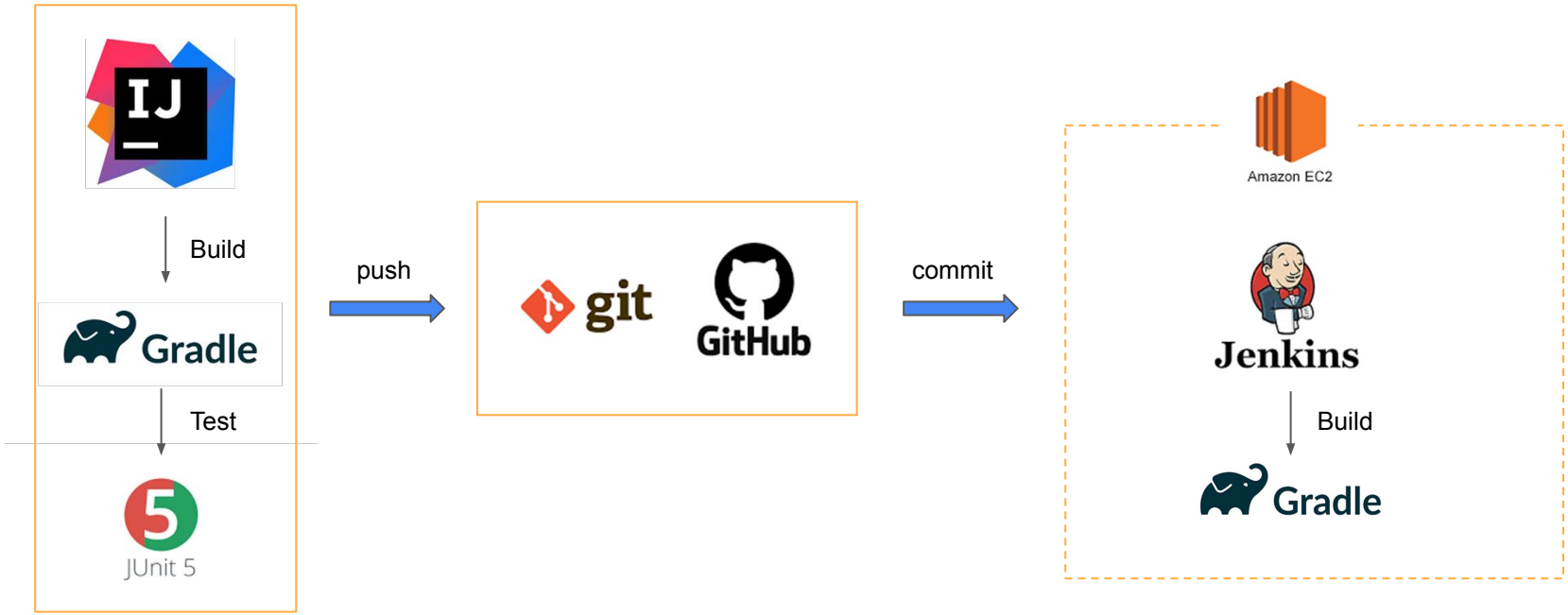
OOPT 2050
OOI

Team 2 강상민 김재윤 이찬양 한승현

개발환경 - CTIP 환경 & UT



System Test

Class	Test Case	Description	Pass
AdminDBManager	testCheckUser_ValidAccount	제공된 관리자 계정이 존재하고 올바른 비밀번호를 가지고 있는지 확인	P
	testCheckUser_InvalidAccount	제공된 관리자 계정이 존재하지 않는지 확인	P
	testCheckUser_WrongPassword	제공된 계정은 존재하지만 잘못된 비밀번호를 가지고 있는 경우인지 확인	P
	testCheckUser_EmptyFile	빈 데이터베이스 파일에서 계정을 확인하는지 확인. 파일이 비어있는 경우 어떤 계정도 유효하지 않은지 확인	P
DrinkDBManager	testHasDrink	hasDrink 메서드 검사. 주어진 음료 코드와 수량에 따라 재고가 있는지 여부를 확인할 뿐만 아니라 재고가 있는 경우와 없는 경우 모두 확인	P
	testGetDrink	getDrink 메서드 검사. 주어진 음료 코드와 수량에 따라 음료를 가져오는지 확인할 뿐만 아니라 재고가 충분한 경우와 부족한 경우 모두 확인	P
	testGetDrinkQuantity	getDrinkQuantity 메서드 검사. 주어진 음료 코드에 대한 재고 수량을 확인	P
	testSetDrink	setDrink 메서드 검사. 음료 재고 갱신 확인 및 존재하지 않는 음료 확인	P

System Test

Class	Test Case	Description	Pass
DrinkDBManager	testGetAllDrinkStatus	getAllDrinkStatus 메서드 검사. 모든 음료의 재고 현황 확인 및 첫 번째 음료 정상적으로 저장되어 있는지 확인	P
	testGetMenuList	getMenuList 메서드 검사. 전체 메뉴목록 확인 및 해당 메뉴에서 첫 번째 메뉴 확인	P
VerificationCodeDBManager	testCheckCode	checkCode 메서드를 검사. 코드가 존재하지 않는다면 null, 존재한다면 알맞은 코드인지 확인	P
	testSaveCode	saveCode 메서드 검사. 새로운 코드를 저장하고 저장한 코드가 올바르게 저장되었는지 확인. 이미 존재하는 코드를 저장하려 할 때 올바르게 처리되는지 확인.	P
ControllerMapper	getDrinkControllerTest	/drink 경로에 대한 컨트롤러가 DrinkController 클래스의 인스턴스인지 확인	P
	getMessageControllerTest	/message/send 경로에 대한 컨트롤러가 MessageController 클래스의 인스턴스인지 확인	P
	getPayControllerTest	/pay 경로에 대한 컨트롤러가 PayController 클래스의 인스턴스인지 확인	P
	getVerificationCodeControllerTest	/code 경로에 대한 컨트롤러가 VerificationCodeController 클래스의 인스턴스인지 확인	P

System Test

Class	Test Case	Description	Pass
ControllerMapper	notProperUrlTest	잘못된 경로 또는 빈 경로에 대한 요청이 들어왔을 때 null이 반환되는 지 확인.	P
	getAdminControllerTest	/admin/login 경로에 대한 컨트롤러가 AdminController 클래스의 인스턴스인지 확인	P
DVMSimulationServer	serverRunningTest	서버가 정상적으로 실행 중인지 확인. 서버에 대한 요청을 보내고, 응답 코드가 200인지 확인하여 서버가 정상적으로 실행 중인지를 검증	P
	getDrinkTest	클라이언트는 /drink 경로로 들어오는 GET 요청에 대한 응답이 올바른지를 확인. 서버에서 응답으로 반환한 음료 정보가 데이터베이스에서 가져온 음료 정보와 일치하는지를 검증	P
	reqDrinkQuantityTest	응답으로 받은 메세지에서 요청한 음료의 재고 수량이 일치하는지 확인. 테스트를 위해 JSON 메세지를 만들어 서버로 송신 후 서버는 해당 요청에 대한 응답을 JSON 형식으로 반환함. 수신한 메세지와 송신한 메세지를 비교하여 테스트 성공 여부 확인.	P
	reqAdvancePaymentTest	클라이언트는 /pay/reqAdvancePayment에 JSON 형식의 인증코드 요청을 보내면, 서버는 해당 응답에 대한 반환을 함. 서버가 해당 요청을 올바르게 처리하여 선결제 기능이 제대로 작동하는지 확인	P
	setDrinkTest	클라이언트는 /pay/setDrink에 음료 코드를 전달하면 서버는 해당 요청을 수신 후 클라이언트에 ok 응답을 반환하는지의 여부를 통해 성공 여부 확인	P
	setDrinkNumTest	클라이언트는 /pay/setDrinkNum에 요청을 보내 서버에 음료 수량을 설정함. 해당 요청의 반환값이 ok 인지에 따라 성공 여부 확인.	P

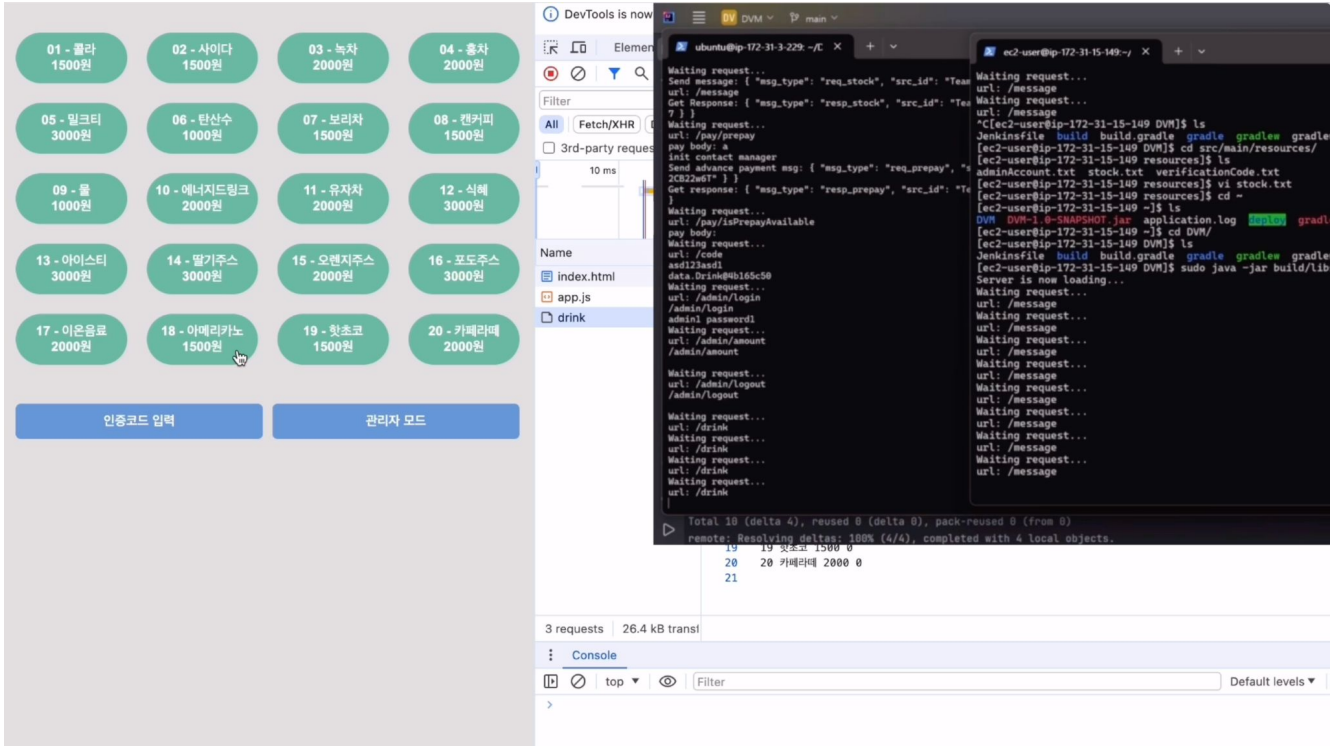
System Test

Class	Test Case	Description	Pass
DVMSimulationServer	isPayAvailableTest	클라이언트가 /pay/isPayAvailable에 요청을 보내 서버에게 결제 가능 여부를 확인. 해당 요청의 반환값이 ok인지에 따라 성공 여부 확인.	P
AdminAccountManager	checkUserTest	checkUser 메서드를 검사. 유효한 계정과 유효하지 않은 계정을 생성하여 확인	P
	logoutTest	logout 메서드를 검증. 이 메서드는 항상 true를 반환해야 하므로 이를 확인.	P
DrinkManager	hasDrinkTest	hasDrink 메서드를 검증. 특정 음료가 요청된 수량만큼 재고가 있는지 여부를 확인.	P
	getDrinkTest	getDrink 메서드 검증. 요청된 수량만큼 특정 음료를 가져오는지 확인하고, 가져온 음료 객체의 속성을 확인.	P
	getMenuInfoTest	getMenuInfo 메서드 검증. 메뉴 리스트를 가져오는지 확인하고, 빈 리스트가 아닌지 확인.	P
	getDrinkQuantityTest	getDrinkQuantity 메서드 검증. 특정 음료의 재고 수량을 확인.	P
	manageDrinkTest	manageDrink 메서드 검증. 음료의 재고 수량을 업데이트하는지 확인하고, 업데이트된 수량을 확인.	P

System Test

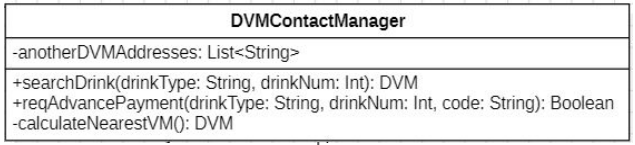
Class	Test Case	Description	Pass
DrinkManager	reqAmountOfDrinkTest	reqAmountOfDrink 메서드 검증. 요청된 음료의 재고량을 가져오는지 확인하고, 빈 리스트가 아닌지 확인.	P
PaymentManager	reqPaySuccessTest	reqPay 메서드 검증. 가짜 `Card` 객체를 만들고, 이를 사용하여 결제를 요청. 결제가 성공적으로 이루어지는지 ok 라는 응답 여부 확인.	P
VerificationManager	getVerificationCode	getVerificationCode 메서드 검증. 메서드를 호출하여 반환된 코드가 String 타입이며 공백이 아닌지 확인.	P
	saveCodeAndVerifyTest	saveCode와 verifyCode 메서드를 검증. getVerificationCode` 메서드를 사용하여 코드를 생성하고 저장한 후, 해당 코드를 검증 후 생성된 코드와 검증된 코드의 일치성을 확인하고, 음료 유형과 수량이 일치하는지 확인.	P

시스템 동작 Demo 영상

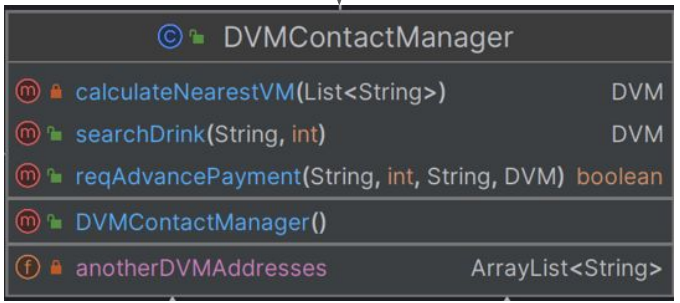


https://youtu.be/Drzq0OFn0_M

OOD(2040)에서 변경/수정된 부분 정리



생성자 추가



구현 시 예상보다 어려웠던 점

- 소켓 통신의 구현이 익숙하지 않아 시행착오를 많이 겪었다.
- CTIP 환경 구축 시 자동 배포가 이루어지지 않아 해당 오류를 찾는 데 시간이 많이 소요되었다.
- iteration을 한번만 돌기 때문에 정해둔 설계의 수정을 최소한으로 구현해야하는 것이 생각보다 까다로웠다.
- 종속성이 있는 클래스에 대한 Unit Test Case를 설정하는 고민이 까다로웠다.
- 소켓 통신에 대한 Test Code를 작성하는 것에서 많은 시행 착오가 있었다.
- 전체적인 흐름은 객체지향적 설계를 통해 팀원 모두 숙지하고 있었으나, 기존 클래스 다이어그램에서의 설계 미스로 세부적인 구현이 다소 서로 일치하지 않는 부분이 있었다.

구현 시 예상보다 쉬웠던 점

- 설계 시 클래스와 메서드를 구상해 주었기 때문에 구현 시간이 많이 줄어들었다.
- 팀원 전체가 Use Case와 각 SSD, Class Diagram을 숙지하고 있는 상태이기 때문에 각 담당하는 부분에 대한 소통과 이해가 비교적 빨랐다.
- 각 클래스가 담당하는 부분이 명확하여 역할 분담에 용이하였다.

객체지향개발방법론의 장점

- 단계별로 조금씩 발전시키는 과정에서 요구사항을 명확하게 인지할 수 있다.
- 문서작업을 진행한 뒤 구현을 수행하여 프로젝트를 효율적으로 개발할 수 있다.
- 코드에서 오류가 발생하였을 때 기존 문서를 참고하면 더욱 쉽게 디버깅할 수 있다.
- CI/CD 환경을 구축하였을 때 배포 주기와 과정을 줄일 수 있다.

객체지향개발방법론의 단점

- 문서화 작업이 오래 걸리고 수정이 필요할 때 연관된 모든 문서를 업데이트 해줘야 하는 번거로움이 있다.
- 개념과 원칙을 완전히 이해하고 활용하기 위해 시간과 노력이 요구된다.
- 클래스들이 세분화 될 수록 코드가 점점 복잡해진다.
- 시스템이 방대해진다면 다이어그램으로 한눈에 알아보기 어려울 수 있다.

소감

강상민

객체지향적 설계와 개발에 대해 많이 고민해 볼 수 있던 시간이었습니다. 생각보다 다소 귀찮은 작업들이 즐비하였지만, 돌이켜보면 전부 나름의 의미가 충분히 있던 작업들이었음을 깨달을 수 있었습니다. 실제 구현해서 많은 어려움을 겪으며 매우 힘들었지만.. 객체 지향적 설계 덕분인지 생각보다 체계적으로 디버깅할 수 있었고 테스팅 또한 의미있는 일이었습니다. 시간이 충분치 못하여 테스팅에 대해 더욱 심도 있게 고민해보지 못한 부분이 아쉬움이 조금 남습니다.

김재윤

객체지향적인 사고를 통한 프로젝트 개발 경험은 새로운 경험이었고 상당히 인상적이었습니다 . 앞으로 프로젝트를 진행할 시 선택적으로 필요한 부분만 채택한다면 더욱 체계적인 프로젝트 개발을 진행할 수 있을 것으로 기대됩니다.

소감

이찬양

객체지향개발에 익숙하진 않지만, 그 기법을 이해하고 실제 프로젝트에 적용시켜볼 수 있어서 좋았습니다. 한 단계씩 늘어갈 때 마다 앞 단계에서 진행했던 과정들의 의의를 깨닫게 되었고, design과 implement 사이의 간극을 직접 느끼며 한번 더 iteration을 돌았다면 더 나은 OOPT 결과물을 낼 수 있었겠다는 생각이 들었습니다. 추후에는 더 큰 시스템에 객체지향개발방법론을 적용시켜보고자 하는 열망이 생겼습니다.

한승현

기존엔 계획 없이 프로그래밍 하거나, 가볍게 구상하고 실행하는 정도에 머물렀지만 이번 기회를 통해 철저한 OOAD를 알게 되었다. 초반엔 코드 제작 계획이 굉장히 깐깐하다는 생각이 들었지만 시간이 지날수록 생각이 바뀌었다. 프로젝트의 규모가 클수록 더욱 더 철저한 계획과 분석이 필요하다는 점을 깨달았고, 처음 뼈대를 잘 잡아두는게 중요하다고 느꼈다. 제대로 분석하고 제대로 된 계획을 세워뒀다면 이후엔 프로그램 제작이 더 쉬웠을 것 같았다. 모든 경우의 수를 생각하고 대비해 다이어그램을 제작해야 했기 때문에 이 부분은 꽤 어려웠다.

감사합니다