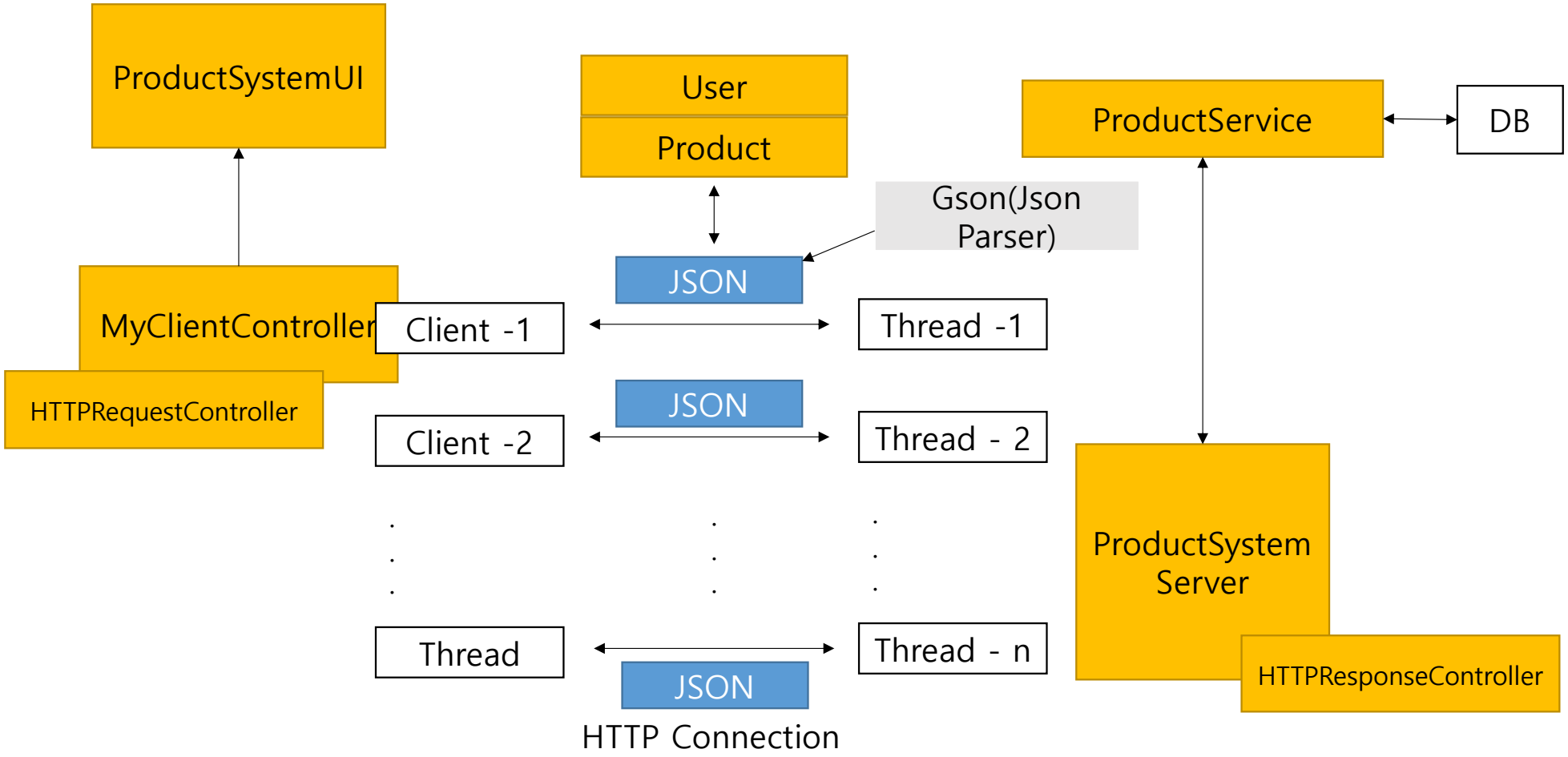
택배 관리 시스템 Client/Server



클래스 설명
Data Class

Product Class

- 택배 상품 정보를 담는 클래스
- 상품 번호, 등록한 관리자 번호, 상품명, 배송 상태, 생성 시간으로 구성
- API 형식에 맞게 생성자 구성 요구됨

```
public class Product {
    private Long order_id;
    private String admin_id;
    private String name;
    private String status;
    private String created_at;
```

User Class

- 로그인시 유저 정보 담는 클래스
- 고유 번호로만 구성

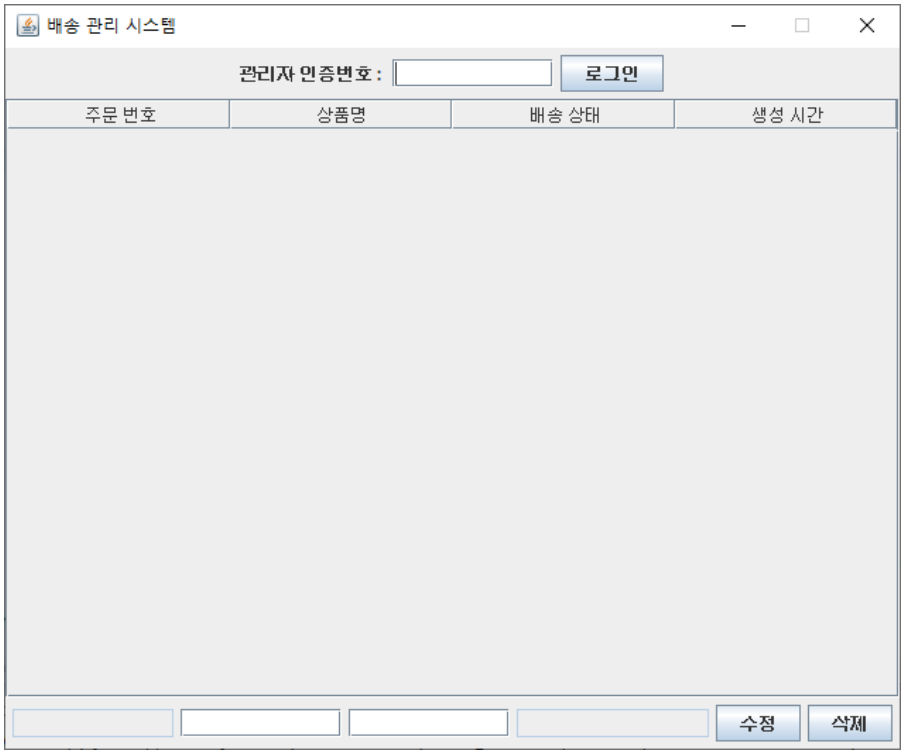
```
public class User {
    private String pw;
```

클래스 설명
Client Side

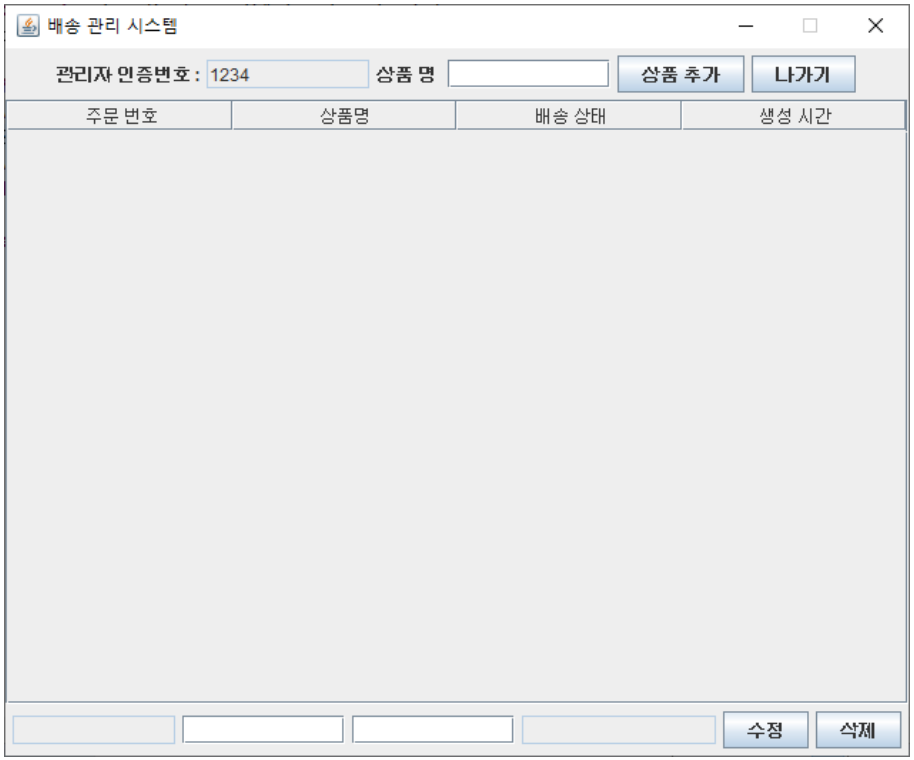
ProductSystemUI Class

- 화면 구성요소 정의, 레이아웃 이용하여 컴포넌트 배치
- 주요 메서드
  - ProductSystemUI() (생성자)
    - 화면을 구성하는 컴포넌트 초기화 및 레이아웃 배치 등
  - void addButttonActionListener
    - 이벤트 핸들러 등록 메서드로, 모든 버튼의 이벤트 핸들러 등록

ProductSystemUI Class – 예시 화면



로그인 전



로그인 후

ClientController Class

- Client의 메인
  - UI와 연결 및 서버 API 호출
- 주요 메서드 (API 요청 관련)
  - void login()
    - 로그인 요청
  - void getProducts()
    - 택배 상품 조회 요청
  - void addProduct()
    - 새로운 택배 추가 요청
  - void updateProduct()
    - 기존 택배의 정보(이름, 상태) 변경 요청
  - void deleteProduct()
    - 택배 삭제 요청

ClientController Class

- 주요 메서드
  - void appMain()
    - 컨트롤러 클래스 메인 로직, UI에서 발생한 이벤트를 위임 받아 처리
    - 각 버튼 클릭시 API 요청하도록 설정

HTTPRequestController Class

- Request를 세팅하고 보내는 과정을 모듈화한 클래스
- 주요 메서드
  - void setLoginRequest()
    - 로그인 API request 세팅 및 보내는 역할
  - void setGetRequest()
    - 상품 조회 API request 세팅 및 보내는 역할
  - void setPostRequest()
    - 상품 추가 API request 세팅 및 보내는 역할
  - void setPutRequest() 또는 void setPatchRequest()
    - 상품 수정 API request 세팅 및 보내는 역할
  - void setDeleteRequest()
    - 상품 삭제 API request 세팅 및 보내는 역할

클래스 설명
Server Side

ProductSystemServer Class

- 여러 클라이언트의 요청 핸들링할 수 있는 HTTP 서버 구현
- 주요 메서드
  - void start()
    - 서버의 메인 실행 메소드, serversocket 실행하고 클라이언트 연결 및 스레드 생성/처리
  - Class HTTPHandler
    - 각 클라이언트에서 요청한 API 핸들링하고 적합한 Service 메서드 호출

ProductService Class

- DB와 직접적으로 통신하는 class
- 서버의 비즈니스 로직을 담당
- 주요 메서드
  - void connectDB()
    - DB와 연결하는 작업 수행
  - void getProducts()
    - DB에 접근하여 택배 상품 조회 작업 수행 및 결과 생성
  - void addProduct()
    - DB에 접근하여 택배 추가 작업 수행 및 결과 생성
  - void updateProduct()
    - DB에 접근하여 기존 택배의 정보(이름, 상태) 변경 작업 수행 및 결과 생성
  - void deleteProduct()
    - DB에 접근하여 기존 택배 삭제 작업 수행 및 작업 결과 및 작업 결과 생성
  - void setDefaultResponse()
    - 등록되지 않은 형식의 API 호출 시 응답 결과 생성

HTTPResponseController Class

- Response를 세팅하고 보내는 과정을 모듈화한 클래스
- Service에서 결과를 리턴하는 역할을 하는 클래스
- 주요 메서드
  - void setSuccessLoginResponse()
    - 로그인 성공시 response 세팅하고 보내는 역할
  - void setSuccessGetResponse()
    - 택배 상품 조회 성공시 response 세팅하고 보내는 역할
  - void setSuccessPostResponse ()
    - 택배 추가 작업 성공시 response 세팅하고 보내는 역할
  - void setSuccessPutResponse () 또는 void setSuccessPatchResponse()
    - 택배 수정 작업 성공시 response 세팅하고 보내는 역할
  - void setSuccessDeleteResponse ()
    - 택배 삭제 작업 성공시 response 세팅하고 보내는 역할
  - void setFailedResponse()
    - 잘못된 API 호출 시 response 세팅하고 보내는 역할

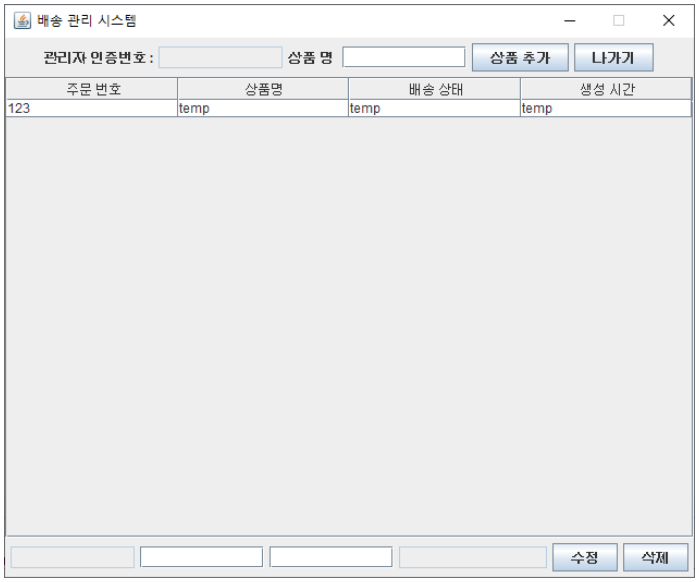
단계별 문제

Step1. Client UI 구현

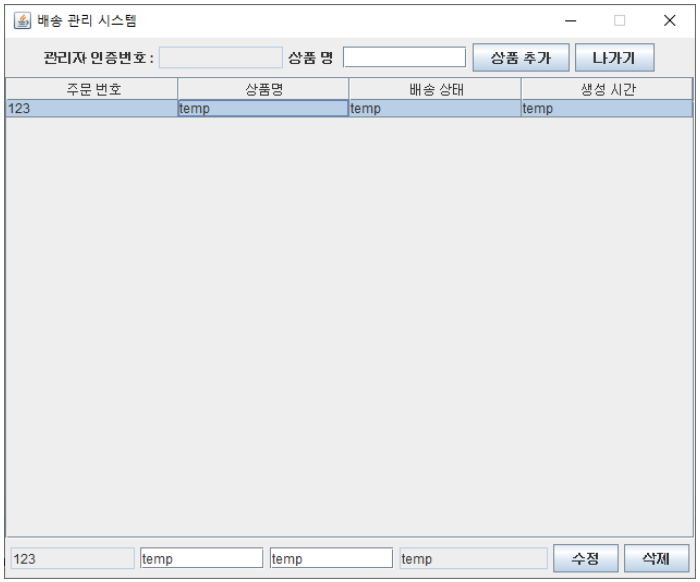
STEP1 : 클라이언트 UI 구성

- 다음 슬라이드에서 기술된 요구사항을 파악하고 ProductSystemUI를 완성하시오
- 생성자 ProductSystemUI() 의 table mouseClicked 이벤트와 main 메서드의 각 코멘트(//TODO)로 기술된 사항을 바로 아래에 추가하면 됨(각 코멘트 별 코드는 여러 줄일 수도 있고 한 줄인 것도 있음)

STEP1 : 완성 결과



로그인 후 화면



테이블 로우 클릭 시 화면

Step2. Client Side 구현

STEP2 : HTTPRequestController 완성

- 다음 슬라이드에서 기술된 요구사항을 파악하고 HTTPRequestController를 완성하시오
- setLoginRequest() 메서드를 참고하여 각 메서드의 각 코멘트(//TODO)로 기술된 사항을 바로 아래에 추가하면 됨(각 코멘트 별 코드는 여러 줄일 수도 있고 한 줄인 것도 있음)
- 필수헤더를 꼭 포함하여 구성하시오.
- 필수헤더
  - HTTP 메서드, API URI, HTTP 버전정보
    - ex) GET / HTTP/1.1
  - RequestBody가 있을 경우 Content-Type, Content-Length
    - ex) Content-Type : application/json; charset=utf-8
    - Ex) Content-Length : 10
- 참고) 수정에는 2가지 메서드 존재하는데 우선 PUT 메서드만 구현하고 PUT이 동작하지 않을 경우 PATCH 메서드 작성하여 사용

STEP2 : MyClientController 완성

- 다음 슬라이드에서 기술된 요구사항을 파악하고 MyClientController를 완성하시오.
- login() 메서드를 참고하여 각 메서드의 각 코멘트(//TODO)로 기술된 사항을 바로 아래에 추가하면 됨(각 코멘트 별 코드는 여러 줄일 수도 있고 한 줄인 것도 있음)
- Response를 받는 파트 작성
- 올바른 Response 일때 만 Response 이벤트 처리하도록
  - 올바른 Response 체크 => Response status code + HTTP 프로토콜 체크
- 로그인은 본인 학번으로 수행하여 테스트

STEP2 : MyClientController 완성

- appMain()에 STEP 1에 적용했던 이벤트 추가하기
  - 상품 업데이트/삭제 시 텍스트 박스 비우기
- 결과
  - 본인 학번으로 로그인 후
  - 상품 조회/추가/수정/삭제가 모두 정상 동작해야 함

Step3. Server Side 구현

STEP3 : HTTPResponseController

- 다음 슬라이드에서 기술된 요구사항을 파악하고 HTTPResponseController 를 완성하시오.
- setSuccessLoginResponse 메서드를 참고하여 각 메서드의 각 코멘트(//TODO)로 기술된 사항을 바로 아래에 추가하면 됨(각 코멘트 별 코드는 여러 줄일 수도 있고 한 줄인 것도 있음)
- 필수 헤더
  - HTTP 버전 정보, Status Code, Status message
    - EX) HTTP/1.1. 200 OK
    - Status code,message 는 200 OK, 201 Created 중에 사용
  - ResponseBody가 있을 경우 content-Type, content-Length

STEP3 : ProductService

- 다음 슬라이드에서 기술된 요구사항을 파악하고 ProductService를 완성하시오.
- 본인 DB를 연결해서 서버를 구성하시오.
- DB는 Product.class의 데이터 형태에 맞춰 Products 테이블을 구성하시오.
- 본인 DB 정보에 맞는(dbURL, dbUser, dbPassword) 입력
- pw[]에는 본인 학번 넣어줌
- Login() 메서드를 참고하여 //TODO 부분을 완성하시오

STEP3 : DB 예시

Column	Type	Default Value	Nullable
order_id	bigint		NO
name	varchar(100)		NO
status	varchar(45)		YES
created_at	datetime	CURRENT_TIMESTA...	NO
admin_id	varchar(45)		NO

Products Table 구조

```
CREATE TABLE `products` (
  `order_id` bigint NOT NULL AUTO_INCREMENT,
  `name` varchar(100) NOT NULL,
  `status` varchar(45) DEFAULT NULL,
  `created_at` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `admin_id` varchar(45) NOT NULL,
  PRIMARY KEY (`order_id`)
) ENGINE=InnoDB AUTO_INCREMENT=18 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
```

DDL 예시

STEP3 : ProductSystemServer

- 다음 슬라이드에서 기술된 요구사항을 파악하고 ProductService를 완성하시오.
- HTTPHandler의 call()과 ProductSystemServer의 start()의 //TODO로 처리된 부분 작성
- call() 파트 – 분기 처리 완성
- start() 파트 – thread로 handling하는 파트 완성