환경 구축
  - ✓ 개발환경을 구축하기 위해서는 해당 프로젝트의 목적과 구축 설계에 대한 명확한 이해가 필요하며 이에 맞는 하드웨어 및 소프트웨어의 선정이 이루어져야 한다.
  - ✓ 개발에 사용되는 제품들의 성능과 라이선스 그리고 사용 편의성 등에 대한 내용도 파악

- 개발 Hardware 환경 – 운영 환경과 유사하게 구축
  - ✓ Web Server: 클라이언트로부터 전송된 웹 요청을 처리하는 서버(Apache HTTP Server, MS IIS Server, Google Web Server 등)
  - ✓ Web Application Server: Web Server로부터 요청을 받아서 작업을 수행한 후 결과를 Web Server가 이해할 수 있는 결과로 변환해서 전송해주는 서버(Apache Tomcat Server, IBM WebSphere, Oracle Web Logic, Zeus, Windows 의 IIS 등)
  - ✓ DataBase Server: 데이터베이스 관리를 위한 서버(Oracle, HANA DB, MS-SQL, MySQL, DB2 등)
  - ✓ File Server: 데이터베이스에 저장하기 어려운 데이터를 파일로 저장하는 서버

개발 환경 구축

- Software
  - ✓ System Software
    - 운영체제(OS: Operation System): 하드웨어 운영을 위한 운영체제로서, Windows/Linux/UNIX 등의 환경으로 구성되는 데, 일반적으로 상세 소프트웨어 명세는 하드웨어를 제공하는 벤더(Vendor)에서 제공하며 Windows, Linux, UNIX(HPUX, Solaris, AIX) 등이 있다.
    - JVM
    - Web Server
    - Web Application Server
    - DBMS

개발 환경 구축

- Software
  - ✓ 개발 소프트웨어
    - 요구 사항 관리 도구: 요구 사항의 수집과 분석, 추적 등을 편리하게 도와주는 소프트웨어로 JIRA, Trello, JFeature 등
    - 설계 및 모델링 도구: UML을 지원하며 개발의 전 과정에서 설계 및 모델링을 도와주는 소프트웨어로 ArgoUML, StarUML, PlantUML, ER-Win, DB Designer 등
    - 구현 도구: 프로그램을 개발할 때 가장 많이 사용되는 도구로서 코드의 작성 및 편집, 디버깅 등 과 같은 다양한 작업이 가능한 IDE라고도 부르는 소프트웨어로 Eclipse, Visual Studio Code, IntelliJ, NetBeans 등 다양한 소프트웨어 도구들이 사용되고 있으며 구현해야 할 소프트웨어가 어떤 프로그래밍 언어로 개발되는지에 따라 선택하여 사용
    - 테스트 도구: 소프트웨어의 품질을 높이기 위해 테스트에 사용되는 소프트웨어 도구들로 코드의 테스트, 테스트에 대한 리포팅 및 분석 등의 작업이 가능한 소프트웨어로 사용되는 테스트 도구 들에는 xUnit, JUnit, CppUnit, Spring Test 등이 있다.
    - 형상관리 도구: 대다수의 프로젝트들은 다수의 개발자들로 구성된 팀 단위 프로젝트로 진행되며, 개발 자들이 작성한 소스 및 리소스 등 산출물에 대한 버전 관리를 위해 형상관리 도구가 사용되며 대표적인 형상관리 도구로는 CVS, Subversion, Git 등이 있다.
    - 빌드 도구: 개발자가 작성한 소스에 대한 빌드 및 배포를 지원하며, 프로젝트에서 사용되는 구성 요소들과 라이브러리들에 대한 의존성 관리를 지원하는 도구이며 Java에서 사용되는 빌드 도구에는 Ant(XML-자바 표준 빌드 도구), Maven(XML - pom.xml을 이용한 의존성 관리), Gradle(JSON - build.gradle을 이용한 의존성 관리) 등이 있다.
    - Group Ware: 개발에 참여하는 사람들이 서로 다른 작업 환경에서 프로젝트를 원할히 수행하도록 해주는 도구로 Jenkins, Slack, Zoom, TeamViewer 등

개발 환경 구축

- 개발 언어 선정 기준
  - ✓ 적정성:대상 업무의 성격, 즉 개발하고자 하는 시스템이나 응용 프로 그램의 목적에 적합
  - ✓ 효율성: 프로그래밍의 효율성이 고려
  - ✓ 이식성: 일반적인 PC 및 OS에 개발환경이 설치 가능
  - ✓ 친밀성: 프로그래머가 그 언어를 이해하고 사용 가능
  - ✓ 범용성: 다양한 과거 개발 실적이나 사례가 존재하고, 광범위한 분야에 사용
  - ✓ 위 고려 항목은 참고 사항으로 활용되며, 위 항목들 이외에도 알고리즘과 계산상의 난이도, 소프트웨어의 수행 환경, 자료 구조의 난이도, 개발 담당자와의 경험과 지식 등을 고려
- 패키지 소프트웨어
  - ✓ 기업에서 일반적으로 사용하는 여러 기능들을 통합하여 제공하는 소프트웨어를 의미
  - ✓ 소프트웨어를 구입하여 기업 환경에 적합하게 커스터마이징(Customizing)하여 사용
  - ✓ 기능 요구사항을 70% 이상 충족시키는 소프트웨어가 있을 때만 사용하는 것이 적합
  - ✓ 개발 조직을 갖추어야할 필요성이 없어 비용을 절감할 수 있고, 이미 개발된 소프트웨어를 사용하기 때문에 프로젝트 기간이 단축

소프트웨어 아키텍쳐

- Software Architecture
  - ✓ 소프트웨어의 골격이 되는 기본 구조
  - ✓ 소프트웨어를 구성하는 요소들 간의 관계를 표현하는 시스템의 구조
  - ✓ 소프트웨어 아키텍처는 애플리케이션의 분할 방법과 분할된 모듈에 할당될 기능, 모듈 간의 인터페이스 등 을 결정
  - ✓ 소프트웨어 아키텍처 설계의 기본 원리
    - 모듈화 (Modularity): 소프트웨어의 성능을 향상시키거나 시스템의 수정 및 재사용, 유지 관리 등이 용이하도록 시스템의 기능들을 모듈 단위로 나누는 것
    - 추상화(Abstraction): 문제의 전체적이고 포괄적인 개념을 설계한 후 차례로 세분화하여 구체화시켜 나가는 것
    - 단계적 분해 (Stepwise Refinement): 하향식 설계 전략으로 문제를 상위의 중요 개념으로부터 하위의 개념으로 구체화시키는 분할 기법
    - 정보 은닉(Information Hiding): 한 모듈 내부에 포함된 절차와 자료들의 정보가 감추어져 다른 모듈이 접근하거나 변경하지 못하도록 하는 기

소프트웨어 아키텍쳐

- Software Architecture
  - ✓ 품질 속성
    - 개념적 무결성: 전체 시스템과 시스템을 이루는 구성 요소들 간 일관성을 유지하는 것
    - 정확성, 완결성: 요구사항과 요구사항을 구현하기 위해 발생하는 제약사항들을 모두 충족시키는 것
    - 구축 가능성: 모듈 단위로 구분된 시스템을 적절하게 분배하여 유연하게 일정을 변경할 수 있도록 하는 것
    - 기타 속성: 변경성, 시험성, 적응성, 일치성, 대체성 등
  - ✓ 비즈니스 측면 속성
    - 시장 적시성: 정해진 시간에 맞춰 프로그램을 출시하는 것
    - 비용과 혜택
      - 개발 비용을 더 투자하여 유연성이 높은 아키텍처를 만들 것인지를 결정하는 것
      - 유연성이 떨어지는 경우 유지보수에 많은 비용이 소모될 수 있다는 것을 고려
    - 예상 시스템 수명:
      - 시스템을 얼마나 오랫동안 사용할 것인지를 고려하는 것
      - 수명이 길어야 한다면 시스템 품질의 '변경 용이성', '확장성'을 중요하게 고려
    - 기타 속성: 목표 시장, 공개 일정, 기존 시스템과의 통합 등

소프트웨어 아키텍쳐

- Software Architecture
  - ✓ 시스템 측면 속성
    - 성능: 사용자의 요청과 같은 이벤트가 발생했을 때, 이를 적절하고 빠르게 처리하는 것
    - 보안: 허용되지 않은 접근을 막고, 허용된 접근에는 적절한 서비스를 제공하는 것
    - 가용성: 장애 없이 정상적으로 서비스를 제공하는 것
    - 기능성: 사용자가 요구한 기능을 만족스럽게 구현하는 것
    - 사용성: 사용자가 소프트웨어를 사용하는데 헤매지 않도록 명확하고 편리하게 구현하는 것
    - 변경 용이성: 소프트웨어가 처음 설계 목표와 다른 하드웨어나 플랫폼에서 동작할 수 있도록 구현하는 것
    - 확장성: 시스템의 용량, 처리능력 등을 확장시켰을 때 이를 효과적으로 활용할 수 있도록 구현하는 것
    - 기타 속성; 테스트 용이성, 배치성, 안정성 등

소프트웨어 아키텍쳐

- Software Architecture
  - ✓ Architecture Pattern
    - 아키텍처를 설계할 때 참조할 수 있는 전형적인 해결 방식 또는 예제를 의미
    - 소프트웨어 시스템의 구조를 구성하기 위한 기본적인 윤곽을 제시
    - 서브시스템들과 그 역할이 정의되어 있으며, 서브시스템 사이의 관계와 여러 규칙·지침 등을 포함
  - ✓ Architecture Pattern 종류
    - 레이어 패턴(Layers Pattern)
      - 시스템을 계층(Layer)으로 구분하여 구성하는 고전적인 방법 중의 하나
      - 각각의 서브시스템들이 계층 구조를 이루며, 상위 계층은 하위 계층에 대한 서비스 제공자가 되고, 하위 계층은 상위 계층의 클라이언트가 되는 패턴
      - 대표적인 모델로 OSI 참조 모델이 있음
    - 클라이언트-서버 패턴(Client-Server Pattern)
      - 하나의 서버 컴포넌트와 다수의 클라이언트 컴포넌트로 구성되는 패턴
      - 사용자가 클라이언트를 통해 서버에 요청하고 클라이언트가 응답을 받아 사용자에게 제공하는 방식으로 서비스를 제공

소프트웨어 아키텍쳐

- Software Architecture
  - ✓ Architecture Pattern 종류
    - 파이프-필터 패턴(Pipe-Filter Pattern)
      - 데이터 스트림 절차의 각 단계를 필터 컴포넌트로 캡슐화하여 파이프를 통해 데이터를 전송하는 패턴
      - 컴포넌트는 재사용성이 좋고, 추가가 쉬워 확장이 용이하며, 재배치를 통해 다양한 처리 루틴을 구축하는 것이 가능하
      - 주로 데이터 변환, 버퍼링, 동기화 등에 사용
    - 모델-뷰-컨트롤러 패턴(Model-View-Controller Pattern) 또는 MVC 패턴(MVC Pattern)
      - 서브시스템을 3개의 부분으로 구조화하는 패턴으로, 각 부분은 별도의 컴포넌트로 분리되어 있으므로 서로 영향을 받지 않고 개발 작업을 수행
      - 한 개의 모델에 대해 여러 개의 뷰를 필요로 하는 대화형 애플리케이션에 적합
      - 각 서브시스템의 역할
        - 모델(Model) : 서브시스템의 핵심 기능과 데이터를 보관
        - 뷰(View) : 사용자에게 정보를 표시
        - 컨트롤러(Controller) : 사용자로부터 받은 입력을 처리해서 Model에 전달하고 처리한 결과를 View에게 전달

소프트웨어 아키텍쳐

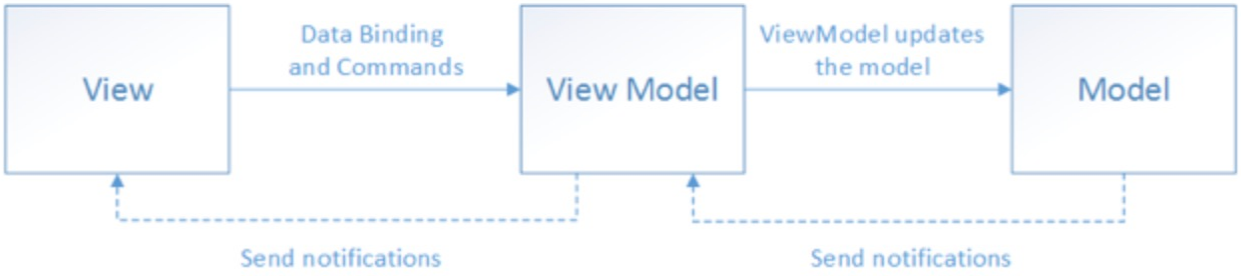
- ❖ Software Architecture
  - ✓ Architecture Pattern 종류
    - ● MVVM 패턴
      - ▪ MVVM 패턴의 목표는 비즈니스 로직과 프레젠테이션 로직을 UI로부터 분리하는 것
      - ▪ 비즈니스 로직과 프레젠테이션 로직을 UI로부터 분리하게 되면, 테스트, 유지 보수, 재사용이 수월
      - ▪ 3가지 구성요소
        - 모델 (Model)
        - 뷰 (View)
        - 뷰 모델 (View Model)
      - ▪ 뷰는 뷰 모델을 알지만 뷰 모델은 뷰를 알지 못하며 뷰 모델은 모델을 알지만 모델은 뷰 모델을 알지 못함
      - ▪ 이런 구조를 통해서 뷰 모델과 모델이 뷰로부터 독립적인 형태를 만들어서 위에서 말한 UI로부터 비즈니스 로직과 프레젠테이션 로직을 분리라는 목적을 이룰 수 있게 된 것

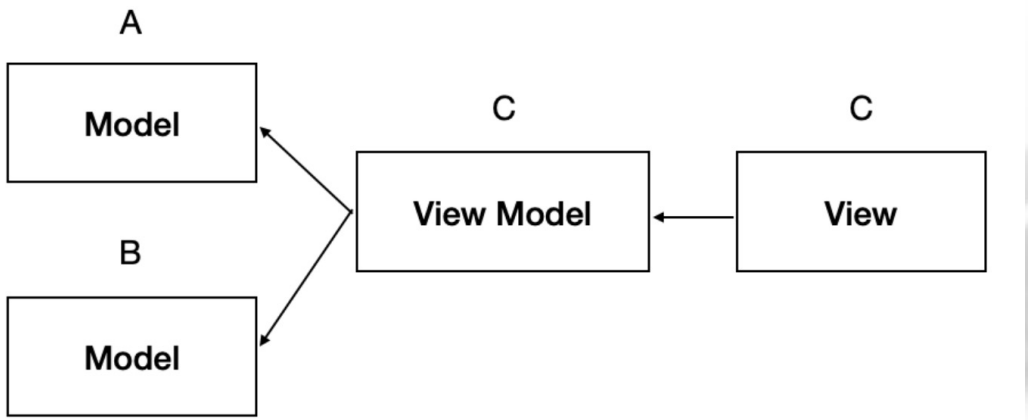
소프트웨어 아키텍쳐

- ❖ Software Architecture
  - ✓ Architecture Pattern 종류
    - ● MVVM 패턴
      - ▪ 뷰
        - UI에 관련된 것을 다루는 것으로 사용자가 스크린을 통해서 보는 것들에 대한 구조, 레이아웃, 형태를 정의하는 것
        - 애니메이션 같은 UI 로직을 포함하되 비즈니스 로직을 포함하지 않아야 함
      - ▪ 뷰 모델
        - 뷰 모델의 역할은 뷰가 사용할 메서드와 필드를 구현하고, 뷰에게 상태 변화를 알리는 것
        - 뷰는 뷰 모델의 상태 변화를 옵저빙
        - 뷰 모델에서 제공하는 메서드와 필드가 UI에서 제공할 기능을 정의
        - 뷰 모델과 모델은 일대다 관계를 형성
        - 뷰 모델은 뷰가 쉽게 사용할 수 있도록 모델의 데이터를 가공해서 뷰에게 제공
      - ▪ 모델
        - 비즈니스 로직과 유효성 검사와 데이터를 포함하는 앱의 도메인 모델
        - 앱에서 사용할 데이터에 관련된 행위와 데이터를 다룸

소프트웨어 아키텍쳐

- Software Architecture
  - ✓ Architecture Pattern 종류
    - MVVM 패턴





소프트웨어 아키텍쳐

- Software Architecture
  - ✓ Architecture Pattern 종류
    - 마스터-슬레이브 패턴 (Master-Slave Pattern)
      - 마스터 컴포넌트에서 슬레이브 컴포넌트로 작업을 분할한 후 슬레이브 컴포넌트에서 처리된 결과물을 다시 돌려받는 방식으로 작업을 수행하는 패턴
      - 장애 허용 시스템과 병렬 컴퓨팅 시스템에서 주로 활용
    - 브로커 패턴 (Broker Pattern)
      - 사용자가 원하는 서비스와 특성을 브로커 컴포넌트에 요청하면 브로커 컴포넌트가 요청에 맞는 컴포넌트와 사용자를 연결해주는 패턴
      - 분산 환경 시스템에서 주로 활용
    - 피어-투-피어 패턴 (Peer-To-Peer Pattern)
      - 피어(Peer)를 하나의 컴포넌트로 간주하며 각 피어는 서비스를 호출하는 클라이언트가 될 수도, 서비스를 제공하는 서버가 될 수도 있는 패턴
    - 이벤트 버스 패턴(Event-Bus Pattern)
      - 소스가 특정 채널에 이벤트 메시지를 발행(Publish)하면 해당 채널을 구독(Subscribe)한 리스너들이 메시지를 받아 이벤트를 처리하는 방식
    - 블랙보드 패턴 (Blackboard Pattern)
      - 모든 컴포넌트들이 공유 데이터 저장소와 블랙보드 컴포넌트에 접근하여 검색 을 통해 원하는 데이터를 찾을 수 있는 패턴
      - 음성 인식, 차량 식별, 신호 해석 등에 주로 활용

소프트웨어 아키텍쳐

- Software Architecture
  - ✓ Architecture Pattern 종류
    - ● 인터프리터 패턴(Interpreter Pattern)
      - ▪ 코드의 각 라인을 수행하는 방법을 지정하고, 기호마다 클래스를 갖도록 구성 되는 패턴
      - ▪ 특정 언어로 작성된 프로그램 코드를 해석하는 컴포넌트를 설계할 때 사용

소프트웨어 아키텍쳐

- Software Architecture
  - ✓ 설계 과정
    - 설계 목표 설정 : 시스템의 개발 방향을 명확히 하기 위해 설계에 영향을 주는 비즈니스 목표, 우선순위 등의 요구사항을 분석하여 전체 시스템의 설계 목표를 설정
    - 시스템 타입 결정 : 시스템과 서브시스템의 타입을 결정하고, 설계 목표와 함께 고려하여 아키텍처 패턴을 선택
    - 아키텍처 패턴 적용 : 아키텍처 패턴을 참조하여 시스템의 표준 아키텍처를 설계
    - 서브시스템 구체화 : 서브시스템의 기능 및 서브시스템 간의 상호작용을 위한 동작과 인터페이스를 정의
    - 검토 : 아키텍처가 설계 목표에 부합하는지, 요구사항이 잘 반영되었는지, 설계의 기본 원리를 만족하는지 등을 검토

객체 지향

- 객체 지향 프로그래밍: 소프트웨어의 모든 요소들을 객체로 만든 후 객체를 이용해서 프로그램을 개발해 나가는 방식
- 객체 지향 용어
  - ✓ 객체(Object): 동일한 목적을 위해 모인 데이터들과 이에 대한 연산을 가지는 것
  - ✓ 클래스(Class): 유사한 역할을 수행하는 객체들의 모임으로 사용자 정의 자료형
  - ✓ 인스턴스: 클래스를 기반으로 생성된 객체
  - ✓ 캡슐화(Encapsulation): 캡슐화는 객체 안에 데이터와 연산들을 패키지 화 한 것으로 정보 은폐를 통해 객체의 세부적 구현을 은폐하므로 변경 작업 시에 부작용의 전파를 최소화할 수 있으며 캡슐화 된 기능은 다른 클래스에서 재사용 가능성을 높임
  - ✓ 속성(Property): 각 객체가 가지고 있는 정보로서 객체의 성질, 분류, 식별, 수량 또는 상태 등을 표현
  - ✓ 메소드(method): 객체에 정의한 연산을 의미하며, 이것은 객체의 상태를 참조 및 변경하는 수단이며 객체가 메시지를 받으면 메소드를 수행
  - ✓ Message: 객체들은 message를 통해 정보를 교환하는데 객체와 객체 사이의 통신수단
  - ✓ 캡슐화 와 정보 은닉: 캡슐화는 데이터와 이에 대한 연산을 합한 것을 의미하며, 캡슐화의 장점은 정보 은닉(information hidden)을 의미하며 캡슐화 된 객체는 외부 인터페이스만을 통하여 접근하며 캡슐화를 하면 결합도는 낮아지고, 응집도는 높아지며, 변경 시의 부작용을 방지하며 정보 은폐를 통해 객체의 세부적 구현을 은폐하므로 변경 작업 시에 부작용의 전파를 최소화할 수 있고 캡슐화하여 처리하므로 인터페이스가 단순해짐

객체 지향

- 객체 지향 용어
  - ✓ 상속(Inheritance): 상위 클래스의 메소드에 존재하는 모든 속성을 하위 클래스가 계승하는 것으로서 하위 클래스는 상위 클래스의 메소드 및 모든 속성을 공유하며 소프트웨어 재사용 가능성을 높여주며 유지 보수를 편리하게 함
  - ✓ 상속하는 클래스를 상위 클래스 나 슈퍼 클래스 또는 기반 클래스라고 하며 상속받는 클래스를 하위 클래스나 서브 클래스 또는 파생 클래스 라고 함
  - ✓ Method Overloading: 하나의 클래스에 메소드의 이름이 같은 메소드가 존재하는 경우
  - ✓ Method Overriding: 상위 클래스와 하위 클래스에 동일한 원형의 메소드가 존재하는 경우
  - ✓ 다형성(Polymerphism): 동일한 메시지에 대하여 서로 다르게 반응하는 성질로 동일한 코드가 대입된 인스턴스에 따라 다른 메소드를 호출하는 것으로 상속과 메소드 오버라이딩으로 구현
- 연관성: 두 개 이상의 객체들이 상호 참조하는 관계
  - ✓ is member of(연관화): 2개 이상의 객체가 관련성이 있는 경우, 링크와 유사
  - ✓ is instance of(분류화): 동일한 형태의 특성을 갖는 객체들 사이의 관계
  - ✓ is part of(집단화): 관련있는 객체들을 모아서 하나의 객체를 구성하는 경우
  - ✓ is a
    - ❑ Generalization: 공통적인 성질들로 추상화한 상위 객체를 구성하는 것
    - ❑ Specialization: 상위 객체를 구체화하여 하위 객체를 구성하는 것

객체 지향

- 객체 지향 분석 및 설계
  - ✓ 객체 지향 분석: 사용자의 요구사항과 관련된 객체, 속성, 연산, 관계 등을 정의하여 모델링 하는 기법
  - ✓ 방법론
    - ❑ Rumbaugh 방법론: 분석 활동을 객체 모델(정보 모델 – 객체를 찾는 작업), 동적 모델 Wirfs(상태 다이어그램을 이용하여 동적인 행위를 표현), 기능 모델(DFD를 이용하여 프로세스들 간의 자료 흐름을 중심으로 처리 과정을 표현한 모델)로 나누어 수행
    - ❑ Booch 방법론: 미시적 개발 프로세스와 거시적 개발 프로세스를 모두 사용하는 방식
    - ❑ Jacobson 방법론: UseCase 강조
    - ❑ Coad & Yourdon 방법론: E-R Diagram 을 사용하여 객체의 행위를 모델링
    - ❑ Wirfs-Brock 방법론: 분석 과 설계 간의 구분이 없고 고객 명세서를 평가해서 설계 작업까지 연속적으로 수행하는 방법
  - ✓ 객체지향 설계 원칙
    - ❑ 단일 책임 원칙 – 객체는 단 하나의 책임만 가져야 함
    - ❑ 개방-폐쇄 원칙 – 기존의 코드를 변경하지 않고 기능을 추가
    - ❑ 리스코프 치환 원칙 – 하위 클래스는 상위 클래스의 기능을 수행할 수 있어야 함
    - ❑ 인터페이스 분리 원칙 – 자신이 사용하지 않는 인터페이스와 분리
    - ❑ 의존 역전 원칙 – 의존 관계 성립 시 추상성이 높은 클래스와 의존 관계를 맺어야 함

모듈

- 모듈
  - ✓ 독립적으로 수행 가능한 코드의 모임으로 서브 루틴 또는 서브 시스템이라고 함
  - ✓ 독립적으로 사용하는 메모리를 할당받아서 수행
  - ✓ 소프트웨어의 성능을 향상시키거나 시스템의 수정 및 재사용, 유지 관리 등이 용이하도록 시스템의 기능들을 모듈 단위로 분해하는 것이 모듈화
  - ✓ 모듈의 기능적 독립성은 소프트웨어를 구성하는 각 모듈의 기능이 서로 독립됨을 의미하는 것으로, 모듈이 하나의 기능만을 수행하고 다른 모듈과의 과도한 상호작용을 배제함으로써 이루어진다.
  - ✓ 독립성이 높은 모듈일수록 모듈을 수정하더라도 다른 모듈들에게는 거의 영향을 미치지 않으며, 오류가 발생해도 쉽게 발견하고 해결할 수 있다.
  - ✓ 모듈의 독립성은 결합도(Coupling)와 응집도(Cohesion)에 의해 측정되며 독립성을 높이려면 모듈의 결합도는 약하게 응집도는 강하게 모듈의 크기는 작게 만들어야 한다.

모듈

- 모듈
  - ✓ 응집도: 한 모듈 안의 기능들의 연관성 정도로 아래로 내려갈 수 록 낮아지며 좋지 못하다
    - 기능적 응집도 (Functional Cohesion): 모듈 내부의 모든 기능이 단일한 목적을 위해 수행되는 경우
    - 순차적 응집도 (Sequential Cohesion): 모듈 내에서 한 활동으로부터 나온 출력값을 다른 활동의 입력값 으로 사용하는 경우
    - 통신적 응집도 (Communication Cohesion): 동일한 입력과 출력을 사용하여 다른 기능을 수행하는 활동들이 모여 있을 경우
    - 절차적 응집도 (Procedural Cohesion): 모듈이 다수의 관련 기능을 가질 때 모듈 안의 구성 요소들이 그 기능을 순차적으로 수행할 경우
    - 시간적 응집도 (Temporal Cohesion): 연관된 기능이라기보다는 특정 시간에 처리되어야 하는 활동들을 한 모듈에서 처리하는 경우
    - 논리적 응집도 (Logical Cohesion): 유사한 성격을 갖거나 특정 형태로 분류되는 처리 요소들이 한 모듈에서 처리되는 경우
    - 우연적 응집도 (Coincidental Cohesion): 모듈 내부의 각 구성 요소들이 연관이 없을 경우

모듈

- 모듈
  - ✓ 결합도: 모듈과 모듈 간의 관련성 정도를 나타내며, 관련이 적을수록 모듈의 독립성이 높아져 모듈 간 영향이 적어지게 되며 아래로 내려갈 수록 결합도가 높음
    - 자료 결합도 (Data Coupling): 모듈 간의 인터페이스로 전달되는 파라미터를 통해서만 모듈 간의 상호 작용이 일어나는 경우
    - 스탬프 결합도 (Stamp Coupling): 모듈 간의 인터페이스로 배열이나 오브젝트(Object), 스트럭처(Structure) 등이 전달되는 경우
    - 제어 결합도 (Control Coupling): 단순 처리할 대상인 값만 전달되는게 아니라 어떻게 처리를 해야 한다는 제어 요소가 전달되는 경우
    - 외부 결합도 (External Coupling): 다수의 모듈이 모듈 밖에서 도입된 데이터, 프로토콜, 인터페이스 등을 공유할 때 발생하는 경우
    - 공통 결합도 (Common Coupling): 파라미터가 아닌 모듈 밖에 선언되어 있는 전역 변수를 참조하고 전역 변수를 갱신하는 식으로 상호 작용하는 경우
    - 내용 결합도 (Content Coupling) 다른 모듈 내부에 있는 변수나 기능을 다른 모듈에서 사용하는 경우
  - ✓ Fan-In: 자신을 호출하는 모듈의 수
  - ✓ Fan-Out: 자신이 호출하는 모듈의 수
  - ✓ Nassi- Schneiderman Chart
    - ❑ 논리의 기술에 중점을 두고 도형을 이용해 표현하는 방법
    - ❑ 화살표 없이 연속, 선택 및 다중 선택, 반복의 3가지 논리 구조로 표현

모듈

- 단위 모듈
  - ✓ 한 가지 동작을 수행하는 기능을 모듈로 구현한 것
- 공통 모듈
  - ✓ 정보 시스템 구축 시 자주 사용하는 기능들로써 재사용이 가능하게 패키지로 제공하는 독립된 모듈을 의미
  - ✓ 자주 사용되는 계산식이나 매번 필요한 사용자 인증과 같은 기능들이 공통 모듈로 구성될 수 있음
  - ✓ 준수해야할 성질
    - 정확성: 시스템 구현 시 해당 기능이 필요하다는 것을 알 수 있도록 작성
    - 명확성: 해당 기능을 이해할 때 중의적으로 해석하지 않도록 작성
    - 완전성: 시스템 구현을 위해 필요한 모든 것을 기술
    - 일관성: 공통 기능들 간 상호 충돌이 발생하지 않도록 작성
    - 추적성: 기능에 대한 요구사항의 출처, 관련 시스템 등의 관계를 파악할 수 있도록 작성
  - ✓ 재사용(Reuse): 목표 시스템의 개발 시간 및 비용 절감을 위하여 검증된 기능을 파악하고 재구성하여 시스템에 응용하기 위한 최적화 작업

모듈

- 공통 모듈
  - ✓ 재사용 범위에 따른 분류
    - 함수와 객체 재사용: 클래스(Class)나 함수(Function) 단위로 구현한 소스코드를 재사용
    - 컴포넌트 재사용: 컴포넌트(Component) 단위로 재사용하며, 컴포넌트의 인터페이스를 통해 통신
    - 애플리케이션 재사용: 공통된 기능을 제공하도록 구현된 애플리케이션과의 통신으로 기능 을 공유하여 재사용
  - ✓ 효과적인 모듈 설계 방안
    - 결합도는 줄이고 응집도는 높이는 형태로 재사용성을 높임
    - 복잡도와 중복성을 줄이고 일관성을 유지
    - 모듈의 기능은 예측 가능해야 하고 지나치게 제한적이면 안됨
    - 이해하기 쉬운 크기로 분해
    - 모듈 간의 계층적 관계를 정의하는 자료가 제시되어야 함

모듈

- 테스트 케이스
  - ✓ 테스트 케이스란 요구 사항을 준수하는지 검증하기 위하여 테스트 조건, 입력값, 예상 출력 값 및 수행한 결과 등 테스트 조건을 명세한 것
  - ✓ 구성 요소
    - 식별자(Identifier): 설명 항목 식별자, 일련번호
    - 테스트 항목(Test Item): 테스트할 모듈 또는 기능
    - 입력 명세(Input Specification): 입력값 또는 테스트 조건
    - 출력 명세(Output Specification): 테스트 케이스 실행 시 기대되는 출력값 결과
    - 환경 설정(Environmental Needs): 테스트 수행 시 필요한 하드웨어나 소프트웨어 환경
    - 특수 절차 요구(Special Procedure Requirement): 테스트 케이스 수행 시 특별히 요구되는 절차
    - 의존성 기술(Inter-case Dependencies): 테스트 케이스 간의 의존성
  - ✓ 테스트 프로세스 단계
    - 계획 및 제어
    - 분석 및 설계
    - 구현 및 실행
    - 평가
    - 완료

Code

- Code
  - ✓ 코드는 데이터의 사용 목적에 따라 식별, 분류, 배열하기 위한 숫자, 문자 혹은 기호
  - ✓ 3대 기능
    - 식별 기능: 데이터의 간의 성격에 따라 구분이 가능
    - 분류 기능: 특정 기준이나 동일한 유형에 해당하는 데이터를 그룹화 가능
    - 배열 기능: 의미를 부여하여 나열 가능

Code

- ❖ Code
  - ✓ 종류
    - 순차 코드(Sequence Code): 코드화 대상 데이터를 발생 순서, 크기 순서 등의 일정 기준에 따라 차례로 일련번호를 붙이는 방식.(가장 단순 → 변화가 적은 경우 적합)
    - 구분 순차 코드(블럭 코드): 순차 코드에 보완을 하여 분류 효과를 두드러지게 한 코드로서, 몇 개의 블록으로 나누어 각 블록에 의미를 갖게 하는 방법
    - 십진 분류 코드(DECIMAL CODE): 코드화 대상을 10진으로 분류하고 각각을 10진으로 중분류. 소분류 해가는 방법.(도서 분류에 이용)
    - 그룹 분류 코드(GROUP Classification): 코드화 대상을 소정의 기준에 따라 대분류, 중분류, 소분류로 구분하여 각 그룹 안에서 순차 번호를 부여하는 방법.
    - 표의 숫자식 코드(Significant digit code): 코드화 대상 항목의 중량, 면적, 용량, 거리 등의 물리적 수치를 직접 코드에 적용시키는 방법.
    - 연상 코드(Mnemonic code): 코드 값을 보면 어떤 대상을 의미하는가를 연상할 수 있도록 그 대상의 의미가 코드의 이름에 그대로 반영된 것.
    - 말미식 코드: 스스로는 코드의 기능을 갖지 못하고 다른 코드의 끝에 추가하여 분류의 기능을 부여한 코드
    - 약자식 코드: 관습상 또는 습관적으로 사용해 오던 단어의 약자

디자인 패턴

- Software Design Pattern
  - ✓ 비슷한 목적으로 사용되는 클래스의 모범 사례를 패턴으로 정리하려고 하는 것이 디자인 패턴
  - ✓ 프로그램의 세계에는 다양한 디자인 패턴이 존재
  - ✓ 유명한 것이 'GoF 디자인 패턴'이다. GoF(고프)란 'The Gang of Four'의 약자이며 에리히 감마, 리처드 헬름, 랄프 존슨, 존 블리시데스의 4명
  - ✓ 객체지향 프로그래밍에 도움이 되는 디자인 패턴을 도입해 《Design Patterns: Elements of Reusable Object Oriented Software》라는 제목의 책에 정리했으며 이 책에 소개된 23가지 디자인 패턴이 GoF 디자인 패턴
  - ✓ 디자인 패턴의 분류
    - 객체의 '생성'에 관한 패턴
    - 프로그램의 '구조'에 관한 패턴
    - 객체의 '행동'에 관한 패턴
  - ✓ 디자인 패턴은 정교한 클래스 구조의 패턴 집합이며 잘 이해해서 올바르게 적용하면 충실하고 알기 쉬운 클래스 구성을 생성할 수 있음

디자인 패턴

- ❖ 디자인 패턴의 장점
  - ✓ 재사용성이 높고 유연성 있는 설계가 가능
    - 프로그램을 처음부터 설계하면 만들어진 성과물의 품질이 설계자의 직관이나 경험 등에 따라 달라지지만 디자인 패턴을 도입하면 초보자도 고수들의 갈고 닦은 '지혜'를 이용한 설계가 가능
  - ✓ 의사 소통이 원할해짐
    - 디자인 패턴을 습득하지 않은 기술자에게 설계를 설명할 경우 '이런 클래스를 만들고, 이 클래스는 이런 역할을 갖고 있고.....'라고 끝없이 설명해야 함
    - 디자인 패턴을 습득하고 있는 기술자라면 '○○패턴으로 만들어!'라는 한마디로 끝남
    - 디자인 패턴을 습득하고 있는 기술자끼리라면 설계에 대해 상담할 때도 디자인 패턴의 이름으로 설계 개요에 대해 동의를 얻는것이 가능해 의사 소통이 원활
  - ✓ 디자인 패턴을 이해하고 개념을 익혀두면 자신이 직접 설계하는 경우에도 적용할 수 있어 설계의 수준을 높일 수 있음

생성과 관련된 패턴

- 싱글톤 패턴
  - ✓ 클래스가 하나의 객체만 생성할 수 있는 디자인 패턴
  - ✓ 시스템 전체에 공통적으로 적용되는 설정정보를 보관하는 관리자 클래스나 전역 변수를 소유한 클래스를 만들 때 이용
  - ✓ 적용 방법
    - 생성자를 private으로 생성
    - 자신의 타입으로 된 static 변수를 선언
    - static 변수가 null이면 생성해서 리턴하고 그렇지 않으면 그냥 리턴하는 static 메서드를 생성
  - ✓ jdk의 클래스 중에서 객체 생성 시 생성자를 호출하지 않고 static 메서드를 이용해서 객체를 리턴받는 클래스 중에는 singleton 패턴을 적용한 클래스가 있음

생성과 관련된 패턴

- Factory Pattern
  - ✓ 객체 생성을 자신의 생성자를 호출하지 않고 인스턴스 생성을 전문적으로 하는 클래스(하위 클래스)의 메서드를 이용해서 인스턴스를 생성하는 패턴
  - ✓ Factory pattern은 구현체 보다는 인터페이스 코드 접근에 좀더 용의하게 해 줌
  - ✓ Factory pattern은 클라이언트 클래스로부터 인스턴스를 구현하는 코드를 떼어내서 코드를 더욱 탄탄하게 하고 결합도를 낮춤
  - ✓ Factory pattern은 구현과 클라이언트 클래스들의 상속을 통해 추상적인 개념을 제공
  - ✓ 일관성을 유지해야 하는 관련된 일련의 인스턴스들을 생성하고자 할 때 사용하는 패턴
  - ✓ 프레임워크의 객체 생성이 주로 이 패턴을 이용 – URLConnection 클래스가 대표적

생성과 관련된 패턴

- 추상 팩토리 패턴
  - ✓ 동일한 주제의 다른 팩토리를 묶어주는 패턴
  - ✓ 인터페이스를 통해 서로 연관·의존하는 객체들의 그룹으로 생성하여 추상적으로 표현
- 빌더 패턴
  - ✓ 작게 분리된 인스턴스를 건축 하듯이 조합하여 객체를 생성하는 패턴
  - ✓ 생성(construction)과 표기(representation)를 분리해 복잡한 객체를 생성
  - ✓ 동일한 객체 생성에서도 서로 다른 결과를 만들어 낼 수 있음
- 프로토타입 패턴
  - ✓ 기존 객체를 복제함으로써 객체를 생성

구조와 관련된 패턴

- Decorator Pattern
  - ✓ 주입을 이용해서 클래스에 기능을 추가하는 구조로 생성자에서 필요한 기능을 매개변수로 넘겨받아서 구현
  - ✓ JDK의 I/O 패키지의 클래스들이 많이 사용
  - ✓ 윈도우는 기본적인 모양을 갖고 여기에 가로 스크롤 바를 가질 수 있고 그렇지 않을 수도 있으며 세로 스크롤 바를 가질 수 있고 그렇지 않을 수도 있음
  - ✓ 상황에 따라서 생성될 수 있는 윈도우의 모양이 선택적 옵션에 따라 여러 가지인 경우 Decorator Pattern을 이용해서 구현하는 경우가 있음
- Façade Pattern
  - ✓ 클래스 라이브러리 같은 어떤 소프트웨어의 다른 커다란 코드 부분에 대한 간략화 된 인터페이스를 제공하는 객체
  - ✓ 퍼사드는 공통적인 작업에 대해 간편한 메소드들을 제공
  - ✓ 퍼사드는 라이브러리 바깥쪽의 코드가 라이브러리의 안쪽 코드에 의존하는 일을 감소시켜줌
  - ✓ 바깥쪽의 코드가 퍼사드를 이용하기 때문에 시스템을 개발하는 데 있어 유연성이 향상
- 프록시 패턴(proxy pattern)
  - ✓ 일반적으로 프록시는 다른 무언가와 이어지는 인터페이스의 역할을 하는 클래스
  - ✓ 프록시는 어떠한 것(이를테면 네트워크 연결, 메모리 안의 커다란 객체, 파일, 또 복제할 수 없거나 수요가 많은 리소스)과도 인터페이스의 역할을 수행

구조와 관련된 패턴

- Adapter pattern
  - ✓ 현재 구현된 인터페이스의 메소드를 통해서 상속받은 클래스의 메소드를 호출하는 패턴
  - ✓ 인터페이스를 구현한 클래스의 경우 상속받은 클래스의 메소드를 직접 호출할 수 없는 경우가 있어서 인터페이스의 메소드를 이용해서 상속받은 클래스의 메소드를 호출하는 방식
- Composite pattern
  - ✓ 재귀적 구조를 쉽게 처리하기 위한 패턴
  - ✓ 파일 시스템에서 디렉토리 안에 디렉토리를 추가할 때 또는 디렉토리 안의 디렉토리를 재귀적으로 지우고자 할 때 사용할 수 있는 패턴
- Bridge Pattern
  - ✓ 구현부에서 추상층을 분리하여, 서로가 독립적으로 확장할 수 있도록 구성한 패턴
  - ✓ 기능과 구현을 두 개의 별도 클래스로 구현
- 플라이웨이트 패턴(Flyweight pattern)
  - ✓ 동일하거나 유사한 객체들 사이에 가능한 많은 데이터를 서로 공유하여 사용하도록 하여 메모리 사용량을 최소화하는 소프트웨어 디자인 패턴
  - ✓ 오브젝트의 일부 상태 정보는 공유될 수 있는데, 플라이웨이트 패턴에서는 이와 같은 상태 정보를 외부 자료 구조에 저장하여 플라이웨이트 오브젝트가 잠깐 동안 사용할 수 있도록 전달

행동과 관련된 패턴

- Command Pattern
  - ✓ 요청을 객체의 형태로 캡슐화하여 사용자가 보낸 요청을 나중에 이용할 수 있도록 메소드 이름, 매개변수 등 요청에 필요한 정보를 저장 또는 로깅, 취소할 수 있게 하는 패턴
  - ✓ 커맨드 패턴에는 명령(command), 수신자(receiver), 발동자(invoker), 클라이언트(client)의 네 개의 용어가 항상 따르며 커맨드 객체는 수신자 객체를 가지고 있으며, 수신자의 메서드를 호출하고, 이에 수신자는 자신에게 정의된 메서드를 수행
- 템플릿 메소드 패턴
  - ✓ 알고리즘의 골격을 미리 정의해두고 하위 클래스에서 필요한 부분만 재정의 해서 사용하는 패턴
  - ✓ 추상 클래스나 인터페이스를 이용해서 구현
- 책임 연쇄 패턴(Chain of responsibility)
  - ✓ 책임들이 연결되어 있어 내가 책임을 못 질 것 같으면 다음 책임자에게 자동으로 넘어가는 구조
- 상태 패턴 (State Pattern): 동일한 동작을 객체의 상태에 따라 다르게 처리해야 할 때 사용하는 디자인 패턴
- 기념품 패턴 (Memento Pattern): Ctrl + z 와 같은 undo 기능 개발할 때 유용한 디자인패턴. 클래스 설계 관점에서 객체의 정보를 저장

행동과 관련된 패턴

- Observer Pattern
  - ✓ 주기적으로 상태가 변환하는 인스턴스가 존재하는 경우 이 인스턴스의 상태가 바뀐 것을 감지하여 처리하도록 해주어야 하는 경우에 사용할 수 있는 패턴
  - ✓ 인스턴스의 상태가 변화한 것을 관찰하고 그 인스턴스 자신이 상태의 변화를 통지하는 구조를 제공하는 패턴
- Strategy Pattern
  - ✓ 공통된 부분과 서로 다른 부분을 찾아내서 공통된 부분과 다른 부분을 분리시켜서 객체를 생성할 때 주입(DI-Defendency Injection)을 이용해서 서로 다른 부분을 추가하는 구조
- 해석자 패턴 (Interpreter Pattern)
  - ✓ 문법 규칙을 클래스 화 한 구조를 갖는 SQL 언어나 통신 프로토콜 같은 것을 개발할 때 사용
- 반복자 패턴 (Iterator Pattern):
  - ✓ 반복이 필요한 자료구조를 모두 동일한 인터페이스를 통해 접근할 수 있도록 메서드를 이용해 자료구조를 활용할 수 있도록 해주는 패턴
- 방문자 패턴 (visitor Pattern): 각 클래스의 데이터 구조로부터 처리 기능을 분리하여 별도의 visitor 클래스로 만들어놓고 해당 클래스의 메서드가 각 클래스를 돌아다니며 특정 작업을 수행하도록 하는 것
- 중재자 패턴 (Mediator Pattern): 클래스간의 복잡한 상호작용을 캡슐화하여 한 클래스에 위임해서 처리 하는 디자인 패턴

업무 프로세스 확인

- 프로세스란 개인이나 조직이 한 개 이상의 정보 자원의 입력을 통해 가치 있는 산출물을 제공하는 모든 관련 활동들의 집합
- 프로세스 구성 요소
  - ✓ 프로세스 책임자 (Owner)
  - ✓ 프로세스 맵(Map): 구조적 분석 기법의 DFD 나 객체 지향 기법에서의 사용 사례 다이어그램 이용
  - ✓ 프로세스 Task 정의서
  - ✓ 프로세스 성과 지표
  - ✓ 프로세스 조직
  - ✓ 경영자의 리더십 (Leadership)

개발 지원 도구

- IDE(Integrated Development Environment): 개발에 필요한 편집기, 컴파일러, 디버거, 빌드 도구 등의 다양한 툴을 하나의 인터페이스로 통합해서 제공하는 환경
  - ✓ Eclipse
  - ✓ Visual Studio
  - ✓ X-Code
  - ✓ Android Studio
  - ✓ IDEA
- 빌드 도구
  - ✓ Ant
  - ✓ Maven
  - ✓ Gradle

Server Application의 구성

- Server Application 구성
  - ✓ Repository: 데이터 저장소
  - ✓ Domain Class: DTO 또는 VO: 저장소에서 가져온 또는 저장할 데이터를 표현하기 위한 클래스
  - ✓ SQL
  - ✓ DAO: 저장소와의 작업에 사용할 클래스
  - ✓ Service: 비지니스 로직을 처리할 클래스
  - ✓ Controller: 사용자의 요청을 처리할 로직과 결과를 출력할 뷰를 연결시켜 주는 클래스

서버 구성

- 서버 이중화
  - ✓ 동일한 기능을 갖는 서버를 여러 계층으로 구성하는 것
  - ✓ 목적
    - Failover(시스템 대체 작동):평소 사용하는 서버와 그 서버의 클론 서버를 가지고 있다가 사용 서버가 장애로 사용이 어렵게 되었을 경우 클론 서버로 그 일을 대신 처리하게 해서 무 정지 시스템을 구축하게 해 주는 것
    - LoadBalance(부하 균형):두 개 이상의 서버가 일을 분담처리 해 서버에 가해지는 부하를 분산시켜주는 것

서버 구성

- 서버 가상화
  - ✓ 서버 가상화는 소프트웨어 애플리케이션을 통해 물리적 서버를 여러 개로 분리된 고유한 가상 서버로 나누는 과정
  - ✓ 각 가상 서버는 자체 운영 체제를 독립적으로 실행
  - ✓ 서버 사용자로부터 서버 리소스를 숨기는 데 사용
  - ✓ 서버 리소스에는 운영 체제, 프로세서, 개별 물리적 서버의 수와 ID 등이 포함
  - ✓ 장점
    - 서버 가용성 증가
    - 운영 비용 절감
    - 서버 복잡성 제거
    - 애플리케이션 성능 향상
    - 더욱 빠른 워크로드 배포
  - ✓ 서버 가상화의 유형:
    - 완전 가상화
    - 반가상화
    - OS 수준 가상화

Framework

- 프레임워크
  - ✓ 효율적인 정보 시스템 개발을 위한 코드 라이브러리, 애플리케이션 인터페이스(Application Interface), 설정 정보 등의 집합으로서 재사용이 가능하도록 소프트웨어 구성에 필요한 기본 뼈대를 제공
  - ✓ 광의적으로 정보 시스템의 개발 및 운영을 지원하는 도구 및 가이드 등을 포함
- 프레임워크 특징
  - ✓ 모듈화 (modularity): 프레임워크는 인터페이스에 의한 캡슐화를 통해서 모듈화를 강화하고 설계와 구현의 변경에 따르는 영향을 극소화하여 소프트웨어의 품질을 향상시킨다.
  - ✓ 재사용성(reusability)
    - 프레임워크가 제공하는 인터페이스는 반복적으로 사용할 수 있는 컴포넌트를 정의할 수 있게 하여 재사용성을 높여 준다.
    - 프레임워크 컴포넌트를 재사용하는 것은 소프트웨어의 품질을 향상시킬 뿐만 아니라 개발자의 생산성도 높여 준다.
  - ✓ 확장성 (extensibility)
    - 프레임워크는 다형성(polymorphism)을 통해 애플리케이션이 프레임 워크의 인터페이스를 확장할 수 있게 한다.
    - 프레임워크 확장성은 애플리케이션 서비스와 특성을 변경하고 프레임워크를 애플리케이션의 가변성으로부터 분리함으로써 재사 용성의 이점을 얻게 한다.
  - ✓ 제어의 역흐름 (inversion of control – 제어의 역전): 프레임워크 코드가 전체 애플리케이션의 처리 흐름을 제어하여 특정한 이벤트가 발생할 때 다형성(Polymorphism)을 통해 애플리케이션이 확장한 메소드를 호출함으로써 제어가 프레임워크로부터 애플리케이션으로 거꾸로 흐르게 함

Framework

- 대표적인 프레임워크
  - ✓ Java – Spring, 전자 정부 프레임워크
  - ✓ JavaScript – Node.js
  - ✓ Python – Django
  - ✓ Ruby – Rails
  - ✓ PHP – Codeigniter, Laravel
  - ✓ Windows Programming – .Net Framework

DBMS

- DBMS 접속 기술
  - ✓ JDBC: Java 데이터베이스 연동 기술
  - ✓ ODBC: Windows에서 데이터베이스를 사용하기 위한 연동 기술
  - ✓ 데이터베이스 접속 프레임워크: SQL Mapper(MyBatis), ORM(Java 의 JPA – Hibernate, Apple의 Core Data, MS 의 LINQ 등)
- 동적 SQL: 상황에 따라 다르게 동작하는 SQL
- 데이터베이스 병행 수행
  - ✓ 클러스터링: 두 대 이상의 서버를 하나의 서버처럼 운영하는 기술로, 서버 이중화 및 공유 스토리지를 사용하여 서버의 고가용성을 제공
    - 고가용성 클러스터링 : 하나의 서버에 장애가 발생하면 다른 노드(서버)가 받아 처리하여 서비스 중단을 방지하는 방식
    - 병렬 처리 클러스터링 : 전체 처리율을 높이기 위해 하나의 작업을 로드 밸런서(Load Balancer)를 이용해 여러 개의 서버에서 분산하여 처리하는 방식

개발 보안

- 소프트웨어(SW) 개발 보안: SW 개발 과정에서 개발자의 실수, 논리적 오류 등으로 인해 SW에 내포될 수 있는 보안 취약점(vulnerability)의 원인, 즉 보안 취약점(weakness)을 최소화하고, 사이버 보안 위협에 대응할 수 있는 안전한 SW를 개발하기 위한 일련의 보안 활동
- 소프트웨어 개발 보안 가이드
  - ✓ 세션 통제: 세션의 연결과 연결로 인해 발생하는 정보를 관리하는 것
  - ✓ 입력 데이터 검증 및 표현
    - 프로그램 입력 값에 대한 검증 누락 또는 부적절한 검증, 데이터의 잘못 된 형식 지정으로 인해 발생할 수 있는 보안 약점으로 SQL 삽입, 자원 삽입, 크로스 사이트 스크립트 등 26개
  - ✓ 보안 기능: 보안 기능(인증, 접근 제어, 기밀성, 암호화, 권한 관리 등)을 적절하지 않게 구현 시 발생할 수 있는 보안 약점으로 부적절한 인가, 중요 정보 평문 저장(또는 전송) 등 24개
  - ✓ 시간 및 상태: 동시 또는 거의 동시 수행을 지원하는 병렬 시스템, 하나 이상의 프로세스가 동작하는 환경에서 시간 및 상태를 부적절하게 관리하여 발생할 수 있는 보안 약점으로 경쟁 조건, 제어문을 사용하지 않는 재귀 함수 등 7개
  - ✓ 에러 처리: 에러를 처리하지 않거나, 불충분하게 처리하여 에러 정보에 중요 정보(시 스템 등)가 포함될 때 발생할 수 있는 보안 약점으로 취약한 패스워드 요구 조건, 오류 메시지를 통한 정보 노출 등 4개
  - ✓ 코드 오류: 타입 변환 오류, 자원(메모리 등)의 부적절한 반환 등과 같이 개발자가 범할 수 있는 코딩 오류로 인해 유발되는 보안 약점으로 널 포인터 역참조, 부적절한 자원 해제 등 7개

개발 보안

- 소프트웨어 개발 보안 가이드
  - ✓ 캡슐화: 중요한 데이터 또는 기능성을 불충분하게 캡슐화하였을 때 인가되지 않는 사용자에게 데이터 누출이 가능해지는 보안 약점으로 제거되지 않고 남은 디버그 코드, 시스템 데이터 정보 노출 등 8개
  - ✓ API 오용: 의도된 사용에 반하는 방법으로 API를 사용하거나, 보안에 취약한 API를 사용하여 발생할 수 있는 보안 약점으로 DNS Lookup에 의존한 보안 결정, 널 매개 변수 미조사 등 7개

서버 프로그램 테스트

- 소프트웨어 테스트의 개념
  - ✓ 소프트웨어 테스트란 구현된 애플리케이션이나 시스템을 사용자의 요구 사항이 만족되었는지 확인하기 위하여 기능 및 비기능 요소의 결함을 찾아내는 활동
  - ✓ 소프트웨어 테스트의 원칙
    - 개발자가 자신이 개발한 프로그램 및 소스코드를 테스팅하지 않는다.
      - 일반적으로 개발자가 자신이 개발한 소스코드에 대해서는 자신이 테스팅을 할 경우 결함을 발견하는 것이 쉬운 일이 아니다.
    - 효율적인 결함 제거 법칙 사용(낚시의 법칙, 파레토의 법칙): 효율적으로 결함을 발견, 가시화, 제거, 예방의 순서로 하여 정량적으로 관리할 수 있어야 한다.
      - 낚시의 법칙: 낚시를 즐겨하는 사람들은 특정 자리에서 물고기가 잘 잡힌다는 사실을 경험적으로 알고 있다. 소프트웨어 제품의 결함도 특정 기능, 모듈, 라이브러리에서 결함이 많이 발견된다는 것이 소프트웨어 테스트에서의 낚시의 법칙이다.
      - 파레토의 법칙: 소프트웨어 제품에서 발견되는 전체 결함의 80%는 소프트웨어 제품의 전체 기능 중 20%에 집중되어 있다.
    - 완벽한 소프트웨어 테스팅은 불가능한데 단순한 애플리케이션이라도 테스트 케이스의 수는 무한대로 발생되기 때문에 완벽한 테스트는 불가능하다.
    - 테스트는 계획 단계부터 해야 하는데 소프트웨어 테스트는 결함의 발견이 목적이긴 하지만 개발 초기 이전인 계획 단계에 서부터 할 수 있다면 결함을 예방할 수 있다.

서버 프로그램 테스트

- 소프트웨어 테스트의 개념
  - ✓ 소프트웨어 테스트의 원칙
    - 살충제 패러독스(Pesticide Paradox): 동일한 테스트 케이스로 반복 실행하면 더 이상 새로운 결함을 발견할 수 없으므로 주기적으로 테스트 케이스를 점검하고 개선해야 한다.
    - 오류-부재의 궤변(Absence of Errors Fallacy): 사용자의 요구 사항을 만족하지 못한다면 오류를 발견하고 제거해도 품질이 높다고 말할 수 없다.
  - ✓ 소프트웨어 테스트의 명세
    - 테스트 결과 정리: 테스트가 완료되면 테스트 계획과 테스트 케이스 설계부터 단계별 테스트 시나리오, 테스트 결과까지 모두 포함된 문서를 일관성 있게 작성한다.
    - 테스트 요약 문서: 테스트 계획, 소요 비용, 테스트 결과에 의해 판단 가능한 대상 소프트웨어의 품질 상태를 포함한 요약 문서를 작성한다.
    - 품질 상태: 품질 상태는 품질 지표인 테스트 성공률, 발생한 결함의 수와 결함의 중요도, 테스트 커버리지 등이 포함된다.
    - 테스트 결과서: 테스트 결과서는 결함에 관련된 내용을 중점적으로 기록하며, 결함의 내용, 결함의 재현 순서를 상세하게 기록한다.
    - 테스트 실행 절차 및 평가: 단계별 테스트 종료 시 테스트 실행 절차를 리뷰하고 결과에 대한 평가를 수행하며, 그 결과에 따라 실행 절차를 최적화하여 다음 테스트에 적용한다.

Batch Program

- 배치 프로그램: 사용자와의 상호 작용 없이 일련의 작업들을 작업 단위로 묶어 정기적으로 반복 수행하거나 정해진 규칙에 따라 일괄 처리하는 것
- 배치 프로그램의 필수 요소
  - ✓ 대용량의 데이터를 처리할 수 있어야 한다.
  - ✓ 자동화: 심각한 오류 상황 외에는 사용자의 개입 없이 동작해야 한다.
  - ✓ 견고성: 유효하지 않은 데이터의 경우도 처리해서 비정상적인 동작 중단이 발생하지 않아야 한다.
  - ✓ 안정성: 어떤 문제가 생겼는지, 언제 발생했는지 등을 추적할 수 있어야 한다.
  - ✓ 성능: 주어진 시간 내에 처리를 완료할 수 있어야 하고, 동시에 동작하고 있는 다른 애플리케이션을 방해하지 말아야 한다.
- 배치 스케줄러(Scheduler)
  - ✓ 배치 스케줄러는 일괄 처리(Batch Processing)를 위해 주기적으로 발생하거나 반복적으로 발생하는 작업을 지원하는 도구
  - ✓ 배치 스케줄러의 종류
    - 스프링 배치(Spring Batch)
    - Quartz 스케줄러(Scheduler)
    - Cron

Batch Program

- 배치 프로그램 테스트
  - ✓ 디버그(Debug) 또는 디버깅(Debugging)은 컴퓨터 프로그램의 논리적인 오류(Bug)를 찾아내는 과정
  - ✓ 디버거는 디버그를 돕는 도구로 디버거는 디버깅을 하려는 코드에 중단점을 지정하여 프로그램 실행을 중단하고, 코드를 단계적으로 실행하여 저장된 값을 확인 할 수 있도록 지원한다.

Package Software

- 기업에서 일반적으로 사용하는 여러 기능을 통합하여 제공하는 소프트웨어

형상관리

- 형상관리
  - ✓ 형상관리(SCM: Software Configuration Management)란 소프트웨어의 개발 과정에서 발생하는 산출물의 변경 사항을 버전 관리하기 위한 일련의 활동
  - ✓ 특성
    - 소프트웨어 변경사항을 파악하고 제어하며, 적절히 변경되고 있는지에 대해 확인하여 해당 담당자에게 통보하는 작업
    - 형상관리는 프로젝트 생명주기의 전 단계에서 수행하는 활동이며, 유지 보수 단계 에서도 수행되는 활동
    - 형상관리를 함으로써 소프트웨어 개발의 전체 비용을 줄이고, 개발 과정에서 발생 하는 여러 가지 문제점 발생 요인이 최소화되도록 보증하는 것이 목적
  - ✓ 형상관리 절차
    - 형상 식별: 형상관리 대상을 식별하여 이름과 관리 번호를 부여하고, 계층(Tree) 구조로 구분하여 수정 및 추적이 용이하도록 하는 작업으로 베이스라인의 기준을 정하는 활동
    - 변경 제어: 식별된 형상항목의 변경 요구를 검토, 승인하여 적절히 통제함으로써 현재의 베이스라 인에 잘 반영될 수 있도록 조정하는 작업으로, 적절한 형상 통제가 이루어지기 위해서 는 형상통제위원회 승인을 통한 통제가 이루어질 수 있어야 한다.
    - 형상 상태 보고: 베이스라인의 현재 상태 및 변경 항목들이 제대로 반영되는지 여부를 보고하는 절차 이며 형상의 식별, 통제, 감사 작업의 결과를 기록 및 관리하고 보고서를 작성하는 작업
    - 형상 감사: 베이스라인의 무결성을 평가하기 위해 확인, 검증 과정을 통해 공식적으로 승인하는 작업

형상관리

- 형상관리
  - ✓ 형상관리(SCM: Software Configuration Management) 항목
    - 소프트웨어 공학 기반 표준과 절차: 방법론, WBS, 개발 표준
    - 소프트웨어 프로젝트 계획서
    - 소프트웨어 요구 사항 명세서
    - 소프트웨어 아키텍처, 실행 가능한 프로토타입
    - 소프트웨어 화면, 프로그램 설계서
    - 데이터베이스 기술서: 스키마, 파일 구조, 초기 내용
    - 소스코드 목록 및 소스코드
    - 실행 프로그램
    - 테스트 계획, 절차, 결과
    - 시스템 사용 및 운영과 설치에 필요한 매뉴얼
    - 유지보수문서: 변경 요청서, 변경처리보고서등