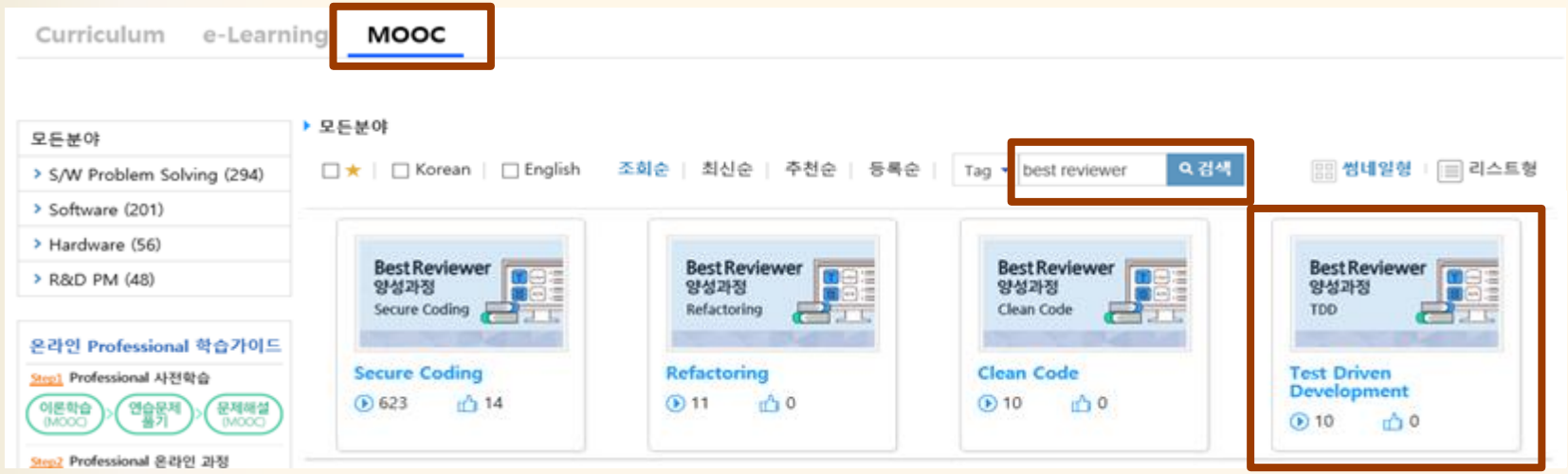
과정 진행 (TDD)

과정 시간표 - TDD

	15일 (수)	16일 (목)	17일(금)
09:00	9:00~10:00 클린코드 평가	9:00~11:30 실습 #2	09:00 ~10:00 실습 #4 안내
10:00	10:00~12:00 TDD 개론 Google Test		10:00~12:00 실습 #4
11:00		11:30~12:00 실습#2 리뷰	
12:00	점심 식사	점심 식사	점심 식사
13:00	13:00~14:30 Google Test	13:00~15:30 Google Mock	13:00~14:00 실습 #4
14:00			14:00~15:00 실습 #5 (TDD report)
15:00	14:30~17:00 실습 #1	15:30~17:30 실습#3	15:00~시험 안내 구현 평가
16:00			
17:00	17:00~17:30 실습 #1 리뷰		
	17:30 일과 마무리	17:30 일과 마무리	

TDD MOOC 영상 자료

- https://swexpertacademy.samsung.com
  -> MOOC -> Best Reviewer 검색 -> Test Driven Development 선택



- #6 xUnit Test Pattern (75분)
- #7 Test Smell (50분)
- MOOC 강의 자료 : SHARED Repo./TDD_xUnitTestPattern_TestSmell.pdf

과정 목표

- 테스트의 중요성을 확인한다.
- TDD 방법론을 이해하고 적용할 줄 안다.
- Google test framework를 사용하여 테스트 코드를 작성할 수 있는 능력을 키운다.
- 주요 xUnit Test Pattern 들을 이해한다.
- Test Smell 의 종류를 알고, Test Smell 의 원인과 해결책을 이해한다.

실습 #1 준비– Gilded Rose

- 삼성 GitHub Organization
  - Organization 이름 : Best-Reviewer-3-24
  - https://github.ecodesamsung.com/Best-Reviewer-3-24
  - Repository : TDD_GILDEDROSE

- 개인별 실습 코드는 Organization 내 Repository 생성
  - Repository : 과정명_실습명_개인번호 (ex: TDD_GILDEDROSE_01, TDD_GILDEDROSE_02,…)
  - description : Author, Reviewer 성명
    -> ex: Author : 강주인, Reviewer : 공객체

Author <-> Reviewer

- Author는 PR 요청하고, Reviewer가 PR 요청에 대해 알 수 있게 Reviewer에게 메신저로도 알려주세요.

- 실습 시간 중 최소 1회 이상 리뷰 진행해 주세요.

- Commit – Push – Pull Request – Review 관련
  - 가능한 1개의 주제 단위로 1 commit 으로 구성해 주세요.
  - Reviewer : 가능하면 질문이나 제안 위주로 comment 작성해 주세요.
  - Author : 리뷰 의견에 반대 또는 문의시 답글 작성
  - 실습이 온라인으로 진행되는 관계로 원활한 진행을 위해 reviewer의 approve 없이 진행하셔도 됩니다.
  - Merge는 하지 않으셔도 됩니다.

실습 #1 설명 – GildedRose 실습 내용

- 실습 목적 : 테스트 코드 작성을 통한 legacy code 이해

- GildedRose 의 Legacy 테스트 코드
  - GildedRoseUnitTests.cc : TEST(GildedRoseTest, Foo)
  - GildedRoseTextTests : main()

- GildedRoseUnitTests.cc : test 추가
  - updateQuality() 의 test case를 추가 작성해 보세요.
  - Unit test 내용 : 문서상에 나타난 동작들 확인

- GildedRoseTextTests.cc : non-automated golden-master test.
  => Automated test 로 변경(ApprovalTest 이용)

- unit test VS golden-master test 비교

실습 #1 설명 – GildedRose

- https://github.com/emilybache/GildedRose-Refactoring-Kata
- Gilded Rose 는 게임 "World of Warcraft"에 나오는 여관 이름입니다.
- 아이템은 <퀄리티> 속성을 가지고 있으며, 예외 사항이 있는 아이템들이 있습니다.
- 아이템은 [아이템 이름, 유통기한, 퀄리티]의 속성을 가집니다.
- 아이템의 퀄리티는 0 이상이고 아이템의 퀄리티는 하루가 지날 때마다 1씩 줄어듭니다.
- 유통 기한이 지난 아이템의 퀄리티는 2배의 속도로 떨어집니다.
- 퀄리티는 최대값이 50입니다.
- Aged Brie(숙성치즈), Backstage Pass(백스테이지입장권), Sulfuras(전설의 아이템) 의 예외적 규칙
  - Aged Brie는 하루가 지날 때마다 퀄리티가 1씩 증가합니다.
    유통기한이 지나면 퀄리티가 2씩 증가합니다.
  - Backstage Pass 는 유통기한(콘서트일)이 다가올수록 퀄리티가 증가합니다.
    유통기한이 11일 이상일 때는 1, 10일 이하일 때는 2, 5일 이하일 때는 3씩 증가하지만
    콘서트 날이 지나면 퀄리티는 0이 됩니다.
  - Sulfuras의 퀄리티는 80으로 변화가 없습니다.

실습 #1 설명 – GildedRose

- Unit Test

```
TEST(GildedRoseTest, testAgedBrieOver50) {
    vector<Item> items;
    items.push_back(Item("Aged Brie", 10, 50));
    GildedRose app(items);
    app.updateQuality();

    //EXPECT_EQ("Aged Brie", app.items[0].name);
    //EXPECT_EQ(1, app.items[0].sellIn);
    EXPECT_EQ(50, app.items[0].quality);
}
```

실습 #1 설명 – GildedRose

- Unit Test

```
TEST(GildedRoseTest, testIncreaseBackStagePassQualityWithin5days) {
    vector<Item> items;
    items.push_back(Item("Backstage passes to a TAFKAL80ETC concert", 5, 44));
    GildedRose app(items);
    app.updateQuality();

    //EXPECT_EQ(4, app.items[0].sellIn);
    EXPECT_EQ(47, app.items[0].quality);
}
```

실습 #1 결과 – Gilded Rose

실습 #2 준비 - Supermarket

- 삼성 GitHub Organization
  - Organization 이름 : Best-Reviewer-3-24
  - https://github.ecodesamsung.com/Best-Reviewer-3-24
  - Repository : TDD_SUPERMARKET

- 조별, Pair별 실습 코드는 Org 내 Repo 생성, 실습 진행
  - Repository : 과정명_실습명_개인번호 (ex: TDD_SUPERMARKET_01, TDD_SUPERMARKET_02,...)
  - description : Author, Reviewer 성명 -> ex: Author : 나소중, Reviewer : 정다운

실습 #2 준비 - Supermarket

- 실습목적 : Test as Document
- SuperMarket pos 프로그램의 일부
- Legacy 코드의 이해
  - 구현 코드들을 읽고 이해하기 vs 테스트 코드를 읽고 이해하기
- Legacy code 에 기능을 추가하기 위한 테스트 코드 작성

실습 #2 설명 - Supermarket

- Supermarket 예제의 다양한 할인 예 : 문서 상에 나타난 할인 예시
  - Buy two toothbrushes, get one free. Normal toothbrush price is €0.99
  - 20% discount on apples, normal price €1.99 per kilo.
  - 10% discount on rice, normal price €2.49 per bag
  - Five tubes of toothpaste for €7.49, normal price €1.79
  - Two boxes of cherry tomatoes for €0.99, normal price €0.69 per box.

실습 #2 설명 - Supermarket

- Test code 작성
  - 다양한 할인 방법이 적용되는 테스트 코드를 만들어 보세요.(Unit Test)
  - ReceiptPrinter 를 테스트 할 수 있는 테스트코드를 만들어 보세요.(Golden master Test)
  - 추가될 기능(bundles)에 대한 테스트를 먼저 만들어 보세요.
    - 새로운 기능(bundle)에 대한 코드를 작성하거나, 현재 코드를 리팩토링 할 필요는 없습니다.
  - 추가될 기능(HTML printer)에 대한 테스트를 먼저 만들어 보세요
    - 새로운 기능(HTML printer)에 대한 코드를 작성하거나, 현재 코드를 리팩토링 할 필요는 없습니다.

실습 #2 설명 - Supermarket

- Unit Test

```
TEST(SupermarketTest, tenPercentDiscount) {
        // Arrange
        FakeCatalog catalog;
        Product toothbrush("toothbrush", ProductUnit::Each);
        Product apples("apples", ProductUnit::Kilo);

        catalog.addProduct(toothbrush, 0.99);
        catalog.addProduct(apples, 1.99);

        Teller teller(catalog);
        teller.addSpecialOffer(SpecialOfferType::TenPercentDisco
unt, toothbrush, 10.0);

        ShoppingCart cart;
        cart.addItemQuantity(apples, 2.5);

        Receipt receipt;
        teller.checksOutArticlesFrom(cart, receipt);
```

```
        ASSERT_EQ(4.975, receipt.getTotalPrice());
        EXPECT_EQ(0, receipt.getDiscounts().size());
        ASSERT_EQ(1, receipt.getReceiptItems().size());

        ReceiptItem& receiptItem = receipt.getReceiptItems()[0];

        ASSERT_EQ(apples, receiptItem.getProduct());
        ASSERT_EQ(1.99, receiptItem.getPrice());
        ASSERT_EQ(2.5 * 1.99, receiptItem.getTotalPrice());
        ASSERT_EQ(2.5, receiptItem.getQuantity());
    }
```

실습 #2 결과 - Supermarket

실습 #3 준비 – Trip Service

- 실습목적 : Dependency 를 갖는 Legacy code에 대하여 테스트 항목 추출 및 테스트 코드 작성

- 실습내용 : 기존 코드의 dependency 를 분리하여 테스트 코드를 작성해 보세요.

- 삼성 GitHub Organization
  - Organization 이름 : Best-Reviewer-3-24
  - https://github.ecodesamsung.com/Best-Reviewer-3-24
  - Repository : TDD_TRIPSERVICE

- 조별, Pair별 실습 코드는 Org 내 Repo 생성, 실습 진행
  - Repository : 과정명_실습명_개인번호 (ex: TDD_TRIPSERVICE_01, TDD_TRIPSERVICE_02,…)
  - description : Author, Reviewer 성명 -> ex: Author : 고단수, Reviewer : 안단순

실습 #3 설명 – Trip Service

<refactoring 대상 – TripService::getTripByUser>

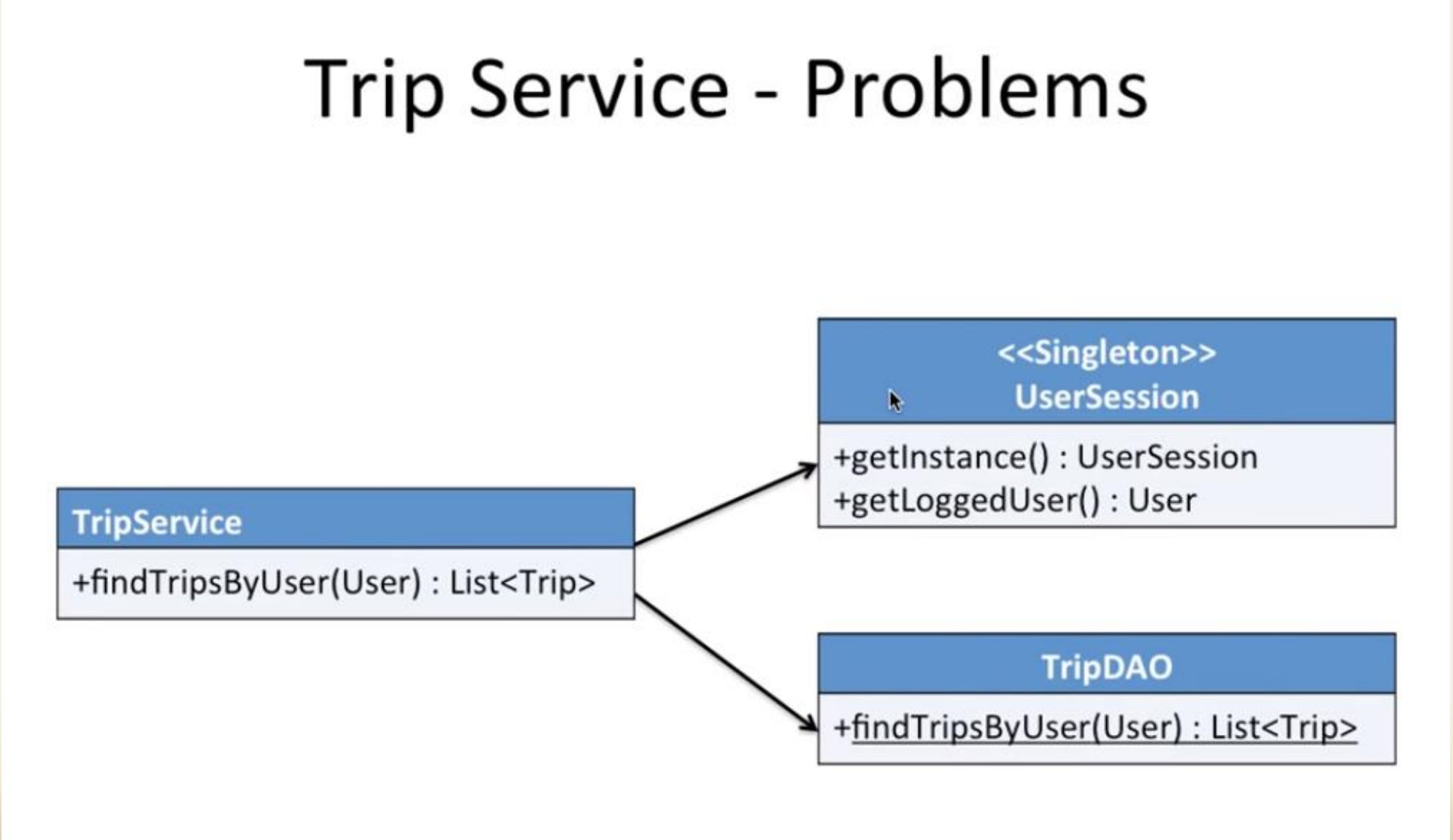
```
template<class T>
std::list<Trip> TripService<T>::GetTripsByUser(User user)
{
	std::list<Trip> triplist;
	User* loggedUser = UserSession::GetInstance()->GetLoggedUser();

	bool isFriend = false;
	if (loggedUser)
	{
		for (auto i = user.GetFriends().begin(); i != user.GetFriends().end(); ++i)
		{
			if (*i == *loggedUser)
			{
				isFriend = true;
				break;
			}
		}
		if (isFriend)
		{
			triplist = T::FindTripsByUser(user);
		}
		return triplist;
	}
	else
	{
		throw UserNotLoggedInException();
	}
}
```

큰범위에서 작은 단위로 테스트를 작성한다

- 로그인한 사용자가 없을 때 exception
- user에게 친구가 없을 때 빈 여행정보를 return
- user의 친구가 로그인한 사용자가 아닐 때 빈 여행정보를 return
- user가 로그인 한 친구가 있을 때 user의 여행정보를 return

실습 #3 설명 – Trip service 클래스 관계도



실습 #3 설명 – Trip Service

- TripDAO와 UserSession은 Mock 객체를 만들어 대체하여 사용한다.
- MockTripDAO 는 제공, MockUserSession은 생성하여 작성

- 참조 코드

```
User user(1);
Auto mockUserSession = new MockUserSession();
UserSessionAccessor::Set(mockUserSession);

// 테스트코드
EXPECT_CALL(*mockUserSession, GetLoggedUser())
        .WillRepeatedly(Return(&user));


UserSessionAccessor::DeleteSession();
delete mockUserSession;
```

실습 #3 결과 – Trip Service

TDD – String Calculator

- 문자열을 계산하는 계산기를 만들어 보자

Requirement	예시
empty string it will return 0	"" = 0
Single number string it will return number itself	"1" = 1 , "3" = 3
Comma "," is delimiters, Act as +	"1,2" = 3, "2,3" = 5, "1,2,3" = 6
New line "₩n" is new delimiters	"1₩n2" = 3, "2₩n3" = 5, "1₩n2₩n3" = 6
Mixed delimiter (, and ₩n)	"1₩n2,3" = 6, "1,2₩n3" = 6
Support custom delimiters	"//;₩n1;2" == 3 ";" will be new delimiter
Negative number : "negatives not allowed" exception	"-1" -> exception "negatives not allowed"
multiple negatives : show all after exception	"-1,-2,3" -> exception "negatives not allowed : -1,-2
Numbers bigger than 1000 should be ignored	"2,1000" = 2
Long custom delimiter support	"//[***]₩n2***3" = 5
Allow multiple delimiters	"//[*][%]₩n1*2%3" == 6
Longer multiple delimiter	"//[**][%%]₩n1**2%%3" == 6.

TDD – String Calculator : 생성 테스트

- 실패하는 테스트케이스 만들기

```
TEST(StringCalculator, frameworkCheckTest)
{
    FAIL();
}
```

- 테스트케이스 성공하게 수정

```
TEST(StringCalculator, frameworkCheckTest)
{
    //FAIL();
    StringCalculator *sc;
    ASSERT_NE(sc, nullptr);
}
```

```
<StringCalculator.h>

#ifndef __STRINGCALCULAOTR_H__
class StringCalculator
{

};
#endif
```

TDD – String Calculator : empty string

1. 실패하는 테스트케이스 작성

```
TEST(StringCalculator, returnZeroOnEmptyString)
{
    StringCalculator sc;
    ASSERT_EQ(0, sc.add(""));
}
```

```
class StringCalculator
{
public:
    int add(string inputString) {
        return -1;
    };
};
```

2. 구현 코드 작성 : 클래스와 메소드 구현

- 실패하는 테스트 코드 check
- 테스트 코드 통과하게 구현 코드 수정
- 테스트 코드 통과 여부 확인

```
class StringCalculator
{
public:
    int add(string inputString) {
        return 0;
    };
};
```

TDD – String Calculator : single number string

1. “Single number string it will return number itself” 테스트 코드 작성

```
TEST(StringCalculator, returnNumOnSingleNum)
{
    StringCalculator sc;
    ASSERT_EQ(1, sc.add("1"));
    ASSERT_EQ(2, sc.add("2"));
}
```

2. 테스트 통과하는 코드 구현

```
int StringCalculator::add(string inputString) {
    if (inputString.empty())
        return 0;

    return stoi(inputString);
};
```

3. 테스트 통과 여부 확인

TDD – String Calculator : comma is delimiter, acts as +

1. “1,2” 테스트 코드 작성

```
TEST(StringCalculator, returnSumTwoNumCommaDelimiter)
{
    StringCalculator sc;
    ASSERT_EQ(3, sc.add("1,2"));
}
```

2. 구현 코드 작성하고 테스트 통과 여부 확인

```
int StringCalculator::add(string inputString) {
    if(inputString.empty())
        return 0;

    string delimiter = ",";
    vector<string> numbers = split(inputString, deli
miter);
    if(numbers.size() > 1)
        return stoi(numbers[0]) + stoi(numbers[1]);
    return stoi(numbers[0]);

}
```

TDD – String Calculator : comma is delimiter, acts as +

1. “1,2,3” 테스트 코드 작성

```
TEST(StringCalculator, returnSumThreeNumCommaDelimit
er)
{
    StringCalculator sc;
    ASSERT_EQ(6, sc.add("1,2,3"));
}
```

2. 구현 코드 작성하고 테스트 통과 여부 확인

```
int StringCalculator::add(string inputString) {
    if(inputString.empty())
        return 0;

    string delimiter = ",";
    vector<string> numbers = split(inputString, deli
miter);
    int value = 0;
    for(int i=0; i<numbers.size(); i++)
        value+=stoi(numbers[i]);

    return value;
}
```

TDD – String Calculator : newline is new delimiters

1. newline 에 대해 실패하는 테스트 코드 작성

```
TEST(StringCalculator, returnSumTwoNumNewLineDelimiter)
{
    StringCalculator sc;
    ASSERT_EQ(3, sc.add("1₩n2"));
}
```

2. 구현 코드 작성하고 테스트 통과 여부 확인

```
int StringCalculator::add(string inputString) {
    if(inputString.empty())
        return 0;

    string delimiter = ",|₩n";
    vector<string> numbers = split(inputString, delim
iter);
    int value = 0;
    for(int i=0; i<numbers.size(); i++)
        value+=stoi(numbers[i]);

    return value;

}
```

TDD – String Calculator : Mixed delimiters (, and ₩n)

1. delimiter 가 혼용되는 경우 테스트 코드 작성

```
TEST(StringCalculator, returnSumMiltiNumCommaDelemiter)
{
    StringCalculator sc;
    ASSERT_EQ(6, sc.add("1₩n2,3"));
    ASSERT_EQ(6, sc.add("1,2₩n3"));
}
```

2. 테스트 통과 확인

TDD – String Calculator : Custom delimiters (//;)

1. Custom delimiter 테스트 코드 작성

```
TEST(StringCalculator, returnSumMultiPrivateDelimiter)
{
    StringCalculator sc;
    ASSERT_EQ(3, sc.add("//;₩n1;2"));
}
```

2. 구현 코드 작성하고 테스트 통과 여부 확인

```
int StringCalculator::add(string inputString) {
    if(inputString.empty())
        return 0;

    string delimiter = ",|₩n";
    if (inputString.substr(0, 2) == "//") {
        delimiter = delimiter + "|" + inputString.substr(
2,1);
        inputString.substr(3, inputString.size()-3);
    }

    vector<string> rawNumbers = split(inputString, d
elimiter);
    vector<int> numbers = findValidNumbers(rawNu
mbers);
    int sum = 0;
    for (int i = 0, n = numbers.size(); i < n; i++)
    {
        sum += numbers[i];
    }
    return sum;
}
```

TDD – String Calculator : Negative number

1. 음수를 확인하는 테스트 코드 작성

```
TEST(StringCalculator, exceptionForMinusValue)
{
    StringCalculator sc;
    ASSERT_THROW( sc.add("-1"), string);
}
```

2. 구현 코드 작성하고 테스트 통과 여부 확인

```
vector<int> StringCalculator::findValidNumbers(vector<string>
 numbers)
{
    string negativeExceptionStr;
    vector<int> validNumbers;

    int i = 0;
    for (string current : numbers)
    {
        int eachNum = atoi(current.c_str());
        if (eachNum < 0)
            negativeExceptionStr = negativeExceptionStr + curren
t + ",";
        validNumbers.push_back(eachNum);
    }
    if(!negativeExceptionStr.empty()) {
        negativeExceptionStr = negativeExceptionStr.substr(0, ne
gativeExceptionStr.length() - 1);
        throw negativeExceptionStr;
    }
    return validNumbers;
}
```

TDD – String Calculator : Multiple negatives

1. Multiple negatives 테스트 코드 작성

```
TEST(StringCalculator, exceptionForMultiMinusValue)
{
    StringCalculator sc;
    try {
        sc.add("-1,2,-12,-3");
    }
    catch (string ex) {
        ASSERT_EQ("-1,-12,-3", ex);
    }
}
```

2. 구현 코드 작성하고 테스트 통과여부 확인

TDD – String Calculator : Ignore numbers bigger than 1000

1. 1000 보다 큰 수의 경우 테스트 코드 작성

```
TEST(StringCalculator, ignoreOver1000Value)
{
    StringCalculator sc;
    ASSERT_EQ(4, sc.add("2₩n2,1001"));
}
```

2. 구현 코드 작성하고 테스트 확인

```
    for (string current : numbers)
    {
        int eachNum = atoi(current.c_str());
        if (eachNum < 0)
            negativeExceptionStr = negativeExceptionStr + current
+ ",";

        if (eachNum < 1000)
            validNumbers.push_back(eachNum);
    }
```

TDD – String Calculator : Long custom delimiter([***])

1. Long custom delimiter 테스트 코드 작성

```
TEST(StringCalculator, returnSumlongDelimiter
)
{
    StringCalculator sc;
    ASSERT_EQ(5, sc.add("//***₩n2***3"));
}
```

2. 구현 코드 작성하고 테스트 확인

```
    string delimiter = ",|₩n";
    if (inputString.substr(0, 2) == "//") {
        vector<string> privateDelimeter = split(inputString.substr(2), "//|₩n");
        delimiter = delimiter + "|" + privateDelimeter[0];
        inputString = privateDelimeter[1];
    }
```

TDD – String Calculator : Multiple delimiter(//[*][%])

1. Multiple delimiter 테스트 코드 작성

```
TEST(StringCalculator, sumMultiPrivateDelimiter)
{
    StringCalculator sc;
    ASSERT_EQ(6, sc.add("//[*][%]₩n1*2%3"));
}
```

2. 구현 코드 작성하고 테스트 확인

```
string StringCalculator::findDelimiters(string inputString) {
    string delimiter = ",|₩n";
    if (inputString.substr(0, 2) == "//") {
        vector<string> privateDelimeter = split(inputString.substr(2),
 "//|₩n");

        replaceAll(privateDelimeter[0], "][", "|");
        replaceAll(privateDelimeter[0], "[", "");
        replaceAll(privateDelimeter[0], "]", "");
        replaceAll(privateDelimeter[0], "*", "₩₩*");
        replaceAll(privateDelimeter[0], "%", "₩₩%");

        delimiter = delimiter + "|" + privateDelimeter[0];
        inputString = privateDelimeter[1];
    }
    return delimiter;
}

string StringCalculator::replaceAll(string &str, string source, string t
arget)
{}
```

TDD – String Calculator : Longer multiple delimiter

1. Longer multiple delimiter 테스트 코드 작성

```
TEST(StringCalculator, sumMultiPrivatelongDelimiter)
{
    StringCalculator sc;
    ASSERT_EQ(15, sc.add("//[**][%%]₩n3**5%%7"));
}
```

2. 구현 코드 작성하고 테스트 확인

TDD – String Calculator : 테스트 코드 refactoring

- 테스트 코드 Refactoring

```
//Parameterized TEST
class StringCalculatorParametersTests : public ::testing::TestWithParam<std::tuple<int, std::string>> {
protected:
    StringCalculator strCalculator;
};

TEST_P(StringCalculatorParametersTests, StringResultTest)
{
    int expected = std::get<0>(GetParam());
    string inputStr = std::get<1>(GetParam());

    ASSERT_EQ(expected, strCalculator.add(inputStr));
}

INSTANTIATE_TEST_CASE_P(
    StringCalculatorTests,
    StringCalculatorParametersTests,
    ::testing::Values(
        std::make_tuple(6, "1,2,3"),
        std::make_tuple(5, "//***₩n2***3"),
        std::make_tuple(6, "//[*][%]₩n1*2%3"),
        std::make_tuple(15, "//[**][%%]₩n3**5%%7")
    )
);
```

실습 #4 준비 – RPN Calculator

- 실습목적 : TDD 방법론 실습
- 실습내용 : 요구사항에 대한 테스트 코드를 먼저 작성하고 구현하는 개발방식 실습.
- 삼성 GitHub Organization
  - Organization 이름 : Best-Reviewer-3-24
  - https://github.ecodesamsung.com/Best-Reviewer-3-24
  - Repository : TDD_RPNCALCULATOR
- 조별, Pair별 실습 코드는 Org 내 Repo 생성, 실습 진행
  - Repository : 과정명_실습명_개인번호 (ex: TDD_RPNCALCULATOR_01, TDD_RPNCALCULATOR_02,…)
  - description : Author, Reviewer 성명 -> ex: Author : 한아름, Reviewer : 최신의

실습 #4 설명 – RPN Calculator

- RPN Calculator 를 TDD practice 로 작성해 보세요.
- 후위 연산 계산기

"1 2 +" => 1 + 2 = **3**

"1 2 –" => 1 - 2 = **-1**

"2 3 *" => 2 * 3 = **6**

"20 5 /" => 20 / 5 = **4**

"4 2 + 3 *" => (4 + 2) * 3 = **18**

"3 5 8 * 7 + *" => ((5 * 8) + 7) * 3 = **141**

"9 SQRT" => √9 = **3** Math.sqrt(9) 사용

"4 5 MAX 1 2 MAX" => MAX(MAX(4, 5), 1, 2) = **5**

"5 3 9 2 4 1 MAX" => MAX(5, 3, 9, 2, 4, 1) = **9**

Divided by Zero Exception

모든 수는 정수이며,

나누기: 결과가 정수가 나오는 입력으로

만드시면 됩니다.

실습 #5 TDD 는 반드시 필요한가?

- Legacy code에 Test 를 반드시 만들어야 하는가?
- Code Review 와 Test 는 어떻게 연관되어야 하는가?

- TDD 는 반드시 필요한가?
  - Is TDD DEAD?
    - https://martinfowler.com/articles/is-tdd-dead/
    - https://junho85.pe.kr/975
  - TDD is the best thing that has happened to software design
    - https://www.thoughtworks.com/insights/blog/test-driven-development-best-thing-has-happened-software-design

- 위 주제에 관한 의견을 서술한 간단한 보고서를 작성해 주세요.

THANK YOU.