서버 프로그램 구현

1. 개발 환경 구축

▼ 구현될 시스템 요구 사항의 명확한 이해 필요

▼ 개발 도구 (사용 편의성, 성능, 라이선스), 서버 선정

- 구현 도구 - 코드 작성, 디버깅, 수정과 같은 작업 지원.
- 빌드 도구 - 작성한 코드의 빌드 및 배포 수행. 각각의 구성요소와 모듈에 대한 의존성 관리 지원
- 형상 관리 도구 - 코드와 리소스 등 산출물에 대한 버전 관리.
- 테스트 도구 - 코드의 기능 검증, 전체 품질 향상 위해 사용.

(1) 개발 환경 구성 요소

1) 하드웨어 개발 환경

▼ 서버 하드웨어 개발환경

- Web Server

HTTP 이용한 요청/응답 처리

웹 상의 정적 컨텐츠(HTML, CSS, JavaScript, Image)를 처리

Apache, Google Web Server

- WAS

사용자 요청 스레드 처리, 데이터베이스에 접속하여 SQL 쿼리문에 대한 결과 값 반환

동적 컨텐츠(Servlet, JSP) 처리

Tomcat

- DB Server

데이터 수집, 저장 용도

MySQL, Oracle, MS-SQL

- File Server

파일 저장 하드웨어로 물리 저장 장치를 활용한 서버

대용량 HDD, SSD 장치

▼ 클라이언트 하드웨어 개발 환경

- 서버 개발 환경에서 제공된 서비스를 사용하기 위해 **UI를 제공**
- 클라이언트 프로그램, 웹 브라우저, 모바일 앱, 모바일 웹

2) 소프트웨어 개발 환경

▼ 운영체제, 미들웨어, 데이터베이스 시스템 선정

- 운영체제
- 미들웨어

컴퓨터와 컴퓨터 간의 연결을 쉽고 안전하게 할 수 있도록 관리를 도와주는 소프트웨어

JVM

- DBMS - 데이터베이스 관리 소프트웨어

3) 형상 관리

▼ 개념

- 개발을 위한 전체 과정에서 발생하는 모든 항목의 **변경 사항을 관리**하기 위한 활동
- 산출물을 체계적으로 관리하여 SW의 가시성, 추적성, 무결성 등의 품질 보증을 보장

▼ 목적

- 프로젝트 생명주기 동안 제품의 무결성과 변경에 대한 추적성을 확보
- 프로젝트 변경 발생 시 처리하는 메커니즘 제공 (형상 관리대상 파악, 베이스라인 지정, 버전관리, 접근제어 등)

▼ 절차

형상 식별

- 형상 관리 대상 정의 및 식별
- 추적성 부여 위해 ID와 관리 번호 부여

형상 통제

- 변경요구 관리, 변경제어, 형상 관리 등 통제 지원
- 베이스라인에 대한 관리 및 형상 통제 수행 가능

형상 감사

- 소프트웨어 베이스라인의 무결성 평가
- 베이스라인 변경 시 요구 사항과 일치 여부 검토

형상 기록

- 소프트웨어 형상 및 변경 관리에 대한 각종 수행 결과 기록
- 형상결과 보고서 작성

(2) 개발 환경 구축 절차

- 통합 개발 환경 설치
- 형상 관리 도구 설치
- 빌드 도구 설치

2. 공통 모듈 구현

(1) 모듈

1) 개념

- 독립된 하나의 소프트웨어 OR 하드웨어 단위
- 모듈화를 통해 분리된 시스템의 각 기능들로 서브 프로그램, 서브 루틴, 단위 프로그램, 작업 단위와 같은 의미로 사용

2) 특징

- 독립성

모듈 수정 시에도 다른 모듈들에 영향을 거의 미치지 않고, 오류 발생 시에도 쉽게 해결 가능

결합도와 응집도에 의해 측정. 결합도 낮게, 응집도 높게, 모듈 크기 작게 만들어야 함

- 단독으로 컴파일할 수 있으며, 재사용 가능

3) 모듈화

- 성능 향상, 복잡한 시스템의 수정, 재사용, 유지 관리 등이 용이하도록 기능 단위의 모듈로 분해하는 설계 및 구현 기법
- 루틴, 메인 루틴, 서브 루틴

4) 공통 모듈 구현 개념

- 소프트웨어 개발에 있어 기능을 분할하고 추상화하여 성능 향상, 유지보수를 효과적으로 하기 위한 공통 컴포넌트 구현 기법
- 인터페이스 모듈, 데이터베이스 접근 모듈 등 필요한 공통 모듈을 구현
- 결합도 줄이고 응집도는 높인 공통 모듈 구현 권장

(2) 응집도

- 모듈의 독립성
- 모듈 내부 구성 요소 간 연관 정도
- 하나의 모듈은 하나의 기능을 수행
- 유형 (아래로 갈수록 높음→좋음)

우연적 응집도 - 연관 없음

논리적 응집도 - 유사한 성격을 갖거나 특정 형태로 분류되는 처리 요소들이 한 모듈에서 처리

시간적 응집도 - **특정 시간에** 처리되어야 하는 활동들을 한 모듈에서 처리

절차적 응집도 - 다수의 관련 기능을 가질 때 모듈 안의 구성 요소들이 그 기능을 **순차적으로 수행**

통신적 응집도 - **동일한 입력과 출력을** 사용하여 **다른 기능을 수행**하는 활동들이 모여 있음

순차적 응집도 - 모듈 내에서 한 활동으로부터 나온 **출력 값을 다른 활동이 사용**

기능적 응집도 - 모듈 내부의 기능이 **단일한 목적**을 위해 수행

(3) 결합도

- 모듈 내부가 아닌 **외부의 모듈과의 연관도 또는 모듈 간의 상호 의존성**
- 모듈 간의 관련성 측정
- 유형 (아래로 갈 수록 낮음→좋음)

내용 결합도 - 다른 모듈 내부에 있는 변수나 기능을 다른 모듈에서 사용

공통 결합도 - 전역 변수를 참조하고 갱신

외부 결합도 - 외부에서 도입된 데이터 포맷, 통신 프로토콜, 디바이스 인터페이스 공유

제어 결합도 - 제어 요소가 전달

스탬프 결합도 - 모듈 간의 인터페이스로 배열이나 객체, 구조 등이 전달

자료 결합도 - 모듈 간의 인터페이스로 전달되는 파라미터를 통해서만 모듈 간의 상호 작용 발생

(4) 대상

- 화면 모듈, 화면에서 입력 받은 데이터 처리를 위한 서비스 컴포넌트, 비즈니스 트랜잭션 컴포넌트 등
- 상세 설계된 공통 모듈, 환경 변수를 실제 프로그래밍 언어로 구현

(5) 구현 절차

▼ 절차

DTO/VO → SQL → DAO → Service → Controller → 화면 구현

- DTO (Data Transfer Object) - 간단한 엔티티. 프로세스 사이에서 데이터를 전송하는 객체로 데이터 저장, 회수 외에 다른 기능이 없는 객체
- VO (Value Object) - 고정 클래스를 가지는 객체
- DAO (Data Access Object) - 특정 타입의 데이터베이스에 추상 인터페이스를 제공하는 객체, 세부 내용 노출 없이 데이터를 조작
- Service - 사용자의 요청을 처리하는 기능을 제공하기 위한 로직 구현, DAO 클래스를 통해 DB 연동을 처리하는 기능을 수행하는 클래스

▼ **MVC 패턴**

- Model

애플리케이션이 무엇을 할 것 인지를 정의

내부 비즈니스 로직을 처리

- View

화면에 무엇 인가를 보여주기 위한 역할

모델, 컨트롤러가 보여주려고 하는 것들을 화면에 처리

- Controller

모델이 어떻게 처리 할지를 알려주는 역할

뷰에 명령을 보내어 화면 요청 결과를 전달

(6) 팬 인 및 팬 아웃

- 모듈을 계층적으로 분석
- 시스템의 복잡도 측정
- 시스템 복잡도 최적화 위해 팬 인은 높게, 팬 아웃은 낮게 설계

1) 팬 인(Fan-In)

- 어떤 모듈을 제어(호출)하는 모듈의 수
- 모듈 자신을 기준으로 모듈에 들어오면 팬 인
- 높으면 재사용 측면에서 설계가 잘 되었지만 단일 장애점 발생 가능
- 높으면 관리 비용 및 테스트 비용 증가

2) 팬 아웃(Fan-Out)

- 어떤 모듈에 의해 제어(호출)되는 모듈의 수
- 모듈 자신을 기준으로 모듈에서 나가면 팬아웃
- 높으면 불필요한 모듈 호출 여부 검토 필요
- 높으면 단순화 여부 검토 필요

(7) 테스트

- IDE 도구 활용하여 개별 공통 모듈에 대한 디버깅 수행

▼ JUnit을 활용하여 테스트 코드 구현

- Annotation - 주석을 달아 특별한 의미를 부여한 메타데이터의 일종. @기호 앞에 사용.

@Test, @Before, @After, @BeforeClass, @AfterClass, @Ignore

- Assert Method

assertEuals(a,b,c); - a와 b가 일치함 확인. a: 예상 값, b: 결과값, c: 오차 범위

assertSame(a,b); - 객체 동일 여부

assertTrue(a); - 조건 a가 참인지 여부

assertNotNull(a); - 객체 a가 null이 아님 확인

assertArrayEquals(a,b)

▼ 테스트 종류

- 화이트박스 테스트

응용 프로그램의 내부 구조와 동작을 검사

소스 코드를 보면서 테스트 케이스를 작성하여 수행

- 메서드 기반 테스트

공통 모듈의 외부에 공개된 메서드 기반의 테스트

메서드에 서로 다른 파라미터 값을 호출 하면서 테스트

- 화면 기반 테스트

화면 단위로 단위 모듈을 개발한 후 화면에 직접 데이터를 입력하여 테스트

사용자의 시나리오에 기반한 공통 모듈 테스트 가능

- 테스트 드라이버 / 스텁

테스트 드라이버 - 하위 모듈은 있지만 상위 모듈은 없는 경우 사용

테스트 스텁 - 상위 모듈은 있지만 하위 모듈은 없는 경우 사용

3. 서버 프로그램 구현

(1) 개념

- 업무 프로세스를 기반으로 개발 언어와 도구를 이용해 서비스 제공에 필요한 업무 프로그램 구현

(2) 절차

- 백엔드와 프론트엔드를 구분하여 구현
- 백엔드 - DTO/VO 구현 → SQL 구현 → DAO 구현 → Service 구현 → Controller 구현
- 프론드엔드 - 화면 구현

(3) 세부 구현

1) DTO/VO 구현

화면에서 전달 받은 정보로 데이터베이스에 저장하는 객체 구현

```
public class JoinVO{
  String id;
  String pw;
  String name;

  public void setId(String id) {
    return id = id;
  }
  public void getId() {
    return id;
  }
  .....
}
```

2) SQL 구현

VO에서 정의한 객체 정보에 맞춰 정보가 저장될 테이블 정보 생성

```
CREATE TABLE CUSTOMER
(
  ID VARCHAR(20) NOT NULL COMMENT '아이디',
  PW VARCHAR(20) COMMENT '비밀번호',
  NAME VARCHAR(10) COMMENT '이름',
```

```
  PRIMARY KEY(PK_ID)
) COMMENT '회원정보';
```

```
<mapper namespace="com.dobi.sql">
  <!-- 회원 이름을 통해 회원 테이블 조회 -->
  <select id="selectJoin"
          parameterType="com.dobi.vo.JoinVO"
          resultType="com.dobi.vo.JoinVO">
  SELECT * FROM CUSTOMER
  WHERE (name=#{name})
  </select>

  <!-- 아이디, 비밀번호, 이름 회원 테이블에 저장-->
  <insert id="insertJoin" parameterType="com.dobi.vo.JoinVO">
    INSERT INTO CUSTOMER values (#{id}, #{pw}, #{name})
  </insert>
</mapper>
```

3) DAO 구현

DAO를 통해 SQL을 구현한 XML id 호출하여 조작 수행

```
public JoinDAO() {
  ....
}
// 회원가입 데이터 조회 프로토 타입
public int selectJoin(joinVO vo) throws Exception {
  return sqlSession.selectJoin("com.dobi.sql.selectJoin", joinVO);

// 회원가입 데이터 입력
public void insertJoin(joinVO vo) {
  sqlSession.insertJoin("com.dobi.sql.insertJoin", joinVO);
}
```

4) Service 구현

JoinDAO 호출 (회원 등록)

```
@Service
public class JoinService implements IMemberService {
  @Autowired
  JoinDao dao;

  @Override
  public void insertJoin(JoinVO join) {
    JoinVO member = dao.selectJoin(join);
    dao.insertJoin(member);
  }
}
```

5) 컨트롤러 구현

메인 로직 구현 (회원 가입 단위 모듈)

```
public class CreateController extends HttpServlet {
  @Override
  protected void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
```

```
    JoinVO vo = new JoinVO();
    JoinDAO dao = new JoinDAO();
    String result;

    vo.setId(req.getParameter("id);
    .....
    try {
      ...
    }
    catch (ParseException e) {
      ...
    }
}
```

6) 입/출력 검증 로직 구현

회원가입 성공/실패 시 메시지 호출 검증 로직

```
public class CreateController extends HttpServlet {
  @Override
  protected void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
    result = dao.insertJoin(vo);
    if(result=="success") {
    }
    else{
    }
}
```

4. 배치 프로그램 구현

(1) 개념

사용자와의 상호작용 없이 **일련의 작업들을 작업 단위**로 **묶어 정기적으로** 반복 수행하거나 정해진 규칙에 따라 **일괄 처리**하는 방법

(2) 필수 요소

- 정기 배치 - 정해진 시점에 정기적으로 실행
- 이벤트 배치 - 사전에 정의해 둔 조건 충족 시 자동으로 실행
- 온디맨드 배치 - 사용자의 명시적 요구가 있을 때마다 실행

(3) 배치 스케줄러

1) 개념

일괄 처리를 위해 주기적으로 발생하거나 반복적으로 발생하는 작업을 지원하는 도구

2) 종류

- 스프링 배치 - 스프링 프레임워크의 DI, AOP, 서비스 추상화 등 요소를 모두 사용할 수 있는 **대용량 처리를 제공하는 스케줄러**
- 쿼츠 스케줄러 - 스프링 프레임워크에 플러그인되어 수행하는 **작업**과 실행 스케줄을 정의하는 **트리거**를 분리하여 유연성 제공하는 **오픈 소스 기반 스케줄러**

3) Cron 표현식

작업이 실행되는 시간 및 주기 등을 설정할 때 이를 통해 배치 수행 시간 설정

초,분,시,일,월,요일,연도

* 모든수, ? 해당 항목 미사용 / 시작 시간과 반복 간격 설정, - 기간 설정

(4) 설계

- 프로그램 관리 대장 확인 - 구현해야 할 배치 프로그램 기능 확인
- 배치 설계서 확인 - 작업 내역을 참고하여 배치 프로그램 구현

(5) 작성

1) DTO, VO 구현

데이터 베이스에 저장할 객체를 구현

2) SQL 구현

SQL 작성

3) DAO 구현

DAO를 통해 SQL을 구현한 XML id 호출하여 조작 수행

4) 서비스 클래스 구현

DAO 호출하는 서비스를 선언 및 구현

5) 스케줄러 등록

작성한 배치 프로그램을 정기적으로 실행하는 쿼츠 스케줄러 등록