**REPORT**

**음식점 매니지먼트 시스템 (RMS)**



|  |  |
| --- | --- |
| **제출일자** | 2022년 06월 11일 (토) |
| **교과목** | 소프트웨어설계공학 |
| **담당교수** | 장희숙 교수님 |
| **학과** | 컴퓨터소프트웨어공학과 |
| **팀 이름** | 짹짹이 |
| **이름(학번)** | 조진혁 (20203120) 이수찬 (20172128)  최인수 (20173236)  박상현 (20183145) |
| **학년** | 3학년 |

**목차**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1. **개요** | ------------------------------- | p.01 |  | **5.4 매출 관리** | ----------------------- | p.38 |
| **1.1 프로젝트 개요** | ----------------------- | p.01 |  | **5.5 직원 등록** | ----------------------- | p.47 |
| **1.2 주제 선정 이유** | ----------------------- | p.02 |  | **5.6 직원 근퇴 변경** | ----------------------- | p.56 |
| **1.3 적용 패턴** | ----------------------- | p.03 |  | **6. 다이어그램** | ------------------------------- | p.65 |
| **1.4 정의, 약어** | ----------------------- | p.03 |  | **6.1 UseCase 다이어그램** | ----------------------- | p.65 |
| 1. **자원 및 일정 예측** | ----------------------- | p.04 |  | **6.2 Sequence 다이어그램** | ----------------------- | p.71 |
| **2.1 비용 (LOC)** | ----------------------- | p.04 |  | **6.3 활동 다이어그램** | ----------------------- | p.72 |
| **2.2 개발 인력** | ----------------------- | p.05 |  | **6.4 상태(State) 다이어그램** | ----------------------- | p.76 |
| **2.3 조직 구성** | ----------------------- | p.06 |  | **7. 개발 환경** | ------------------------------- | p.80 |
| 1. **일정 및 개발 절차** | ------------------------------- | p.07 |  | **7.1 개발 도구** | ----------------------- | p.80 |
| **3.1 WBS 차트** | ----------------------- | p.07 |  | **7.2 회의** | ----------------------- | p.80 |
| **3.2 CPM 네트워크** | ----------------------- | p.08 |  | **8. 기술 관리 방법** | ------------------------------- | p.82 |
| **3.3 임계 경로 및 임계치** | ----------------------- | p.08 |  | **8.1 변경 관리** | ----------------------- | p.82 |
| **3.4 간트 차트** | ----------------------- | p.09 |  | **8.2 형상 관리** | ----------------------- | p.83 |
| **3.5 기능요구사항** | ----------------------- | p.10 |  | **8.3 위험 관리** | ----------------------- | p.85 |
| **3.5.1 구현에 차질이 생긴 이유** | ----------------------- | p.11 |  | **9. 성능 시험 방법** | ------------------------------- | p.86 |
| **3.6 COCOMO II** | ------------------------------- | p.12 |  | **9.1 단위 테스트** | ----------------------- | p.86 |
| 1. **표준 및 개발 절차** | ----------------------- | p.15 |  | **9.2 통합 테스트** | ----------------------- | p.87 |
| * 1. **개발 방법론** | ----------------------- | p.15 |  | **9.3 시스템 테스트** | ----------------------- | p.88 |
| **5. 시스템 설명** | ------------------------------- | p.16 |  | **10. 느낀 점** | ------------------------------- | p.89 |
| **5.1 기능 적용 패턴** | ----------------------- | p.16 |  | **11. 참고문헌 및 부록** |  |  |
| **5.2 주문 관리** | ----------------------- | p.17 |  |  |  |  |
| **5.3 메뉴 관리** | ----------------------- | p.29 |  |  |  |  |

|  |  |
| --- | --- |
| **1** | **개요** |

|  |  |
| --- | --- |
| **1.1** | **프로젝트 개요** |

|  |  |
| --- | --- |
| **프로그램 명** | RMS (Restaurants Management System) |
| **참여 인원** | 조진혁 (20203120) 이수찬 (20172128)  최인수 (20173236)  박상현 (20183145) |
| **목적** | 재사용성이 높은 식당 관리 시스템 |
| **기대효과** | 1. 식당에서의 전체적인 관리가 가능하다 2. 사장과 직원의 접근 기능을 분류한다 3. 추가적인 기능에 유연하게 추가가 가능하다 |
| **주요 기능** | 1. 주문 관리 2. 결제 관리 3. 메뉴 관리 4. 매출 관리 5. 직원 관리 6. 월급 관리 7. 테이블 관리 |
| **상세 설명** | RMS 프로그램은 결제 처리만 해 주는 POS기(Point of sale)의 기존 제품을 확장 및 개선하여 결제뿐만 아니라 메뉴, 잔액, 직원 관리 등의 기능을 추가하여 음식점에서 이루어지는 전체적인 역할을 담당하는 프로그램이다.  RMS 프로그램은 크게 사장과 직원(아르바이트 생)을 분류하여 접근이 가능한 정보를 구분하였고 프로젝트의 특성 상 요구사항이 자주 변경될 수 있으며 비슷한 주제의 여러 프로그램의 수요가 많으므로 개발 과정 중에 디자인패턴을 적극 활용하여 유지보수와 기능 추가/변형이 수월하게 하였다. |

|  |  |
| --- | --- |
| **1.2** | **주제 선정 이유** |

* **배달의 민족, 요기요 등 여러 플랫폼을 하나로 관리하는 동시에 홀 주문 관리 및 매장 관리**
  + 요즘 음식점들의 거의 배달의 민족 요기요 서비스를 사용하여 주문 관리를 하고 있다.
  + 실제 각각의 서비스 마다 다른 주문 환경을 가지고 있으며, 다른 태블릿 PC를 설치하고 사용하여 주문받고 있다.
  + 각기 다른 서비스 마다 다른 주문 환경을 가지고 있어, 다른 태블릿 PC를 설치하고 사용하여 주문받고 있다.
  + 각기 다른 서비스마다 다른 주문 환경을 합치면 보다 수월한 주문 관리를 할 수 있을 것 같다는 생각하였다
* **음식점에서 일하는 직원들의 근태, 월급 관리를 할 수 있다.**
  + 단순히 매장 내의 포장 주문, 홀 주문만 관리하는 시스템이 아닌, 매장에서 일하는 직원들의 근무 시간, 근무 날짜 등을 기록할 수 있는 서비스를 만들어서 관리하고자 한다.
  + 직접 눈으로 확인하다 보면 사람인지라, 누구나 실수하고 잘못된 재고 관리를 할 수 있는데, 이러한 문제점을 해결하기 위해 재고 관리에 대한 시스템도 함께 만들고자 한다.

|  |
| --- |
| **POS 홀 주문 관리 시스템** |
| Anatomy of POS Testing and why we need it? - TelecomDrive |

|  |  |
| --- | --- |
| **1.3** | **적용 패턴** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **계획** | |  | **구현** | |
| **작업 목록** | **적용 계획 패턴** |  | **작업 목록** | **적용 패턴** |
| 주문 입력 | 커맨드 패턴 |  | 직원 근퇴 관리 | 커맨드 패턴 |
| 주문한 영수증 관리 | 옵저버 패턴 |  | 메뉴 등록 | 옵저버 패턴 |
| 결제 관리 | 스트레티지 패턴 |  | 주문 입력 | 스트레티지 패턴 |
| 추가 주문 | 데코레이터 패턴 |  | 매출 조회 | 데코레이터 패턴 |
| 주방 및 주문 관리 | 미정 |  | 직원 등록 | 팩토리 패턴 |

|  |  |
| --- | --- |
| **1.4** | **정의, 약어** |

|  |  |
| --- | --- |
| **정의 & 약어** | **설명** |
| **POS 시스템** | Point of Sale의 약어로, 비즈니스 사업에 필요한 재정 및 관리 작업을 처리해준다. |
| **동백전** | 소상공인과 시민, 전통시장이 함께 소비의 선순환을 만들어 주는 부산지역화폐 |
| **누비전** | 소상공인의 매출증대와 지역소비를 통한 지역경제 활성화를 위한 창원지역화폐 |
| **CRM** | Customer Relationship Management의 약어로, 고객과의 관계를 관리하는  소프트웨어 시스템 |
| **SKU** | Stock Keeping Unit의 약어로, 재고를 관리하는 데에 있어서 필요한 데이터를  판별하는 코드 |

|  |  |
| --- | --- |
| **2** | **자원 및 인력** |

|  |  |
| --- | --- |
| **2.1** | **비용 (LOC)** |

|  |
| --- |
| **LOC 산정 계산식** |
| * **노력 (M/M)**   + 참여인원 x 개발 기간 * **개발비용**   + 노력 x 단위비용(1인당 월평균 인건비) |

|  |
| --- |
| **LOC 산정 과정** |
| 1. **노력 산정**    1. **4(M/M) = 4 x 1** 2. **개발 비용 산정**    1. 개발 비용 = 4 x (최저시급(9,160원\*5시간 (근무시간)) x (일주일 :5일) x (한달 4주기준))    2. 3,664,000원 = (4 x (45,800 x 5 x 4)) |

|  |
| --- |
| **최종 개발 비용** |
| * **최종 개발 비용 (4인 개발자 기준)**   + **총 3,664,000원 예상** |

|  |  |
| --- | --- |
| **2.2** | **개발 인력** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **인력 별 주요 개발 담당** | | | | |
| **총 개발 인원** | 4명 | | | |
| **이름** | **조진혁** | **이수찬** | **최인수** | **박상현** |
| **사진** |  |  |  |  |
| **프로젝트 담당** | 팀장 | 보고서 검토  개발 | 보고서 작성  개발 | 회의록 작성  개발 |
| **담당 기능** | 결제 관리  주문 관리 | 매출 관리  잔액 관리 | 메뉴 관리  주문 관리 | 주문 관리  잔액 관리 |
| **테스트 담당** | Windows 10  IntelliJ IDEA | Windows 11  NetBeans | Mac OS  IntelliJ IDEA | Windows 10  IntelliJ IDEA |
| **디자인 패턴 설계** | 공동 작업 | | | |

|  |  |
| --- | --- |
| **2.3** | **조직 구성** |

Diagram

Description automatically generated

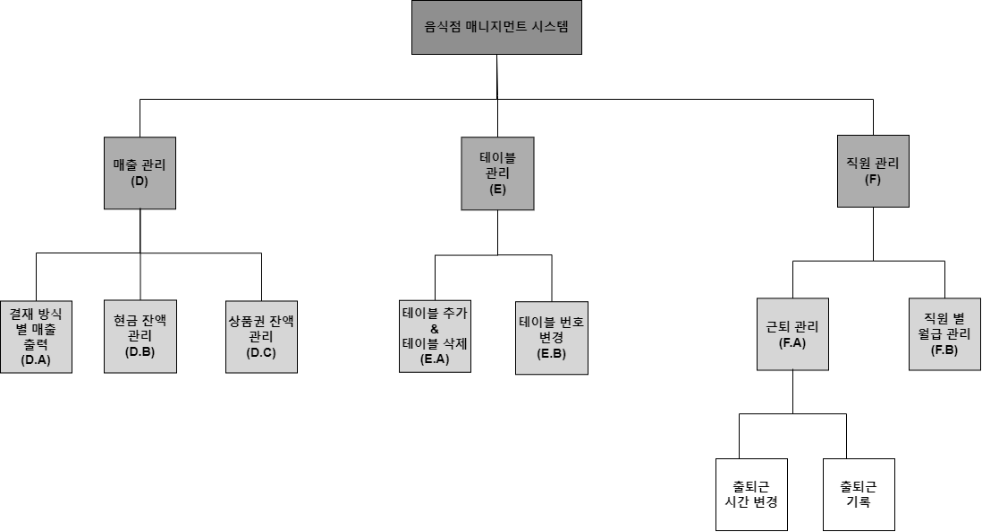
|  |  |
| --- | --- |
| **프로젝트 관리자** | 프로젝트의 전반적인 부분을 관리. 일정, 요구사항, 개발 비용 등을 관련 관리자들과 토론/의견교환 등을 통해 조율하여 프로젝트 전체를 제어한다. |
| **일정 관리자** | 프로젝트의 전체적인 일정을 관리한다. 프로젝트의 지연/잉여 시간 관리, 병렬 작업 수행 등에서의 책임을 진다. |
| **요구사항 관리자** | 프로젝트의 사용자 요구 사항 관리, 사용자와의 협의 등을 담당한다. 프로젝트 내 요구사항 변경이 있을 시 일정 관리자와 협의하여 새로운 일정을 이끌어내거나 기존 일정에 변경을 가해야 한다. |
| **개발 비용 관리자** | 프로젝트 진행 중의 개발 비용/필요한 리소스 관리를 담당한다. |

|  |  |
| --- | --- |
| **3** | **일정 및 개발 절차** |

|  |  |
| --- | --- |
| **3.1** | **WBS 차트** |

텍스트, 표지판이(가) 표시된 사진

자동 생성된 설명



|  |  |
| --- | --- |
| **3.2** | **CPM 네트워크** |

텍스트이(가) 표시된 사진

자동 생성된 설명

|  |  |
| --- | --- |
| **3.3** | **임계 경로 및 임계치** |

|  |  |
| --- | --- |
| **임계 경로** | A.A → A.B → A.D →C.A → C.B → D.A → D.B → E.A → E.B → F.A → F.B |
| **임계치** | 총 24일 |

|  |  |
| --- | --- |
| **3.4** | **간트 차트** |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ID** | **작업명** | **시작일** | **기간(주)** | **5월** | | | **6월** | |
| **2** | **3** | **4** | **1** | **2** |
| 1 | 주문 관리 | 22.05.09 | 1 |  |  |  |  |  |
| 2 | 메뉴 관리 | 22.05.17 | 2 |  |  |  |  |  |
| 3 | 매출 관리 | 22.05.25 | 2 |  |  |  |  |  |
| 4 | 직원 관리 | 22.05.29 | 1 |  |  |  |  |  |
| 5 | 테이블 관리 | 22.06.10 | 1 |  |  |  |  |  |

|  |  |
| --- | --- |
| **3.5** | **기능요구사항** |

|  |  |  |
| --- | --- | --- |
| **Id** | **기능** | **설명** |
| **SFR-0100** | **주문 관리** | **-** |
| SFR-0101 | 주문 등록 | 사장/직원은 주문을 입력/취소한다 |
| SFR-0102 | 주문 홀 등록 | 사장/직원은 홀에 배정되는 주문을 입력/취소한다. |
| SFR-0103 | 주문 배달 등록 | 사장/직원은 배달에 배정되는 주문을 입력/취소한다. |
| SFR-0104 | 주문 포장 등록 | 사장/직원은 포장에 배정되는 주문을 입력/취소한다. |
| SFR-0105 | 주문서 출력 | 사장/직원은 주문서를 출력한다. |
| SFR-0106 | 주문 조회 | 사장/직원은 주문 리스트를 출력한다 |
| **SFR-0300** | **메뉴 관리** | **-** |
| SFR-0301 | 메뉴 등록 | 사장은 메뉴를 추가/수정/삭제한다. |
| **SFR-0400** | **매출 관리** | **-** |
| SFR-0401 | 기간 별 매출 조회 | 사장은 연간/월간/주간 매출을 조회한다. |
| SFR-0402 | 방식 별 매출 조회 | 사장은 카드/현금/상품권 매출을 조회한다. |
| SFR-0403 | 잔액 조회 | 사장/직원은 잔액을 조회한다 |
| SFR-0404 | 현금 관리 | 사장/직원은 현금을 추가/제거한다. |
| SFR-0405 | 상품권 관리 | 사장/직원은 상품권을 추가/제거한다. |
| **SFR-0500** | **직원 관리** | **-** |
| SFR-0501 | 직원 조회 | 사장은 직원 리스트를 조회한다. |
| SFR-0502 | 직원 등록 | 사장은 직원을 추가/변경/제거한다. |
| SFR-0504 | 근퇴 조회 | 직원은 자신의 근퇴를 조회한다 |
| SFR-0505 | 직원 근퇴 변경 | 직원은 출근/퇴근을 변경한다. |
| SFR-0506 | 총 월급 조회 | 사장은 모든 직원의 월급을 조회한다. |
| **SFR-0600** | **테이블 관리** | **-** |
| SFR-0601 | 테이블 조회 | 사장은 테이블을 조회한다 |
| SFR-0602 | 테이블 별 주문 관리 | 직원은 테이블에 주문을 적용한다. |

|  |  |
| --- | --- |
| **3.5.1** | **구현에 차질이 생긴 이유** |

|  |  |  |
| --- | --- | --- |
| **ID** | **기능 이름** | **이유** |
| **SFR-0200** | 결제 관리 | 패턴 적용에 집중하다 보니 우선순위를 정하게 되었고, 우선순위가 높은 기능부터 구현하다가 일정 관리에 지장이 생겨 구현하지 못하였다. |
| **SFR-0503** | 출퇴근 시간 관리 | 출퇴근 시간을 적용하면 등록 시간에 맞게 직원이 출근 또는 퇴근 기능을 실행해야 하며, 이 과정에서 많은 예측 비용이 들어갈 것으로 판단되어 팀원과의 회의 끝에 구현에서 제외하기로 결정하였다. |
| 프로그램 요구사항이 자주 변경되는 것을 전제로 한 개발 계획이었기 때문에, 변경에 강한 프로젝트 아키텍처를 구축하는데 많은 시간을 할애했다. 따라서 개발 목적에 따라 개발 일정을 조금 더 유연하게 할당했다면 프로젝트의 완성도가 더 높아졌을 것으로 예상된다. | | |

|  |  |
| --- | --- |
| **3.6** | **COCOMO II** |

* **프로젝트에 사용될 금액적 / 시간적 비용과 그 진행도를 나타낸다**
  + COCOMO || 방법에 의한 개발 비용 예측 (간이 기능 점수법 이용 (KOSA 사업대가 산정 Excel File)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **기능 점수 산정** | | | | | |
| **기능명** | | | | **데이터 및 트랜잭션 기능** | **FP 산출** |
| 1. **어플리케이션 명** | **②세부 업무명** | **③단위 프로세스명** | **단위프로세스 설명** | **④FP유형** | **⑤가중치** |
| 음식점매니지먼트시스템 | 주문관리 | 주문 등록 | 사장/직원은 주문을 입력/취소한다 | EI | 4.0 |
| 음식점매니지먼트시스템 | 주문관리 | 주문 홀 등록 | 사장/직원은 홀에 배정되는 주문을 입력/취소한다. | EQ | 3.9 |
| 음식점매니지먼트시스템 | 주문관리 | 주문 배달 등록 | 사장/직원은 배달에 배정되는 주문을 입력/취소한다. | EI | 4.0 |
| 음식점매니지먼트시스템 | 주문관리 | 주문 포장 등록 | 사장/직원은 포장에 배정되는 주문을 입력/취소한다. | EO | 5.2 |
| 음식점매니지먼트시스템 | 주문관리 | 주문서 출력 | 사장/직원은 주문서를 출력한다. | EO | 5.2 |
| 음식점매니지먼트시스템 | 주문관리 | 주문 조회 | 사장/직원은 주문 리스트를 출력한다 | EI | 4.0 |
| 음식점매니지먼트시스템 | 메뉴 관리 | 메뉴 등록 | 사장은 메뉴를 추가/수정/삭제한다. | EI | 4.0 |
| 음식점매니지먼트시스템 | 매출 관리 | 기간 별 매출 조회 | 사장은 연간/월간/주간 매출을 조회한다. | EO | 5.2 |
| 음식점매니지먼트시스템 | 매출 관리 | 방식 별 매출 조회 | 사장은 카드/현금/상품권 매출을 조회한다. | EO | 5.2 |
| 음식점매니지먼트시스템 | 매출 관리 | 잔액 조회 | 사장/직원은 잔액을 조회한다 | EO | 5.2 |
| 음식점매니지먼트시스템 | 매출 관리 | 현금 관리 | 사장/직원은 현금을 추가/제거한다. | EI | 4.0 |
| 음식점매니지먼트시스템 | 매출 관리 | 상품권 관리 | 사장/직원은 상품권을 추가/제거한다. | EI | 4.0 |
| 음식점매니지먼트시스템 | 직원 관리 | 직원 조회 | 사장은 직원 리스트를 조회한다. | EO | 5.2 |
| 음식점매니지먼트시스템 | 직원 관리 | 직원 등록 | 사장은 직원을 추가/변경/제거한다. | EQ | 3.9 |
| 음식점매니지먼트시스템 | 직원 관리 | 출퇴근 시간 관리 | 사장은 직원의 출퇴근 시간을 설정/변경한다. | EI | 4.0 |
| 음식점매니지먼트시스템 | 직원 관리 | 근퇴 조회 | 직원은 자신의 근퇴를 조회한다 | EO | 5.2 |
| 음식점매니지먼트시스템 | 직원 관리 | 직원 근퇴 변경 | 직원은 출근/퇴근을 변경한다. | EI | 4.0 |
| 음식점매니지먼트시스템 | 직원 관리 | 총 월급 조회 | 사장은 모든 직원의 월급을 조회한다. | EQ | 3.9 |
| 음식점매니지먼트시스템 | 테이블 관리 | 테이블 조회 | 사장은 테이블을 조회한다 | EQ | 3.9 |
| 음식점매니지먼트시스템 | 테이블 관리 | 테이블 별 주문 관리 | 직원은 테이블에 주문을 적용한다. | EI | 4.0 |

|  |  |
| --- | --- |
| **트랜잭션 기능 점수 계산** | |
| **ILF Internal Logical File (내부 논리 파일)** | 사용자가 등록/수정/삭제/조회를 하기 위한 대상으로, 해당 프로젝트에는 사용되지 않는다. |
| **EIF External Interface File (외부 연계 파일)** | 측정 대상 애플리케이션에서 참조만 하고 다른 애플리케이션에서는 유지되는 파일로, 해당 프로젝트에는 사용되지 않는다. |
| **EI External Input (외부 입력)** | 데이터베이스에 데이터를 등록/수정/삭제하는 것으로 ‘주문 등록’, ‘카드 결제’ 등으로 사용된다 |
| **EO External Output (외부 출력)** | 계산하는 로직을 거쳐 사용자에게 보여주는 기능으로 ‘매출 조회’, ‘영수증 출력’ 등으로 사용된다 |
| **EQ External inQuiry (외부 조회)** | 단순히 데이터나 제어 정보를 사용자에게 제공하는 것으로 ‘직원 등록’, ‘테이블 조회’ 등으로 사용된다 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **보정 계수 산정** | | | | | | |
| **구분** | **ILF** | **EIF** | **EI** | **EO** | **EQ** | **계** |
| 기능수 | 0 | 0 | 8 | 7 | 4 | **19** |
| 기능점수 | 0.0 | 0.0 | 32.0 | 36.4 | 15.6 | **84.0** |
| 비중 | 0% | 0% | 38% | 43% | 19% | **100%** |

|  |  |
| --- | --- |
| **4** | **표준 및 개발 절차** |

|  |  |
| --- | --- |
| **4.1** | **개발 방법론** |

|  |
| --- |
| **소프트웨어 개발 생명주기** |
|  |
| **진화적 프로세스 모델** |
|  |

|  |  |
| --- | --- |
| **5** | **시스템 설명** |

|  |  |
| --- | --- |
| **5.1** | **기능 적용 패턴** |

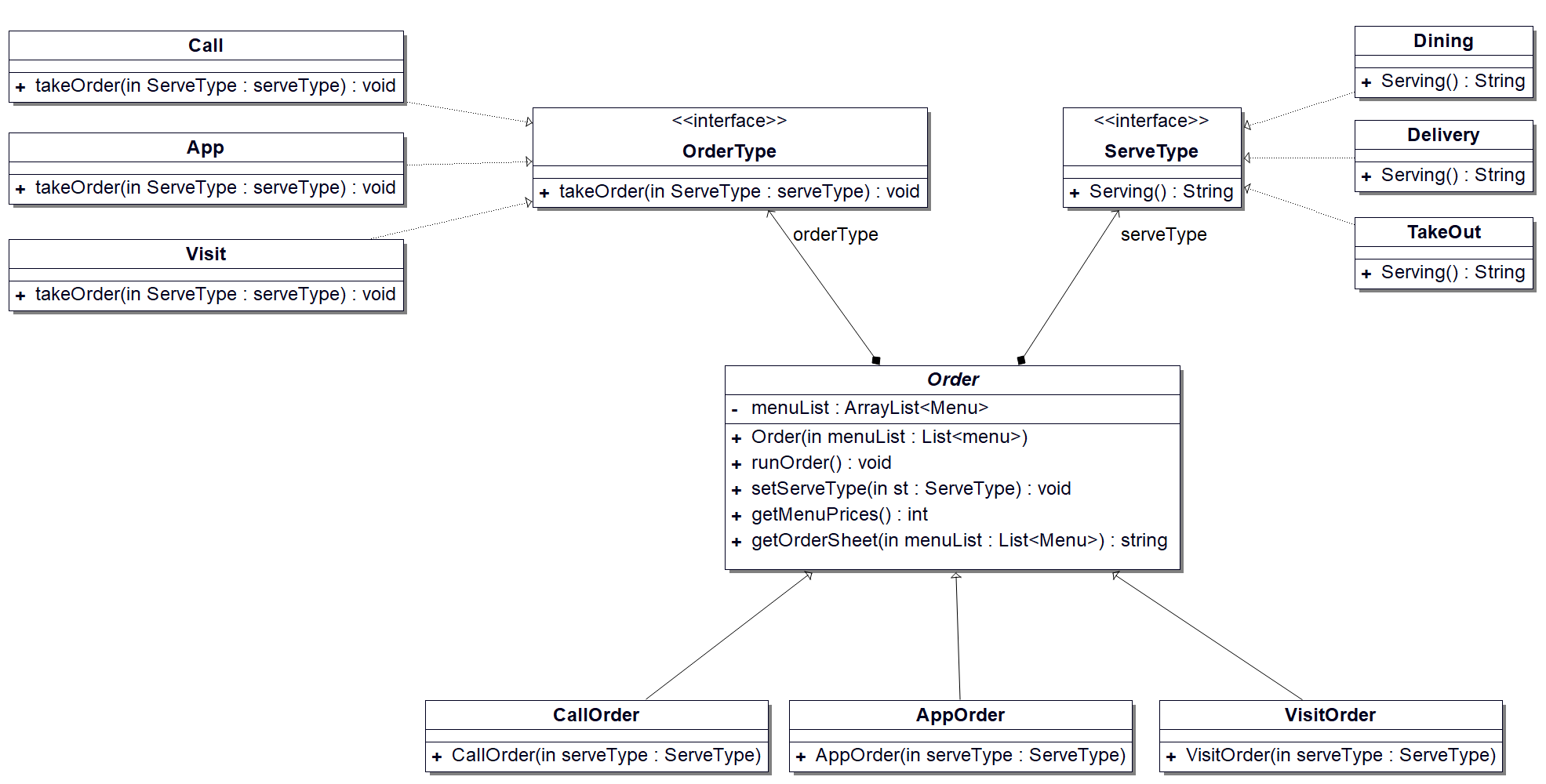
|  |  |  |
| --- | --- | --- |
| **ID** | **주요 기능** | **적용 패턴** |
| **SFR-0100** | 주문 관리 | Strategy pattern |
| **SFR-0300** | 메뉴 관리 | Observer pattern |
| **SFR-0400** | 매출 관리 | Decorator pattern |
| **SFR-0500** | 직원 관리 | Factory pattern, Command pattern |

|  |  |
| --- | --- |
| **5.2** | **주문 관리** |

|  |  |
| --- | --- |
| **사용 디자인 패턴** | 스트레티지 패턴 (Strategy pattern) |
| **패턴 소개** | 달라지지 않는 부분(메뉴)과 달라지는 부분(주문 방식/서빙 방식)을 분리한다 |
| **적용 계획** | 손님이 주문을 진행할 때 달라지지 않는 부분인 메뉴와 달라지는 부분인 주문 방식/서빙 방식으로 분리된다. 이에 따라 주문 관리 기능을 스트레티지 패턴으로 적용하였다. |
| **패턴 적용 방식** | |
| 1. **주문을 받을 때의 방식이 달라진다**(OrderType)    1. 전화(Call), 어플(App), 방문(Visit) 2. **조리 완료 후 서빙 방식이 달라진다**(ServeType)    1. 홀(Dining), 배달(Deilvery), 포장(TakeOut) 3. **주문 방식과 서빙 방식에 맞는 주문 클래스를 생성한다(**Order) 4. **배달 받을 때의 방식에 따라 서빙의 방식을 추가/변경한다**    1. 전화 주문(CallOrder), 어플 주문(AppOrder), 방문 주문(VisitOrder) | |
| **프로젝트 적용 예시** | * **사용자가 어플로 포장 주문을 시켰을 때**   + Order클래스를 상속받는 AppOrder 클래스에서 ServeType이 포장(TakeOut)이 된다 |

|  |
| --- |
| **UseCase** |
|  |
| **상태 Diagram** |
|  |

|  |  |
| --- | --- |
| **5.2.1** | **패턴 UML** |



|  |  |
| --- | --- |
| **5.2.2** | **주요 클래스** |

|  |  |
| --- | --- |
| **주문 방식 클래스** | |
|  | |
| **+ void takeOrder(ServeType : serveType)** | 주문을 받을 때 |

|  |  |
| --- | --- |
| **서빙 방식 클래스** | |
|  | |
| **+ String Serving()** | 서빙 방식에 따른 행동을 담고 있는 메소드 |

|  |  |
| --- | --- |
| **주문 클래스** | |
|  | |
| **- menuList : ArrayList<menu>** | 주문의 메뉴 리스트를 담는 필드 |
| **+ Order(in menuList : List<menu>** | 주문 클래스의 생성자. 주문 메뉴/서빙 방식을 지정 |
| **+ setServeTyoe(st : ServeType) : void** | 서빙 방식을 변경하는 메소드 |
| **CallOrder 클래스** | 전화로 온 주문 정보를 처리하는 클래스 |
| **AppOrder 클래스** | 앱을 통해 받은 주문 정보를 처리하는 클래스 |
| **VisitOrder 클래스** | 방문으로 받은 주문 정보를 처리하는 클래스 |

|  |  |
| --- | --- |
| **5.2.3** | **기능 설명** |

|  |  |
| --- | --- |
| **소주제** | 주문 등록 |
| **ID** | SFR-0101 |
| **클래스** | order/input/Order.java |
| **Actor** | 사장, 직원 |
| **목적** | 주문이 들어오면 사용자가 각 형식에 맞는 주문을 입력하고 취소한다 |

|  |  |
| --- | --- |
| **소주제** | 주문 홀 등록 |
| **ID** | SFR-0102 |
| **클래스** | order/input/Dining.java |
| **Actor** | 사장, 직원 |
| **목적** | 사용자는 홀에 제공(서빙)되는 주문을 입력하고 취소한다. |

|  |  |
| --- | --- |
| **소주제** | 주문 배달 등록 |
| **ID** | SFR-0103 |
| **클래스** | order/input/Delivery.java |
| **Actor** | 사장, 직원 |
| **목적** | 사용자는 배달에 제공(서빙)되는 주문을 입력하고 취소한다. |

|  |  |
| --- | --- |
| **소주제** | 주문 포장 등록 |
| **ID** | SFR-0104 |
| **클래스** | order/input/TakeOut.java |
| **Actor** | 사장, 직원 |
| **목적** | 사용자는 포장에 제공(서빙)되는 주문을 입력하고 취소한다. |

|  |  |
| --- | --- |
| **소주제** | 주문서 출력 |
| **ID** | SFR-0105 |
| **클래스** | order.input.Order#getOrderSheet |
| **Actor** | 사장, 직원 |
| **목적** | 사용자는 식당에 등록된 주문의 전체를 출력한다. |

|  |  |
| --- | --- |
| **소주제** | 주문 조회 |
| **ID** | SFR-0106 |
| **클래스** | order.input.Order#Order |
| **Actor** | 사장, 직원 |
| **목적** | 사용자는 현재 주문을 출력한다. |

|  |  |
| --- | --- |
| **5.2.4** | **주요 코드** |

|  |
| --- |
| **order/input/OrderManager.java** |
| /\* 중략 \*/  public void createOrder() {  // Get order information from console  // loop while user types -1 as menu name  List<Menu> menus = new ArrayList<>();  while (true) {  String menuName = Console.getInput("메뉴를 입력하세요(-1을 입력해 반복 입력을 종료): ");  if (menuName.equals("-1")) {  break;  }  Optional<Menu> menu = menuRepository.getMenuByName(menuName);  if (!menu.isPresent()) {  System.out.println("존재하지 않는 메뉴입니다.");  continue;  }  menus.add(menu.get());  }  // check menus name again  System.out.println("주문하신 메뉴는 다음과 같습니다.");  menus.forEach(menu -> {  System.out.println(menu.getName() + ": " + menu.getPrice() + "원");  });  // set pickup type  System.out.println("배달 유형을 선택하세요");  for (ServingType type : ServingType.values()){  System.out.println(type.getName());  }  String servingType = Console.getInput("선택: ");  ServingType servingTypeEnum = ServingType.getServeType(servingType);  if (servingTypeEnum == null) {  System.out.println("잘못된 입력입니다.");  return;  }  // set order type  System.out.println("주문 유형을 선택하세요");  for (OrderType type : OrderType.values()){  System.out.println(type.getName());  }  String orderType = Console.getInput("선택: ");  OrderType orderTypeEnum = OrderType.getOrderType(orderType);  if (orderTypeEnum == null) {  System.out.println("잘못된 입력입니다.");  return;  }  // create Order  Order order = orderTypeEnum.getOrderSupplier().apply(menus, servingTypeEnum.getServeTypeSupplier().get());  // add order to repository  int ordered = orderRepository.addOrder(order);  order.runOrder();  System.out.println("주문이 완료되었습니다. 주문 번호는 " + ordered + "입니다.");  }  public void doneOrder() {  int orderId = Integer.parseInt(Console.getInput("주문 번호를 입력하세요: "));  Order order = orderRepository.getOrder(orderId);  if (order == null) {  System.out.println("주문이 존재하지 않습니다.");  return;  }  order.finishOrder();  System.out.println("주문이 완료되었습니다.");  }  /\* 중략 \*/ |

|  |
| --- |
| **order/input/Order.java** |
| public abstract class Order {  protected OrderType orderType;  protected ServeType serveType;  List<Menu> menuList = new ArrayList<Menu>();  public void runOrder() {  orderType.takeOrder(serveType);  }  public void finishOrder() {  System.out.println(serveType.serving());  }  public void setServeType(ServeType st) {  serveType = st;  }  public int getMenuPrices() {  int priceSum = menuList.stream().mapToInt(Menu::getPrice).sum();  return priceSum;  }  } |

|  |
| --- |
| **order/input/Dining.java** |
| public class Dining implements ServeType{  public String serving(){  return "(서빙을 시작합니다.)";  }  } |

|  |  |
| --- | --- |
| **5.2.5** | **결과 화면** |

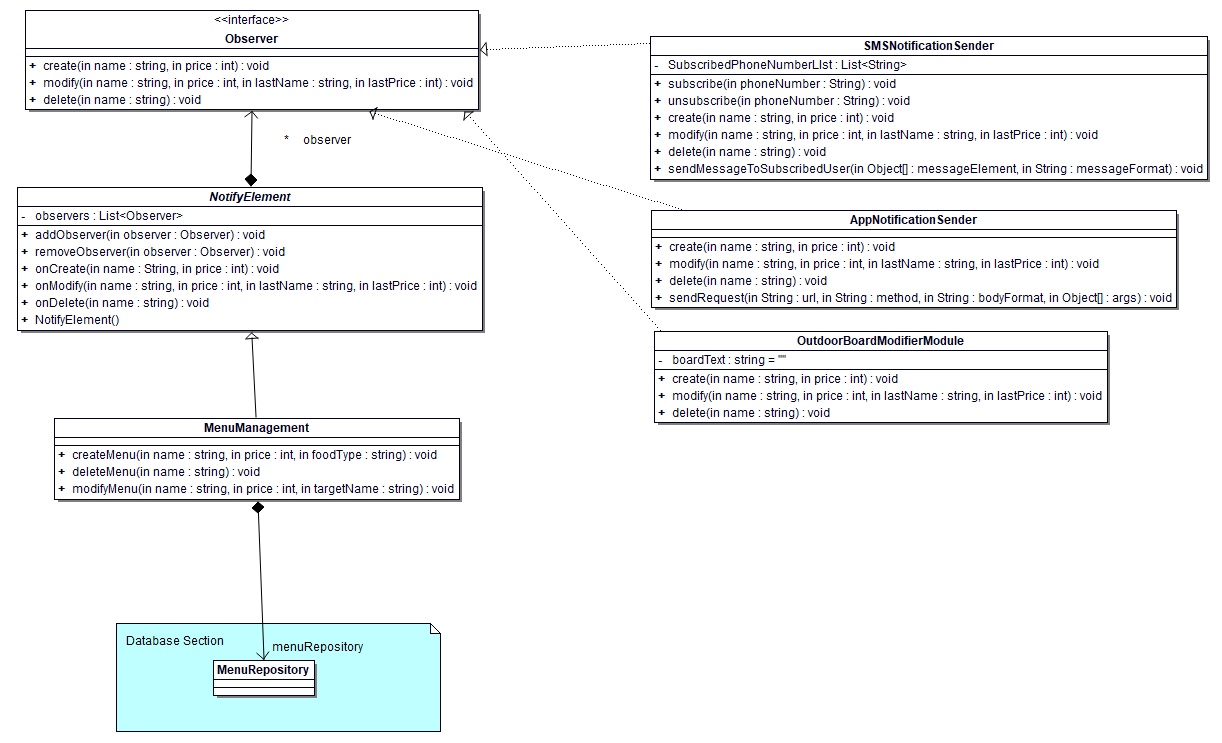
|  |
| --- |
| /\* 주문 생성 \*/  주문을 받습니다.  메뉴를 입력하세요(-1을 입력해 반복 입력을 종료): 생선까스  메뉴를 입력하세요(-1을 입력해 반복 입력을 종료): 돈까스  메뉴를 입력하세요(-1을 입력해 반복 입력을 종료): -1  주문하신 메뉴는 다음과 같습니다.  생선까스: 10000원  돈까스: 10000원  배달 유형을 선택하세요  배달  픽업  식사  선택: 배달  주문 유형을 선택하세요  어플리케이션  전화  방문  선택: 어플리케이션  [어플 주문]  주문이 완료되었습니다. 주문 번호는 0입니다.  주문을 받습니다.  메뉴를 입력하세요(-1을 입력해 반복 입력을 종료): 돈까스  메뉴를 입력하세요(-1을 입력해 반복 입력을 종료): 돈까스  메뉴를 입력하세요(-1을 입력해 반복 입력을 종료): 냉면  메뉴를 입력하세요(-1을 입력해 반복 입력을 종료): -1  주문하신 메뉴는 다음과 같습니다.  돈까스: 10000원  돈까스: 10000원  냉면: 10000원  배달 유형을 선택하세요  배달  픽업  식사  선택: 픽업  주문 유형을 선택하세요  어플리케이션  전화  방문  선택: 방문  [방문 주문]  주문이 완료되었습니다. 주문 번호는 **1**입니다.  /\* 주문 완료 \*/  주문을 완료합니다.  주문 번호를 입력하세요: **0**  (조리 완료 알람을 보냅니다.)  주문이 완료되었습니다.  주문을 완료합니다.  주문 번호를 입력하세요: **1**  (포장합니다.)  주문이 완료되었습니다. |

|  |  |
| --- | --- |
| **5.3** | **메뉴 관리** |

|  |  |
| --- | --- |
| **사용 디자인 패턴** | 옵저버 패턴 (Observer pattern) |
| **패턴 소개** | 하나의 객체의 상태가 변경되면 그 객체에 의존하는 다른 객체에게 연락이 가고, 자동으로 갱신되는 1:n 의존성 방식 |
| **적용 계획** | 사장이 메뉴를 추가/수정/삭제할 때마다 단골 손님으로 추가된 연락처, 즐겨찾기로 추가된 배달 어플, 매장 내 메뉴판에 변경된 내용이 적용되어야 한다. 이에 따라 메뉴 관리 기능을 옵저버 패턴으로 적용하였다. |
| **패턴 적용 방식** | |
| 1. **메뉴 관리에 따라 해당되는 메소드가 실행된다**(MenuManagement)    1. createMenu : 메뉴 추가    2. deleteMenu : 메뉴 제거    3. modifyMenu : 메뉴 수정 2. **①에서 실행된 메소드를 통해 다른 클래스에게 연락하는 옵저버 클래스**(Observer) 3. **변경된 내용을 적용한다.**    1. **SMSNotificationSender** : 단골 손님으로 추가된 연락처에 변경 사항을 문자메시지로 전송한다.    2. **AppNotificationSender** : 배달 어플에서 해당 식당을 즐겨찾기로 추가된 사용자에게 변경 사항을 어플 알림으로 전송한다.    3. **MenuBoardDisplay** : 매장 내에 설치되어 있는 전자 메뉴판에 변경 사항을 적용하여 손님에게 메뉴 정보를 표기한다. | |
| **프로젝트 적용 예시** | * **사장이 ‘간장 돈까스’ 메뉴를 추가한다.**  1. MenuManagement의 createMenu() 메소드가 실행된다. 2. Observer 클래스의 create() 메소드가 실행된다. 3. Observer 클래스를 구현하는 각 클래스에 연락이 간다. 4. 각 클래스의 create() 메소드를 통해 변경 사항을 적용한다 |

|  |
| --- |
| **Use Case 다이어그램** |
|  |

|  |  |
| --- | --- |
| **5.3.1** | **패턴 UML** |

****

|  |  |
| --- | --- |
| **5.3.2** | **주요 클래스** |

|  |  |
| --- | --- |
| **메뉴 관리 클래스** | |
|  | |
| **+ createMenu**  **(name : string, price : int, foodType : string) : void** | 메뉴를 추가할 때 수행되는 메소드 |
| **+ deleteMenumodifyMenu**  **(name : string, price : int, targetName : string) : void** | 메뉴를 제거할 때 수행되는 메소드 |
| **+ modifyMenu**  **(name : string, price : int, targetName : string) : void** | 메뉴를 수정할 때 수행되는 메소드 |

|  |  |
| --- | --- |
| **클래스** | |
|  | |
| **- observers : List<Observer>** | 서비스에 등록된 Observer 객체 |
| **+ addObserver(observer : Observer) : void** | Subject 클래스에 변경 사항을 listen할 Observer 객체 추가 |
| **+ removeObserver(observer : Observer) : void** | Subject 클래스에 변경 사항을 listen할 Observer 객체 제거 |
| **+ onCreate(name : String, prince : int) : void** | 메뉴가 추가되었을 때 Observer에게 신호를 보낸다 |
| **+ onModify**  **(name : String, prince : int, lastName : string, lastPrice : int) : void** | 메뉴가 변경되었을 때 Observer에게 신호를 보낸다 |
| **+ onDelete(name : string) : void** | 메뉴가 제거되었을 때 Observer에게 신호를 보낸다 |

|  |
| --- |
| **Observer 클래스** |
|  |
| MenuManagement에서 실행되는 메뉴 추가/제거/수정 메소드를 통해 Observer 객체에서 각 메소드에 따라 호출된다 |

|  |  |  |
| --- | --- | --- |
| **SMSNotificationSenderㆍAppNotificationSenderㆍMenuBoardDisplay 클래스** | | |
|  | | |
| **+ Create, modify, delete()** | 메뉴가 추가/수정/제거될 때 각 클래스에 맞는 행동을 수행하는 메소드 |

|  |  |
| --- | --- |
| **5.3.3** | **기능 설명** |

|  |  |
| --- | --- |
| **소주제** | 메뉴 등록 |
| **ID** | SFR-0301 |
| **클래스** | menu/MenuManagement.java |
| **Actor** | 사장 |
| **목적** | 사장이 메뉴를 추가/수정/삭제하여 문자, 배달 어플, 전광판에 변경 사항을 적용한다 |

|  |  |
| --- | --- |
| **5.3.4** | **주요 코드** |

|  |
| --- |
| **Menu/MenuManagement.java#createMenu** |
| public void createMenu() {  try{  // Create menu  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  System.out.print("Name: ");  String name = br.readLine();  System.out.print("Price: ");  int price = Integer.parseInt(br.readLine());  this.menuRepository.createMenu(Menu.builder()  .name(name)  .price(price).build());  this.onCreate(name, price);  }  catch (IOException e) {  System.err.println("입력 중 문제가 발생했습니다.");  }  catch (NumberFormatException e) {  System.err.println("잘못된 값을 입력했습니다.");  }  } |
| **menu/interface/NotifyElement.java** |
| public void addObserver(Observer observer) {  observers.add(observer);  }  /\*\*  \* remove observer from the list of observers  \* @param observer observer to be removed  \*/  public void removeObserver(Observer observer) {  observers.remove(observer);  }  /\*\*  \* This method is used to notify all observers that the menu has been created.  \* @param name the name of the new menu.  \* @param price the price of the new menu.  \*/  protected void onCreate(String name, int price){  observers.forEach(observer -> observer.create(name, price));  }  /\*\*  \* This method is used to notify all observers that the menu has been updated.  \* @param lastName the name of the menu before update.  \* @param lastPrice the price of the menu before update.  \* @param name the name of the menu after update.  \* @param price the price of the menu after update.  \*/  protected void onModify(String lastName, int lastPrice, String name, int price){  observers.forEach(observer -> observer.modify(lastName, lastPrice, name, price));  }  /\*\*  \* This method is used to notify all observers that the menu has been deleted.  \* @param name the name of the menu to delete.  \*/  protected void onDelete(String name){  observers.forEach(observer -> observer.delete(name));  } |
| **menu/observers/SMSNotificationSendert.java** |
| List<String> subscribedPhoneNumberList;  final String CREATE\_MENU\_MESSAGE\_FORMAT = "새로운 메뉴 %s이(가) 추가되었습니다. %d 원으로 신메뉴를 만나보세요.\n";  @Override  public void create(String name, int price) {  System.out.println("Observer Detected create signal -------");  sendMessageToSubscribedUser(new Object[]{name, price}, CREATE\_MENU\_MESSAGE\_FORMAT);  System.out.println("Done sending SMS to subscribers.");  } |

|  |  |
| --- | --- |
| **5.3.5** | **결과 화면** |

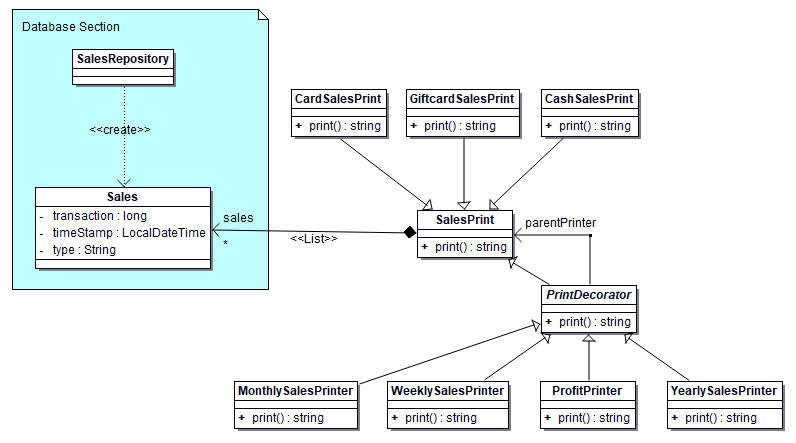
|  |
| --- |
| [1]: 메뉴 생성  [2]: 메뉴 수정  [3]: 메뉴 삭제  [4]: 종료  원하는 동작을 선택하세요: 1  Name: **간장 돈까스**  **Price: 8000**  **Observer Detected create signal -------**  Send SMS to +82 10-1234-1234 : 새로운 메뉴 간장 돈까스이(가) 추가되었습니다. 8000 원으로 신메뉴를 만나보세요.  Send SMS to +82 10-1234-5678 : 새로운 메뉴 간장 돈까스이(가) 추가되었습니다. 8000 원으로 신메뉴를 만나보세요.  Done sending SMS to subscribers.  **AppNotificationSender got create signal -----------**  POST https://your.delivery.com/api/v1/app/notifications/menu HTTP/1.1  Host: https://your.delivery.com/api/v1/app/notifications/menu  Content-Type: application/json  Content-Length: 37  Connection: close  {"menu\_name":"간장 돈까스","price":"8000"}  **AppNotificationSender sent create signal -----------**  OutdoorBoardModifierModule accepted create signal ------------  간장 돈까스 8000  create signal handle done ------------ |

|  |  |
| --- | --- |
| **5.4** | **매출 관리** |

|  |  |
| --- | --- |
| **사용 디자인 패턴** | 데코레이터 패턴 (Decorator pattern) |
| **패턴 소개** | 구성요소를 동적으로 감싸면서 어떤 행동을 위임하는 것 외에 원하는 추가적인 작업을 할 수 있는 확장성을 보인다 |
| **적용 계획** | 객체에 추가적인 요건인 매출 조회 방식을 주간/월간/연간으로 감싸여 출력하면서 카드/상품권/현금으로 추가적인 요건을 동적으로 첨가한다 |
| **패턴 적용 방식** | |
| 1. **방식 별로 매출 조회를 전체적으로 담당한다 (SalesPrint)**    1. **CardSalesPrint** : 카드 별 매출    2. **GiftcardSalesPrint** : 상품권 별 매출    3. **CashSalesPrint** : 현금 별 매출 2. **추가 구성요소를 모아, 데코레이터로 모은다 (PrintDecorator)**    1. **MonthlySalesPrinter** : 월간 매출 출력    2. **WeeklySalesPrinter** : 주간 매출 출력    3. **YearlySalesPrinter** : 연간 매출 출력    4. **ProfitPrinter** : 순이익 출력 3. **총 매출 데이터를 Database에 접근하여 정보를 받는다 (Sales)** | |
| **프로젝트 적용 예시** | * **사용자가 카드 방식의 계산을 연간 매출로 출력할 때**  1. Sales 객체에 매출이 저장된다 2. 카드 클래스인 CardSalesPrint 객체를 생성한다 3. 연간 출력인 YearlySalesPrinter클래스로 감싼 SalesPrint 클래스가 생성된다 4. Print() 메소드를 통해 매출을 출력한다 |

|  |
| --- |
| **UseCase** |
|  |

|  |  |
| --- | --- |
| **5.4.1** | **패턴 UML** |



|  |  |
| --- | --- |
| **5.4.2** | **주요 클래스** |

|  |
| --- |
| **SalesPrint** |
|  |
| 매출 출력을 전체적으로 담당하는 클래스 |

|  |
| --- |
| **PrintDecorator** |
|  |
| 주간, 월간, 월간 매출의 구성요소를 모으는 Decorator 역할을 수행한다.  사용자가 추가적인 매출 방식을 요구할 시, 클래스 추가만으로도 가능하게 확장할 수 있다, |

|  |  |
| --- | --- |
| **5.4.3** | **기능 설명** |

|  |  |
| --- | --- |
| **소주제** | 기간 별 매출 조회 |
| **ID** | SFR-0401 |
| **클래스** | * + - 월간 매출 출력 : sales/print/printer/decorator/MonthlySalesPrinter.java     - 연간 매출 출력 : sales/print/printer/decorator/YearlySalesPrinter.java |
| **Actor** | 사장 |
| **목적** | 사장이 월간 또는 연간 매출을 출력하여 확인한다. |

|  |  |
| --- | --- |
| **소주제** | 방식 별 매출 조회 |
| **ID** | SFR-0402 |
| **클래스** | * + - 카드 매출 출력 : sales/print/printer/CardSalesPrint.java     - 상품권 매출 출력 : sales/print/printer/GiftcardSalesPrint.java     - 현금 매출 출력 : sales/print/printer/CashSalesPrint.java |
| **Actor** | 사장 |
| **목적** | 사장이 카드, 상품권, 현금별로 계산된 데이터를 통해 매출을 출력하여 확인한다. |

|  |  |
| --- | --- |
| **소주제** | 현금 관리 |
| **ID** | SFR-0404 |
| **클래스** | * + - 잔액 조회 : sales/SalesManagement.java |
| **Actor** | 사용자는 카운터 계산대에 있는 현금의 잔액을 출력하여 남은 현금을 확인한다. |
| **목적** |  |

|  |  |
| --- | --- |
| **5.4.4** | **주요 코드** |

|  |
| --- |
| **SalesPrint.java** |
| public class SalesPrint {  List<Sales> sales;  /\*\*  \* This constructor retrieves whole sales list from SalesRepository.  \* Create this class to print whole sales list.  \*/  public SalesPrint() {  sales = SalesRepository.getInstance().getSales().stream()  .sorted(Comparator.comparing(Sales::getTimestamp).reversed())  .collect(Collectors.toList());  }  /\*\*  \* Returns the sales list on specified format. Override this method on decorator class to apply format.  \* @return sales list represented by string on specified format  \*/  public String print() {  return "--매출 출력 종료--";  }  } |
| **PrintDecorator.java** |
| public abstract class PrintDecorator extends SalesPrint {  /\*\*  \* The SalesPrint object to be decorated.  \*/  protected SalesPrint parentPrinter;  /\*\*  \* @see sales.print.printer.SalesPrint#print()  \*/  public abstract String print();  } |
| **MonthlySalesPrinter.java**  **월간 매출 출력** |
| public class MonthlySalesPrinter extends PrintDecorator{  /\*\*  \* Constructor.  \* @param printer The SalesPrint object to be decorated.  \*/  public MonthlySalesPrinter(SalesPrint printer) {  this.parentPrinter = printer;  }  @Override  public String print() {  // TODO: need working source code for printing monthly sales  StringBuilder sb = new StringBuilder();  sb.append(parentPrinter.print());  sb.append("\n");  sb.append("Monthly sales: \n");  List<MonthlySalesDto> monthlySales = new ArrayList<>();  for (Sales sales : parentPrinter.getSales()) {  MonthlySalesDto search = monthlySales.stream()  .filter(e -> e.getMonth() == sales.getTimestamp().getMonth())  .filter(e -> e.getYear() == sales.getTimestamp().getYear())  .findFirst()  .orElse(null);  if (search == null) {  monthlySales.add(MonthlySalesDto.builder()  .month(sales.getTimestamp().getMonth())  .year(sales.getTimestamp().getYear())  .sales(sales.getTransaction())  .build());  } else {  search.setSales(search.getSales() + sales.getTransaction());  }  }  monthlySales.forEach(e -> sb.append(e.getYear()).append("/").append(e.getMonth().getValue())  .append(": ").append(e.getSales()).append("\n"));  return sb.toString();  }  }  @Builder @Getter @Setter  class MonthlySalesDto {  private int year;  private Month month;  private Long sales;  } |

|  |  |
| --- | --- |
| **5.4.5** | **결과 화면** |

|  |
| --- |
| 회계 정보를 출력하고 싶은 형식을 선택하세요.  1: 월별 회계 정리본  2: 연별 회계 정리본  3: 종료  선택: **1**  회계 정보를 출력하고 싶은 형식을 선택하세요.  1: 월별 회계 정리본  2: 연별 회계 정리본  3: 종료  선택: **2**  회계 정보를 출력하고 싶은 형식을 선택하세요.  1: 월별 회계 정리본  2: 연별 회계 정리본  3: 종료  선택: **3**  --매출 출력 종료--  Monthly sales:  2021/9: 27500  2020/7: 15000  2020/6: 162000  2020/5: 177000  2020/3: 30000  2020/1: 27000  Yearly sales:  2021: 27500  2020: 411000 |

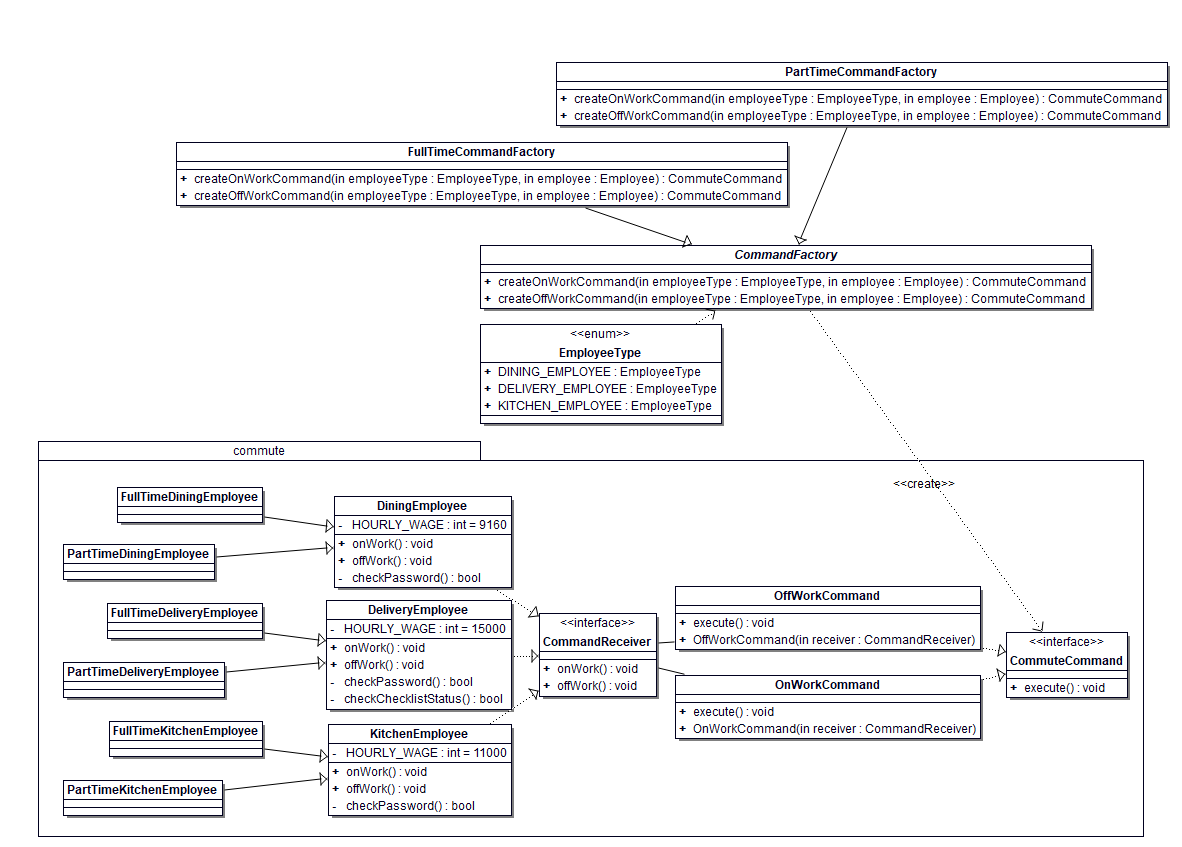
|  |  |
| --- | --- |
| **5.5** | **직원 등록** |

|  |  |
| --- | --- |
| **사용 디자인 패턴** | 팩토리 패턴 (팩토리 메서드 패턴) |
| **패턴 소개** | 팩토리 메서드 패턴을 이용하면 클래스의 인스턴스를 만드는 일을  서브클래스에게 맡기는 것. |
| **적용 계획** | 객체에 상태 사용자가 아르바이트생, 정직원 여부에 따라 추가/ 변경/ 삭제를 통해 직원을 관리하고 new사용을 최소화 하는 것이 핵심이다.  직원 근퇴 변경에 커맨드 패턴과 상호작용하며 처리한다. |
| **패턴 적용 방식** | |
| **1. 직원 생성을 담당한다. (CommandFactory)**  A. PartTimeCommandFactory : 아르바이트생 직원 생성을 담당한다.  B. FullTimeCommandFactory : 정직원 직원 생성을 담당한다.  **2. 생성된 직원 근무형태를 구분한다. (** **Dining Employee, DeliveryEmployee, KitchenEmployee)**  A. (HOURLY\_WAGE : int = 1950) : 홀직원의 경우 시급 9160원 급여를 받게 된다.  B. (HOURLY\_WAGE : int = 15000) : 배달직원의 경우 시급 15,000원을 받게 된다.  C. (HOURLY\_WAGE : int = 11,000) : 조리사의 경우 시급 11,000원을 받게 된다.  **3. 최종 근무형태에 따라 직급을 분류한다.**  A. FullTimeDiningEmployee : 홀에서 근무하는 정직원을 의미한다.  B. PartTimeDiningEmployee : 홀에서 근무하는 아르바이트생을 의미한다.  C. FullTimeDeliveryEmployee: 배달을 담당하는 정직원을 의미한다.  D. PartTimeDeliveryEmployee: 배달을 담당하는 아르바이트생을 의미한다.  E. FullTimeKitchenEmployee : 조리를 담당하는 정직원을 의미한다.  F. PartTimeKitchenEmployee : 조리를 담당하는 아르바이트생을 의미한다. | |
| **프로젝트 적용 예시** | * 홀에서 근무하는 정직원을 생성하는 경우   1. CommandFactory 클래스에서 상속받는 FullTimeCommandFactory  클래스에서 createOnWorkCommand() 정직원을 객체를 생성한다.  2. DiningEmployee 클래스에서 HOURLY\_WAGE 인스턴스를 통해  9160원 시급이 결정된다.  3. FullTimeDiningEmployee 클래스를 통해 최종적으로 ‘홀에서 근무하는  정직원’이 생성된다. |

|  |
| --- |
| **UseCase** |
|  |

|  |
| --- |
| **상태 Diagram** |
| 텍스트, 모니터, 스크린샷, 화면이(가) 표시된 사진  자동 생성된 설명 |

|  |  |
| --- | --- |
| **5.5.1** | **패턴 UML** |



|  |  |
| --- | --- |
| **5.5.2** | **주요 클래스** |

|  |  |
| --- | --- |
| **신규 직원 (정직원, 아르바이트생) 생성 클래스** | |
|  | |
| - 해당 클래스는 CommandFactory 클래스를 상속받는 FullTimeCommandFactory에서 정직원을 생성하게되며,  PartTimeCommandFactory 클래스에서는 아르바이트생을 생성하게 된다. | |
| **+ createOnWorkCommand()** | 직원을 생성할 때 직원 정보를 근퇴 관리 서비스에 등록하기 위해 출근을 제어하는 클래스를 생성하는 메소드 |
| **+ createOffWorkCommand()** | 직원을 생성할 때 직원 정보를 근퇴 관리 서비스에 등록하기 위해 퇴근을 제어하는 클래스를 생성하는 메소드 |

|  |  |
| --- | --- |
| **직무별 직원 생성 클래스** | |
|  | |
| - 해당 클래스 (Dining,Delivery,Kitchen)Employee 클래스 경우 시급, 업무에 따라 직무가 구별된다.  - 직무 선택이후 FullTimeDiningEmployee, PartTimeDiningEmployee (정직원, 아르바이트생)이 구별된다. | |
| **HOURLY\_WAGE** | 직무별 시급을 담당한다. |
| **onWork()** | 근무를 시작하는 메서드이다. |
| **offWork()** | 근무를 종료하는 메서드이다. |
| **checkPassword()** | 본인 확인을 위한 패스워드 확인 메서드이다. |

|  |  |
| --- | --- |
| **5.5.3** | **기능 설명** |

|  |  |
| --- | --- |
| **소주제** | 직원 조회 |
| **ID** | SFR-0501 |
| **클래스** | employee/EmployeeManagement.java |
| **Actor** | 사장 |
| **목적** | 직원 현황을 조회 할 수 있다. |

|  |  |
| --- | --- |
| **소주제** | 직원 등록 |
| **ID** | SFR-0502 |
| **클래스** | employee/EmployeeManagement.java |
| **Actor** | 사장 |
| **목적** | 직원 등록을 수행할 수 있다. |

|  |  |
| --- | --- |
| **5.5.4** | **주요 코드** |

|  |
| --- |
| **EmployeeManagement.java** |
| protected void createEmployee() {  // Get employee information from console  final String name = Console.getInput("직원 이름을 등록하세요: ");  final String password = Console.getInput("직원 비밀번호를 등록하세요: ");  // check name and password is not null  if (name == null || password == null) {  System.out.println("Invalid input.");  return;  }  // Create employee  Employee employee = Employee.builder()  .name(name)  .password(password)  .build();  // generate index of command slot for employee commute management  // find empty slot  try {  CommuteManager commuteManager = CommuteManager.getInstance();  int index = commuteManager.findEmptyIndex();  System.out.println("로그인을 위한 개인 번호는 " + index + " 입니다. 잊지 않게 주의하세요!");  employeeRepository.addEmployee(employee);  } catch (NoSpaceForCommandException e) {  System.err.println("There is no empty slot for new employee.");  return;  }  } |

|  |
| --- |
| **factory/FullTimeCommandFactory.java** |
| public class FullTimeCommandFactory extends EmployeeCommandFactory {  @Override  public CommuteCommand createOnWorkCommand(EmployeeType employeeType, Employee employee) {  return new OnWorkCommand(  FullTimeEmployeeType.getFullTimeEmployeeType(employeeType)  .getCommandReceiverSupplier()  .apply(employee));  }  @Override  public CommuteCommand createOffWorkCommand(EmployeeType employeeType, Employee employee) {  return new OffWorkCommand(  FullTimeEmployeeType.getFullTimeEmployeeType(employeeType)  .getCommandReceiverSupplier()  .apply(employee));  }  @AllArgsConstructor @Getter  private enum FullTimeEmployeeType {  FULL\_DINING\_EMPLOYEE(EmployeeType.DINING\_EMPLOYEE, FullTimeDiningEmployee::new),  FULL\_DELIVERY\_EMPLOYEE(EmployeeType.DELIVERY\_EMPLOYEE, FullTimeDeliveryEmployee::new),  FULL\_KITCHEN\_EMPLOYEE(EmployeeType.KITCHEN\_EMPLOYEE, FullTimeKitchenEmployee::new),  ;  private final EmployeeType employeeType;  private final Function<Employee, CommandReceiver> CommandReceiverSupplier;  public static FullTimeEmployeeType getFullTimeEmployeeType(EmployeeType employeeType) {  return Arrays.stream(FullTimeEmployeeType.values())  .filter(fullTimeEmployeeType -> fullTimeEmployeeType.getEmployeeType().equals(employeeType))  .findAny()  .orElseThrow(() -> new IllegalArgumentException("EmployeeType is not supported."));  }  }  } |

|  |  |
| --- | --- |
| **5.5.5** | **결과 화면** |

|  |
| --- |
| 직원을 데이터베이스에 주가합니다.  직원 이름을 등록하세요: **조진혁**  직원 비밀번호를 등록하세요: **paswrd**  직원의 직급을 선택하세요(정규직/비정규직): **비정규직**  - 홀  - 부엌  - 배달  직원의 직종을 선택하세요: **부엌**  로그인을 위한 개인 번호는 3 입니다. 잊지 않게 주의하세요!  [1]: 직원 추가  [2]: 직원 조회  [3]: 출퇴근 관리  [4]: 모든 직원 월급 조회  [5]: 종료  원하는 동작을 선택하세요: **2**  직원 목록을 조회합니다.  Employee{name='sajang', password='password', admin=false}  Employee{name='정우성', password='password', admin=false}  Employee{name='이순신', password='password', admin=false}  Employee{name='박봉팔', password='password', admin=false}  Employee{name='조진혁', password='paswrd', admin=false} /\* This record is created one \*/ |

|  |  |
| --- | --- |
| **5.6** | **직원 근퇴 변경** |

|  |  |
| --- | --- |
| **사용 디자인 패턴** | 커맨드 패턴 (Command pattern) |
| **패턴 소개** | 단일행위에 집중(근퇴)하고 그 단일 행위에 대한 작업을 요청한 부분과 작업하는 쪽을 분리하며 캡슐화 하여 정의한다 |
| **적용 계획** | 직원의 출근과 퇴근 기능을 Invoker를 통해 요청하면 직원의 종류에 따라 명령을 처리한다. 또한 직원의 직종에 따라 시급이 다르기 때문에 각 단일 행위에 따라 캡슐화 한다. 직원 등록 기능의 팩토리 패턴과 상호작용하며 처리한다. |
| **패턴 적용 방식** | |
| 1. 직원의 근퇴 동작을 관리하는 Invoker 역할을 한다 (CommuteManager) 2. 실행 메서드를 interface로 선언한 Command 역할을 한다 (CommuteCommand) 3. 출근과 퇴근 기능을 실행하는 Concrete Command 역할을 한다 (OnWorkCommand/OffWorkCommand) 4. 각 ConcreteCommand를 통한 출근과 퇴근 기능을 처리하는 Receiver 역할을 한다 (CommandReceiver) | |
| **프로젝트 적용 예시** | * **배달 직원이 출근 했을 때**  1. 배달 직원이 출근 기능인 onWork() 메서드를 실행한다, 2. 직원 정보를 가지고 있는 Employee 객체에서 배달 직원을 사용한다. 3. 배달 직원의 월급 데이터를 가지고 시급 계산에 이용한다. |

|  |
| --- |
| **UseCase** |
|  |

|  |  |
| --- | --- |
| **5.6.1** | **패턴 UML** |



|  |  |
| --- | --- |
| **5.6.2** | **주요 클래스** |

|  |  |
| --- | --- |
| **동작 관리 클래스** | |
|  | |
| **- MAN\_EMPLOYEE : int** | 최대 직원 인원 수 (인원 수는 변경이 가능하다.) |
| **+ setCommuteCommand**  **(index : int,**  **onWork : CommuteCommand,**  **offWork : CommuteCommand)** | 출근과 퇴근의 동작을 통해 Command를 등록하는 메서드 |
| **+ onWork(index : int)** | 출근 동작 메서드 |
| **+ offWork(index : int)** | 퇴근 동작 메서드 |

|  |
| --- |
| **실행 클래스(interface)** |
|  |
| 실행 명령문을 가지고 있는 Command 역할을 하는 interface 클래스 |

|  |  |
| --- | --- |
| **근퇴 설정 클래스** | |
|  | |
| **+ OnWorkCommand**  **(receiver : CommandReceiver)** | 출근 Command 메서드 |
| **+ OffWorkCommand**  **(receiver : CommandReceiver)** | 퇴근 Command 메서드 |

|  |  |
| --- | --- |
| **직원 클래스** | |
|  | |
| **+ HOURLY\_WAGE : int** | 직원의 분류에 따라 달라지는 월급 값 |
| **DeliveryEmployee 클래스** | 배달 직원 클래스 |
| **DiningEmployee 클래스** | 홀 직원 클래스 |
| **KitchenEmployee 클래스** | 주방 직원 클래스 |

|  |  |
| --- | --- |
| **5.6.3** | **기능 설명** |

|  |  |
| --- | --- |
| **소주제** | 근퇴 조회 |
| **ID** | SFR-0504 |
| **클래스** | employee/EmployeeManagement.java |
| **Actor** | 직원 |
| **목적** | 직원이 근퇴 변경하기 전에 자신의 현재 근무 상황을 출력하여 확인한다. |

|  |  |
| --- | --- |
| **소주제** | 직원 근퇴 변경 |
| **ID** | SFR-0505 |
| **클래스** | 퇴근 : employee/commute/command/OffWorkCommand.java  출근 : employee/commute/command/OnWorkCommand.java |
| **Actor** | 직원 |
| **목적** | 직원이 출근 또는 퇴근 시 자신의 근무 상황을 변경한다. |

|  |  |
| --- | --- |
| **5.6.4** | **주요 코드** |

|  |
| --- |
| **CommuteManager.java** |
| public class CommuteManager implements Manager {  private final int MAX\_EMPLOYEE = 10;  private final CommuteCommand[] onWorkCommands = new OnWorkCommand[MAX\_EMPLOYEE];  private final CommuteCommand[] offWorkCommands = new OffWorkCommand[MAX\_EMPLOYEE];  // Use singleton pattern  private static CommuteManager instance = new CommuteManager();  public static CommuteManager getInstance() {  if (instance == null) {  instance = new CommuteManager();  }  return instance;  }  public void setCommuteCommand(int index, CommuteCommand onWorkCommand, CommuteCommand offWorkCommand) {  onWorkCommands[index] = onWorkCommand;  offWorkCommands[index] = offWorkCommand;  }  …중략  } |

|  |
| --- |
| **CommuteCommand.java** |
| /\*\*  \* This class work as a command role on command pattern.  \* execute() method is used to execute the command.  \*/  public interface CommuteCommand {  /\*\*  \* This method is used to execute the command.  \* Please override this method and javadoc on each concrete command.  \*/  public void execute();  } |

|  |
| --- |
| **OnWorkCommand.java** |
| public class OnWorkCommand implements CommuteCommand{  private CommandReceiver receiver;  @Override  public void execute() {  receiver.onWork();  }  } |
| **OffWorkCommand.java** |
| public class OffWorkCommand implements CommuteCommand{  private CommandReceiver receiver;  @Override  public void execute() {  receiver.offWork();  }  } |

|  |
| --- |
| **CommandReceiver.java** |
| /\*\*  \* This class work as a command role on command pattern.  \* execute() method is used to execute the command.  \*/  public interface CommuteCommand {  /\*\*  \* This method is used to execute the command.  \* Please override this method and javadoc on each concrete command.  \*/  public void execute();  } |

|  |  |
| --- | --- |
| **5.6.5** | **결과 화면** |

|  |
| --- |
| **출근 등록** |
| 등록할 때 부여된 개인 번호를 통해 출근/퇴근 확인을 해주시기 바랍니다.  1. 출근 등록  2. 퇴근 등록  **1**  출근 등록을 위해 등록할 개인 번호를 입력해주세요.  **0**  Please enter password:  **password**  Password is correct  헬멧을 쓰고 있습니까?(Y/N): Y  주유가 정상적으로 되어 있습니까?(Y/N): Y  GPS가 정상적으로 작동하고 있습니까?(Y/N): Y  속도 제한기가 정상적으로 작동 중입니까?(Y/N): Y  **Delivery Employee 정우성 is on work**  **Current time : 2022-06-11T17:49:19.258337700** |

|  |
| --- |
| **퇴근 등록** |
| 등록할 때 부여된 개인 번호를 통해 출근/퇴근 확인을 해주시기 바랍니다.  1. 출근 등록  2. 퇴근 등록  **2**  퇴근 등록을 위해 등록할 개인 번호를 입력해주세요.  **1**  Please enter password:  **password**  Password is correct  **Delivery Employee 이순신 is off work**  **Work time : 00:00:24**  **Wage : 0** |

|  |  |
| --- | --- |
| **6** | **다이어그램** |

|  |  |
| --- | --- |
| **6.1** | **UseCase 다이어그램** |

|  |
| --- |
| **Actor 관계** |
|  |
| **UseCase에 사용되는 Actor의 관계를 나타낸다** |
| * + - 사용자 엑터는 직원과 사장으로 이루어진다     - 사용자는 직원과 사장을 포함하는 엑터이다     - 보조 엑터는 유지보수 담당자를 가지고 있다 |

|  |
| --- |
| **각 Actor의 기능 UseCase** |
|  |
| **각 Actor가 어떤 기능을 사용하는지 대략적인 관계를 나타낸다** |

|  |
| --- |
| **주문 관리** |
|  |
| **사용자가 주문관리 기능을 사용할 때 수행되는 상세 UseCase** |

|  |
| --- |
| **매출 관리** |
|  |
| **사용자가 매출 관리 기능을 사용할 때 수행되는 상세 UseCase** |

|  |
| --- |
| **매출 관리 (사장)** |
|  |
| **사장이 매출 관리의 매출 조회 기능을 사용할 때 수행되는 상세 UseCase**  **사장이 매출 관리의 잔액 관리 기능을 사용할 때 수행되는 상세 UseCase** |
| * 사장이 매출 관리를 사용하며, 하나의 UseCase로 표현하면 그림이 커지기 때문에 매출 조회와 잔액 관리를 따로 분리하여 표현하였다. |

|  |
| --- |
| **직원 관리** |
| 제어이(가) 표시된 사진  자동 생성된 설명 |
| **직원 또는 사장이 직원 관리 기능을 사용할 때 수행되는 상세 UseCase** |
| * + - 직원과 사장이 같은 직원 관리 기능을 사용하지만, 각자 수행되어야 하는 상세 기능은 다르기 때문에 엑터별로 따로 분류하였다 |

|  |
| --- |
| **메뉴 관리** |
|  |
| **사장이 메뉴 관리 기능을 사용할 때 수행되는 상세 UseCase** |

|  |
| --- |
| **테이블 관리** |
|  |
| **사장이 테이블 관리 기능을 사용할 때 수행되는 상세 UseCase** |

|  |  |
| --- | --- |
| **6.2** | **Sequence 다이어그램** |

|  |
| --- |
| **주문 관리** |
| 텍스트이(가) 표시된 사진  자동 생성된 설명 |
| **사용자가 주문관리 기능을 사용할 때 흐름을 파악하기 위한 시퀀스 다이어그램** |
| * + - 직원이 주문기계를 통해 주문을 입력할 수 있다.     - 직원이 입력한 주문정보를 표시해주며, 카드, 상품권, 현금결제에 대해 선택한다     - 해당 결제를 수행하기위해 각단계의 유효성검사를 과정을 거친다.     - 최종적으로 결제 수행된 결과가 직원 주문기계(POS 기계)로 결과를 보여준다. |

|  |  |
| --- | --- |
| **6.3** | **활동 다이어그램** |

|  |
| --- |
| **신규 직원 생성 활동 다이어그램** |
| 텍스트, 모니터, 스크린샷, 화면이(가) 표시된 사진  자동 생성된 설명 |
| **사용자가 신규 직원을 상세히 명세한 활동 다이어그램** |
| * + - 신규 직원 생성 시 구분 기준점을 (정직원, 아르바이트생) 기점을 두었다.     - 정직원의 경우 ‘시간외 근무 가능’ 메시지를 출력 후 근무 시간을 측정한다.     - 아르바이트생일 경우 ‘시간외 근무 불가능’ 메시지 출력 후 일일 근무시간 최대 8시간 근무만 가능하다 |
| **정직원 생성 활동 다이어그램** |
| 사용자가 정직원을 상세히 명세한 활동 다이어그램 |
| 텍스트, 모니터, 스크린샷, 화면이(가) 표시된 사진  자동 생성된 설명 |
| * + - 시스템에서 정직원일 경우 시간외 근무 메시지 가능 여부를 출력한다.     - 시간외 근무 메시지 출력된 정보를 정직원을 추가한다.     - 근무 버튼을 선택하면 (취소, 확인)을 선택한다.       * 확인을 선택하면 (근무시간, 시간외 근무시작 메시지)를 출력한다.       * 취소 버튼을 클릭할 경우 메인 화면으로 바로 이동한다.     - 근무 종료 버튼을 눌렀을 경우 시간외 근무 종료 메시지를 출력한다. * 3가지 일이 동시에 처리되는 동기 (근무시간 측정 종료, 시간외 근무 일당 DB 반영, 근무 일단 DB 반영) 업무가 동시에 일어나게 된다.   + - 모든 업무가 종료되면 메인 화면으로 이동한다. |

|  |
| --- |
| **아르바이트생 활동 다이어그램** |
| 사용자가 아르바이트생 직원을 상세히 명세한 활동 다이어그램 |
| 텍스트, 모니터, 스크린샷, 화면이(가) 표시된 사진  자동 생성된 설명 |
| * + - 시스템에서 만약 정직원이 아닐 경우 작동된다.     - 시간외 근무 불가 메시지를 출력한다.     - 아르바이트생 추가기능을 수행한다.     - 아르바이트생은 근무버튼을 선택한다.  1. 만약 여기서 아르바이트생이 근무 취소를 하면 메인 화면으로 이동한다. 2. 아르바이트생이 확인을 하면 근무시간이 측정된다 3. 아르바이트생이 근무종료를 누르면 동시에(근무시간 측정 종료, 근무 일단 DB반영)을 진행한다 4. 작업이 모두 마치고 병합되어 메인 화면으로 이동한다. |

|  |
| --- |
| **시스템 활동 다이어그램** |
| 사용자가 시스템일 경우 상세히 명세한 활동 다이어그램 |
| 텍스트, 모니터, 스크린샷, 화면이(가) 표시된 사진  자동 생성된 설명 |
| 1. 시스템이 이전에 신규 직원 생성 활동 버튼이 작동된다.  2. 직원 추가 버튼을 사용자가 누른다.  3. 정직원과 아르바이트생 차이점을 출력한다.  4. 아르바이트생 과 정직원 차이점을 사용자에게 알려준다.  5. 사용자는 다음 분기점에서 정직원인지 아닌지 여부를 선택하여야 한다. |

|  |  |
| --- | --- |
| **6.4** | **상태(State) 다이어그램** |

|  |
| --- |
| **주문 관리 전체 다이어그램** |
|  |
| **사용자가 주문관리를 통한 상태 다이어그램** |
| * + - 상태 다이어그램은 “ 주문 관리” 기능을 다이어그램으로 나타냈다.     - 주문대기를 기점으로 (주문 수락, 완료,취소,삭제,변경)을 수행한다.     - 주문 관리의 경우 시작점에서 프로그램이 시작된다고 가정하에 작동된다. |

|  |
| --- |
| **주문 수락 다이어그램** |
|  |
| **사용자가 주문관리를 통한 상태 다이어그램 (주문 수락)** |
| 1. 프로그램 시작 후 주문대기 상태에서 <<주문 입력>> 이벤트가 감지되면 주문 수락 대기로 이동한다.    1. 주문 수락 대기에서 << 확인 버튼을 클릭하면 >> 주문 수락으로 이동한다.       1. 주문 수락 수락에서 주문 수락 출력>> 이벤트를 수행하고 주문 대기로 돌아간다.    2. 주문 수락대기에서 <취소 버튼 클릭>> 이벤트가 감지되면 다시 주문대기로 이동한다. |

|  |
| --- |
| **주문 완료 다이어그램** |
|  |
| **사용자가 주문관리를 통한 상태 다이어그램 (주문 완료)** |
| 1. 사용자가 << 조리 완료 입력 >> 이벤트가 입력되면 주문 완료로 이동한다. 2. << 완료 버튼 클릭 >> 이벤트가 입력되면 주문 완료 처리로 이동한다. 3. 주문 완료 처리에서 주문 대기로 이동한다. |

|  |
| --- |
| **주문 취소 및 변경 다이어그램** |
|  |
| **사용자가 주문관리를 통한 상태 다이어그램 (주문 취소 대기, 주문 수정 대기)** |
| 1. 주문 대기 상태에서 << 주문 출력, 버튼>> 이벤트가 발생하면 주문 선택으로 이동한다.    1. 주문 선택에서 << 취소 >> 이벤트가 발생하면 주문 대기로 이동한다. 2. 주문 선택에서 << 주문 취소 선택>> 이벤트가 발생하면 주문 취소 대기로 이동한다.    1. 주문 취소 대기에서 <<확인>> 이벤트가 감지되면 주문 삭제로 이동하고, 주문 대기로 이동한다    2. 주문 취소 대기에서 << 취소>> 이벤트가 발생하면 주문 선택으로 이동한다. 3. 주문 선택에서 이후 << 주문 변경 선택>> 이벤트가 감지되면 주문 수정 대기로 이동한다.    1. 주문 수정 대기 상태에서 << 변경 정보 입력 >> 이벤트가 감지되면 주문 변경 이동하고 문 대기 상태로 이동한다.    2. 주문 수정 대기 상태에서 << 취소>> 이벤트가 감지되면 주문 선택으로 이동하여 주문 대기로 이동한다. |

|  |
| --- |
| **주문 관리 다이어그램** |
|  |
| **사용자가 주문관리를 통한 상태 다이어그램 (주문 대기)** |
| 1. 시작점에서 프로그램이 시작된다. 2. << 주문 대기 버튼 선택 >> 이벤트가 감지되면 주문 대기로 이동한다. 3. 주문 대기에서 각 입력 상황별로 작동한다.    1. 모든 기능 수행 후 << 주문 대기 정지 선택>> 이벤트가 발생하면 모듈 종료로 이동한다.      * 1. 종료점에서 프로그램이 종료된다. |

|  |  |
| --- | --- |
| **7** | **개발 환경** |

|  |  |
| --- | --- |
| **7.1** | **개발 도구** |

|  |  |  |
| --- | --- | --- |
|  | | **도구 및 테스트 담당** |
| **IDE** | IntelliJ IDEA | 조진혁, 박상현, 최인수 |
| NetBeans | 이수찬 |
| **OS 환경** | Windows 10 | 조진혁, 박상현 |
| Windows 11 | 이수찬 |
| Mac OS | 최인수 |

|  |  |
| --- | --- |
| **7.2** | **회의** |

|  |  |
| --- | --- |
| **대면 회의** | **비대면 회의** |
| **동의대학교 정보공학관** | **Discord** |
|  |  |

|  |
| --- |
| **회의록 작성** |
|  |

|  |  |
| --- | --- |
| **8** | **기술 관리 방법** |

|  |  |
| --- | --- |
| **8.1** | **변경 관리** |

|  |  |
| --- | --- |
| **결제 서비스** | 새로운 결제 서비스 제공자가 추가 / 기존의 결재 서비스 제공자의 종료에 따른 결재 관리 시스템 추가 / 제거하였다 |
| **영수증 출력 기능** | 영수증 형식의 변경이 있을 경우 기존 소스코드의 변경이 필요하다. |
| **메뉴 관리** | 메뉴에 추가적인 정보(알레르기 등)을 제공해야 할 때 클래스 filed의 변경이 있을 수 있다. |
| **주문 관리** | 프로그램 실행 환경에 따라 주문이 데이터베이스에 저장될 수도, 메모리에 저장될 수도 있으며 저장 매체 또한 개발 중 변경될 수 있다. |

|  |  |
| --- | --- |
| **8.2** | **형상 관리** |

|  |
| --- |
| **Github 형상 관리(1)** |
|  |

|  |
| --- |
| **Github Repository 링크** |
| <https://github.com/Nifskor/Design_pattern_Java> |

|  |
| --- |
| **Github 형상 관리(2)** |
|  |

|  |  |
| --- | --- |
| **8.3** | **위험 관리** |

|  |  |
| --- | --- |
| **개발 기간 부족** | |
| 프로젝트 개발에서의 내/외부 요소에 의한 개발 지연 | |
| **위험 발생 가능성** | 보통 (25% ~ 50%) |
| **원인** | 다른 프로젝트와의 동시의 진행으로 인한 일정 관리 실패, 외부 라이브러리의 에러 연결 및 데이터베이스 연결 지연 등 문제로 개발 지연 등 |
| **해결 방법** | 요구사항 조정 / 일정 재검토 등을 통해 마감일에 위반되지 않도록 조치한다. |

|  |  |
| --- | --- |
| **팀원 간 불화** | |
| 팀원 간의 의견 충돌 / 불화에 의한 위기 | |
| **위험 발생 가능성** | 매우 낮음 ( 0% ~ 10%) |
| **원인** | 프로젝트 진행에 대한 팀원 간의 의견 충돌, 사적인 감정 격화 등 팀원과의 관계가 악화될 수 있는 요소 |
| **해결 방법** | 요구사항 조정 / 일정 재검토 등을 통해 마감일에 위반되지 않도록 조치한다. |

|  |  |
| --- | --- |
| **요구사항 변경** | |
| 기존에 정의된 요구사항에 추가적으로 요구사항이 증가 | |
| **위험 발생 가능성** | 높음 (50% ~ 75%) |
| **원인** | 클라이언트, 팀원 내부 회의 등으로 부족한 부분을 추가적으로 필요한 요구 사항의 추가 |
| **해결 방법** | 요구사항 조정 / 일정 재검토 등을 통해 마감일에 위반되지 않도록 조치한다. |

|  |  |
| --- | --- |
| **9** | **성능 시험 방법** |

|  |  |
| --- | --- |
| **9.1** | **단위 테스트** |

|  |  |  |  |
| --- | --- | --- | --- |
| **개발 환경 (IDE)** | **테스트 담당** | **테스트 확인** | **비고** |
| **IntelliJ IDEA** | 조진혁 | O | 프로젝트가 아닌 폴더 경로를 통해 열었을 시 발생하는 오류 발견 |
| 최인수 | O | - |
| 박상현 | O | - |
| **NetBeans** | 이수찬 | O | 한글이 깨지는 오류 발견 |

|  |  |
| --- | --- |
| **9.2** | **통합 테스트** |

|  |
| --- |
| **GitHub 버전 관리** |
|  |
| **코드 리뷰** |
|  |

|  |  |
| --- | --- |
| **9.3** | **시스템 테스트** |

|  |  |  |  |
| --- | --- | --- | --- |
| **운영체제 (OS)** | **테스트 담당** | **테스트 확인** | **비고** |
| **Windows 10** | 조진혁 | O | - |
| 박상현 | O | - |
| **Windows 11** | 이수찬 | O | - |
| **Mac OS** | 최인수 | O | - |

|  |  |
| --- | --- |
| **10** | **느낀 점** |

|  |  |
| --- | --- |
| **조진혁** | 기존에 약한 분야인 문서화, 설계 과정에서의 체계 등에 대해서 고민할 기회가 되었다. 다이어그램 작성 등의 설계 과정에서의 활동 또한 굉장히 가치가 있었다고 생각한다. 다만, 팀원과의 협업에 대한 경험이 부족하여 일정/버전 관리 등에 미흡했던 점은 아쉬웠다. 다음에 기회가 된다면 이런 방향으로도 경험을 쌓아 보고 싶다. |
| **이수찬** | 이번 수업을 통해서 평소에 유지 보수성이 떨어지던 코드에 대해서 반성하는 시간을 갖게 되었다. 앞으론 적절한 상황에 디자인 패턴을 활용하여 유지보수까지 고려하며 설계를 해야겠다고 느꼈다. 또한 실제 Product가 나오기까지의 과정을 직접 팀원들과 함께 부딪혀 보면서 코드도 중요하지만, 코드가 짜이기까지의 과정이 더 중요하다는 생각을 하였다. |
| **최인수** | 이번 디자인 패턴 프로젝트를 통해 느낀 점은, 처음에는 과연 디자인 패턴이 중요할까? 라는 생각이 들었다. 어떻게 프로그램을 효율적으로 유지 보수하게 쉽게 프로그램을 설계 및 코딩 할 것 인가라는 관점에서 정말 중요한 것을 느끼게 되었다. 특히 프로젝트를 수행하면서, 스트레티지, 데코레이터 개념적인 면에서 완전히 이해하는데 상당히 어려웠다. 추후 학기가 끝나더라도 별도로 추가로 꼭 공부해서 앞으로는 구동을 위한 코드보다, 유지보수. 테스팅을 위한 면도 같이 고려하여 프로그램을 짜야겠다는 생각을 하게 되는 좋은 시간이었다. |
| **박상현** | 수업을 통해 배운 디자인 패턴을 가지고 여름방학 때 진행했던 개인 프로젝트에 설계적 미흡함을 알게 되었다. 좋은 개발자가 되기 위해서는 눈앞의 코드 뿐만 아니라 유지보수를 생각하며 코드를 짜는 것이 좋을 거로 생각한다. 공부하면서 2학년 때 부족했던 캡슐화에 대해 더 상세하게 알 수 있었던 가치 있는 시간이 되었다. 앞으로 프로그램 개발을 해 가면서 더욱 다양한 패턴을 적용해보고 싶고, 실제로 프로그램 개발 이후에도 계속 유지보수와 패치를 통해 쉽게 건드릴 수 있는 코드를 작성하고 싶다. |

|  |  |
| --- | --- |
| **11** | **참고문헌 및 부록** |

1. [쉽게 배우는 소프트웨어 공학. 김치수. 2021](http://www.kyobobook.co.kr/product/detailViewKor.laf?mallGb=KOR&ejkGb=KOR&barcode=9791156645429)
2. [Head First Design Patterns: 스토리가 있는 패턴 학습법. 에릭 프리먼. 2005](https://www.hanbit.co.kr/store/books/look.php?p_code=B9860513241)
3. [ECOMMERCE PLATFORMS](https://ecommerce-platforms.com/)
4. [부산 지역화폐 동백전](https://busandong100.kr/)
5. [창원사랑상품권(누비전)](https://www.changwon.go.kr/biz/contents.do?mId=0605020100)
6. [Pull Request를 통해 코드리뷰(Code Review)하는 법. Gyun’s 개발일지. 2020](https://devlog-wjdrbs96.tistory.com/231)
7. [Loyverse POS System](https://loyverse.com)
8. [KOSA 사업지원 – SW업 대가산정 가이드(2022 개정판)](https://www.sw.or.kr/site/sw/ex/board/View.do?cbIdx=276&bcIdx=51935&searchExt1=)
9. [KOSA 사업지원 - SW사업대가 산정 방식별 엑셀 템플릿 (2022 개정판)](https://www.sw.or.kr/site/sw/ex/board/View.do?cbIdx=276&bcIdx=49636&searchExt1=)
10. [Github Docs. Github. 2022](https://docs.github.com/en)
11. [코리아텍, 실시간 식당만족도 측정 소프트웨어 운영. 채원상. 2018](http://www.goodmorningcc.com/news/articleView.html?idxno=94485)
12. [의료기기 소프트웨어 위험관리 적용 안내서. 디지털헬스케어 소프트웨어 시험평가 센터. 2019](https://stl.ktl.re.kr/web/contents/DataCentre.do?schM=view&id=316)