

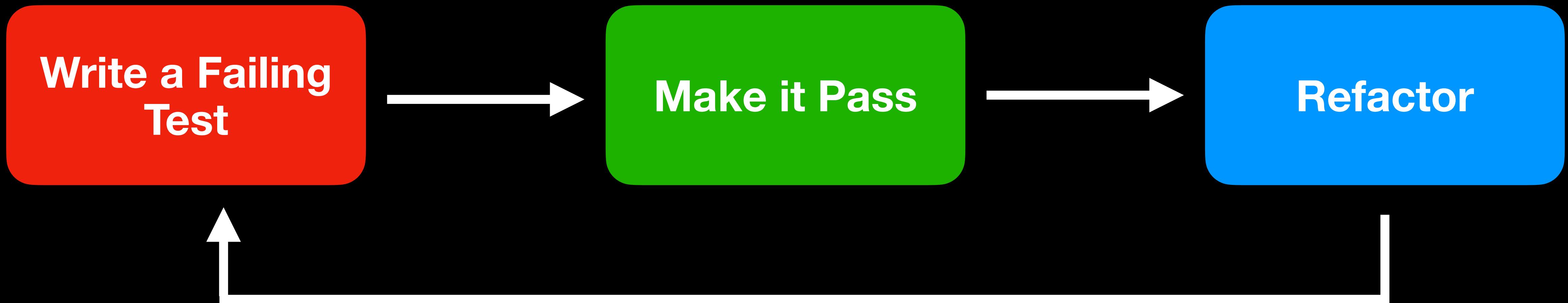
Test-Driven Development

Chapter2 : The TDD Cycle

조장희

The TDD Cycle

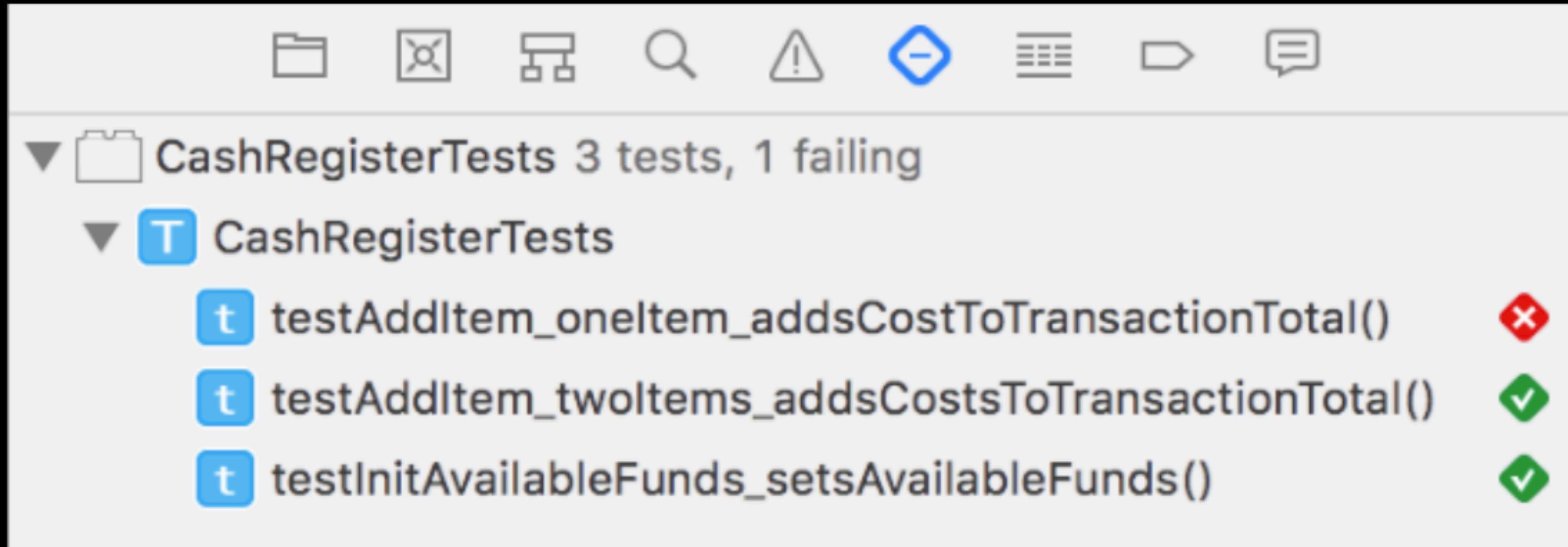
TDD Cycle



- Red : 코드를 작성하기 전 실패한 테스트를 작성
- Green : 테스트를 통화 하기 위한 최소 코드 작성
- Refactor : 앱과 테스트 코드를 정리

The TDD Cycle

TDD Cycle



- 실패한 테스트는 Red(X)
- 통과한 테스트는 Green(v)

The TDD Cycle

Getting Started

플레이 그라운드로 진행

테스트 식이나 테스트 대상을 설정하는 방법은 다음 챕터

여기서는 TDD Cycle에만 집중

The TDD Cycle

요구사항

- ChshRegister를 만들려고 한다.
- 사용 가능한 자금(availableFunds)을 승인하는 이니셜 라이저 작성
- 매매(transaction)를 위해 금액을 추가(addItem)가 되어야 함
- 금액 추가가 여러명이 되어야 함
- 결재를 승인 하는 기능이 있어야 함(acceptPayment mehtod)

The TDD Cycle

Red: 실패 테스트 케이스 작성

- ChshRegister를 만들려고 한다.

```
import Foundation
import XCTest

class CashRegisterTests: XCTestCase {
}

}
```

The TDD Cycle

Red: 실패 테스트 케이스 작성

```
import Foundation
import XCTest

class CashRegisterTests: XCTestCase {
}

CashRegisterTests.defaultTestSuite.run()
```

The TDD Cycle

Red: 실패 테스트 케이스 작성

```
import Foundation
import XCTest

class CashRegisterTests: XCTestCase {
    ...
    ...
    func testInitCreatesCashRegister() {
        XCTAssertNotNil(CashRegister())
    }
}

CashRegisterTests.defaultTestSuite.run()
```

! Use of unresolved identifier 'CashRegister'

The TDD Cycle

테스트 이름 규칙

앞으로는 이 규칙에 따라 테스트 이름을 지정합니다.

The TDD Cycle

테스트 이름 규칙

```
class MyProjectTests: XCTestCase {  
    func testSum_TwoNumbers_ReturnsSum() {
```

- 모든 테스트 Method는 test로 시작합니다.
- 테스트 할 동작에 대한 이름이 test 뒤로 따라 갑니다.
- 선택적으로 특별한 설정이 필요한 경우 밑줄로 구분하여 추가 합니다.
- 마지막으로 예상 결과가 이어 집니다.
- test테스트동작_특별한설정_예상결과

The TDD Cycle

Red: 실패 테스트 케이스 작성

```
import Foundation
import XCTest

class CashRegisterTests: XCTestCase {
    ...
    func testInitCreatesCashRegister() {
        XCTAssertNotNil(CashRegister())
    }
}

CashRegisterTests.defaultTestSuite.run()
```

컴파일 실패도 테스트 실패 케이스

⚠ Use of unresolved identifier 'CashRegister'

The TDD Cycle

Green: 테스트 통과 케이스 작성

```
import Foundation
import XCTest

class CashRegisterTests: XCTestCase {
    ...
    func testInitCreatesCashRegister() {
        XCTAssertNotNil(CashRegister())
    }
}

CashRegisterTests.defaultTestSuite.run()

class CashRegister {
    ...
}
```

The TDD Cycle

Green: 테스트 통과 케이스 작성

```
Test Suite 'CashRegisterTests' started at 2020-07-01 20:26:59.628
Test Case '-[__lldb_expr_4.CashRegisterTests testInit_createsCashRegister]' started.
Test Case '-[__lldb_expr_4.CashRegisterTests testInit_createsCashRegister]' passed (0.213 seconds).
Test Suite 'CashRegisterTests' passed at 2020-07-01 20:26:59.843.
Executed 1 test, with 0 failures (0 unexpected) in 0.213 (0.215) seconds
```

The TDD Cycle

Refactor: 클린 코드 작성

이제 리팩토링을 해야 하는데 어떻게 해야 할까?

The TDD Cycle

Refactor: 클린 코드 작성

- 중복 로직 : 중복을 제거 하기 위해 properties, methods, class를 제거
- 주석 : 주석은 어떤 일이 왜 일어 났는지를 설명하는 도구, 코드가 어떻게 이루어져 있는지 를 설명해서는 안됨 -> 프로퍼티 및 메소드의 이름을 변경하여 보다 명확하게 하거나 단순히 코드를 더 잘 구조화 하여 전달하는 방법이 더 좋습니다.
- 냄새나는 코드 : 많은 가정을 하고 있거나 하드 코딩 된 문자열을 사용하거나 다른 이슈가 있을 수 있는 부분들이 단순히 잘 못된 것처럼 보인다면 해당 코드를 정리 할려고 노력하는 게 좋습니다. 요령은 주석과 동일을 합니다. method와 class를 꺼내고, 이름을 바꾸고, 코드를 재 구성합니다.

깨끗한 코드를 위한 5가지 팁

The TDD Cycle

Refactor: 클린 코드 작성

```
import Foundation
import XCTest

class CashRegisterTests: XCTestCase {
    ...
    func testInitCreatesCashRegister() {
        XCTAssertNotNil(CashRegister())
    }
}

CashRegisterTests.defaultTestSuite.run()

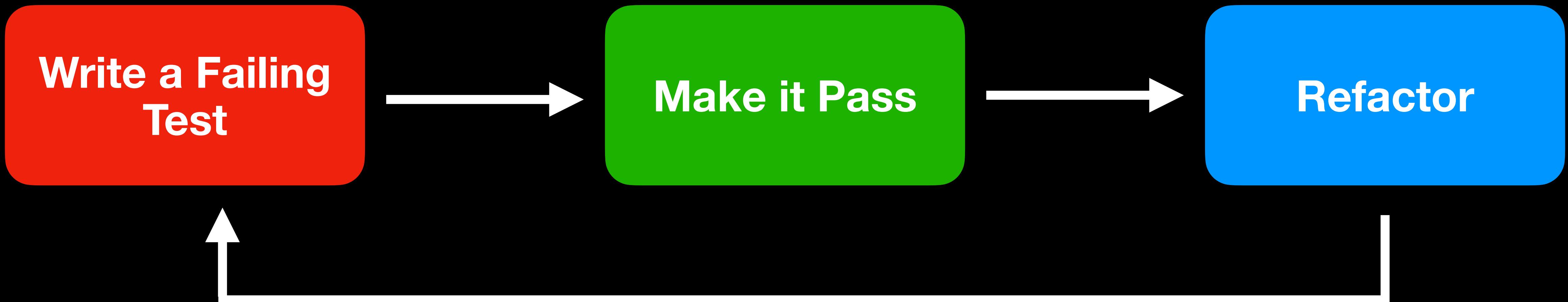
class CashRegister {
    ...
}
```

추가할 로직도 없고, 리팩토링 할 내용도 없기 때문에 끝!

The TDD Cycle

반복

- 이제 부터는 Red -> Green -> Refactor 을 반복해서 진행



The TDD Cycle

요구사항

- ~~ChshRegister를 만들려고 한다.~~
- 사용 가능한 자금(availableFunds)을 승인하는 이니셜 라이저 작성
- 매매(transaction)를 위해 금액을 추가(addItem)가 되어야 함
- 금액 추가가 여러명이 되어야 함
- 결재를 승인 하는 기능이 있어야 함(acceptPayment mehtod)

The TDD Cycle

반복 - Red: 실패 테스트 케이스 작성

- 사용 가능한 자금(availableFunds)을 승인하는 이니셜 라이저 작성

```
class CashRegisterTests: XCTestCase {  
    ...  
    func testInit_createsCashRegister() {  
        XCTAssertNotNil(CashRegister())  
    }  
    ...  
    func testInitAvailableFunds_setsAvailableFunds() {  
        // given  
        let availableFunds = Decimal(100)  
        // when  
        let sut = CashRegister(availableFunds: availableFunds)  
        // then  
        XCTAssertEqual(sut.availableFunds, availableFunds)  
    }  
}
```

이니셜 라이저가 정의 되어 있지 않아 테스트 실패

⚠ Argument passed to call that takes no arguments

The TDD Cycle

조금 더 복잡한 테스트 작성 요령

```
class CashRegisterTests: XCTestCase {  
    ...  
    func testInitCreatesCashRegister() {  
        XCTAssertNotNil(CashRegister())  
    }  
    ...  
    func testInitAvailableFunds_setsAvailableFunds() {  
        // given  
        let availableFunds = Decimal(100)  
        // when  
        let sut = CashRegister(availableFunds: availableFunds) // Argument passed to call that takes no arguments  
        // then  
        XCTAssertEqual(sut.availableFunds, availableFunds)  
    }  
}
```

- given, when, then 으로 나눠서 생각하면 유용합니다.
- given : 어떤 조건을 줄건지 - 100을 조건으로 주고
- when : 언제 어떤 행동이 일어 나야 하는지 - availableFunds를 통해 주면
- then : when의 결과는 무엇인지 - sut.availableFunds의 값과 조건이 같다고 예상

The TDD Cycle

반복 - Green: 테스트 통과 케이스 작성

```
class CashRegister {  
    ... var availableFunds: Decimal  
  
    ...  
    ... init(availableFunds: Decimal = 0) {  
        ... self.availableFunds = availableFunds  
        ...  
    }  
}
```

```
Test Suite 'CashRegisterTests' started at 2020-07-01 20:53:06.697  
Test Case '-[__lldb_expr_7.CashRegisterTests testInitCreatesCashRegister]' started.  
Test Case '-[__lldb_expr_7.CashRegisterTests testInitCreatesCashRegister]' passed (0.106 seconds).  
Test Case '-[__lldb_expr_7.CashRegisterTests testInitAvailableFunds_setsAvailableFunds]' started.  
Test Case '-[__lldb_expr_7.CashRegisterTests testInitAvailableFunds_setsAvailableFunds]' passed (0.005 seconds).  
Test Suite 'CashRegisterTests' passed at 2020-07-01 20:53:06.810.  
Executed 2 tests, with 0 failures (0 unexpected) in 0.112 (0.113) seconds
```

The TDD Cycle

반복 - Refactor: 클린 코드 작성

```
import Foundation
import XCTest

class CashRegisterTests: XCTestCase {
    ...
    ...
    func testInitCreatesCashRegister() {
        XCTAssertNotNil(CashRegister())
    }

    ...
    ...
    func testInitAvailableFunds_setsAvailableFunds() {
        // given
        let availableFunds = Decimal(100)
        // when
        let sut = CashRegister(availableFunds: availableFunds)
        // then
        XCTAssertEqual(sut.availableFunds, availableFunds)
    }
}

CashRegisterTests.defaultTestSuite.run()

class CashRegister {
    var availableFunds: Decimal

    init(availableFunds: Decimal = 0) {
        self.availableFunds = availableFunds
    }
}
```

The TDD Cycle

반복 - Refactor: 클린 코드 작성

- `testInit_createCashRegister`는 이제 사용되지 않습니다.
- 이 테스트는 아무것도 없는 상태인 것처럼 보이지만 실제로는 0을 사용해서 호출하는 것입니다. (명명된 테스트와 동작이 다름)
- 사용하지 않은 `testInit_createCashRegister`를 제거 합니다.

```
import Foundation
import XCTest

class CashRegisterTests: XCTestCase {
    func testInitCreatesCashRegister() {
        XCTAssertNotNil(CashRegister())
    }

    func testInitAvailableFunds_setsAvailableFunds() {
        // given
        let availableFunds = Decimal(100)
        // when
        let sut = CashRegister(availableFunds: availableFunds)
        // then
        XCTAssertEqual(sut.availableFunds, availableFunds)
    }
}

CashRegisterTests.defaultTestSuite.run()

class CashRegister {
    var availableFunds: Decimal

    init(availableFunds: Decimal = 0) {
        self.availableFunds = availableFunds
    }
}
```

The TDD Cycle

반복 - Refactor: 클린 코드 작성

- 코드에서 `init(availableFunds: Decimal = 0)` 을 봅니다.
- 기본 파라미터 값이 0인것이 맞을까요?
- 이건 설계상 결정을 하면됩니다.
- 기본 매개변수를 추가하면 `testInit_setDefaultAvailableFunds`에 대한 테스트를 추가 합니다.
- 기본 매개변수를 제거 할 수도 있습니다.
- 여기서는 제거 하겠습니다.

```
import Foundation
import XCTest

class CashRegisterTests: XCTestCase {
    ...
    func testInitAvailableFunds_setsAvailableFunds() {
        // given
        let availableFunds = Decimal(100)
        // when
        let sut = CashRegister(availableFunds: availableFunds)
        // then
        XCTAssertEqual(sut.availableFunds, availableFunds)
    }
}

CashRegisterTests.defaultTestSuite.run()

class CashRegister {
    var availableFunds: Decimal
    ...
    init(availableFunds: Decimal = 0) {
        self.availableFunds = availableFunds
    }
}
```

The TDD Cycle

반복 - Refactor: 클린 코드 작성

- 리팩토링을 해도 테스트는 잘 통과 합니다.
`init(availableFunds:)`를 변경 했지만 기존 기능을 손상 시키지 않았다는 안정감을 제공 합니다.
- 리팩토링에 자신감을 더 하는 것이 TDD의 주요 이점 이라 볼 수 있습니다.

```
import Foundation
import XCTest

class CashRegisterTests: XCTestCase {
    ...

    func testInitAvailableFunds_setsAvailableFunds() {
        // given
        let availableFunds = Decimal(100)
        // when
        let sut = CashRegister(availableFunds: availableFunds)
        // then
        XCTAssertEqual(sut.availableFunds, availableFunds)
    }
}

CashRegisterTests.defaultTestSuite.run()

class CashRegister {
    var availableFunds: Decimal

    init(availableFunds: Decimal) {
        self.availableFunds = availableFunds
    }
}
```

```
Test Suite 'CashRegisterTests' started at 2020-07-01 22:40:06.984
Test Case '-[__lldb_expr_9.CashRegisterTests testInitAvailableFunds_setsAvailableFunds]' started.
Test Case '-[__lldb_expr_9.CashRegisterTests testInitAvailableFunds_setsAvailableFunds]' passed (0.098 seconds).
Test Suite 'CashRegisterTests' passed at 2020-07-01 22:40:07.083.
Executed 1 test, with 0 failures (0 unexpected) in 0.098 (0.099) seconds
```

The TDD Cycle

요구사항

- ~~ChshRegister를 만들려고 한다.~~
- 사용 가능한 자금(availableFunds)을 승인하는 이니셜 라이저 작성
- 매매(transaction)를 위해 금액을 추가(addItem)가 되어야 함
- 금액 추가가 여러명이 되어야 함
- 결재를 승인 하는 기능이 있어야 함(acceptPayment method)

The TDD Cycle

반복 - Red: 실패 테스트 케이스 작성

- 매매(transaction)를 위해 금액을 추가(addItem)가 되어야 합니다.

```
func testAddItem_oneItem() {
    // given
    let availableFunds = Decimal(100)
    let sut = CashRegister(availableFunds: availableFunds)
    let itemCost = Decimal(42)
    // when
    sut.addItem(itemCost)
    // then
    XCTAssertEqual(sut.transactionTotal, itemCost)
}
```

addItem이 정의 되어 있지 않아 테스트 실패

⚠ Value of type 'CashRegister' has no member 'addItem'

⚠ Value of type 'CashRegister' has no member 'transactionTotal'

The TDD Cycle

반복 - Green: 테스트 통과 케이스 작성

```
func testAddItem_oneItem_addsCostToTransactionTotal() {  
    // given  
    let availableFunds = Decimal(100)  
    let sut = CashRegister(availableFunds: availableFunds)  
    let itemCost = Decimal(42)  
    // when  
    sut.addItem(itemCost)  
    // then  
    XCTAssertEqual(sut.transactionTotal, itemCost)  
}
```

```
class CashRegister {  
    var availableFunds: Decimal  
  
    init(availableFunds: Decimal) {  
        self.availableFunds = availableFunds  
    }  
}
```

```
class CashRegister {  
    var availableFunds: Decimal  
    var transactionTotal: Decimal = 0  
  
    init(availableFunds: Decimal) {  
        self.availableFunds = availableFunds  
    }  
  
    func addItem(_ cost: Decimal) {  
        transactionTotal += cost  
    }  
}
```

The TDD Cycle

반복 - Green: 테스트 통과 케이스 작성

```
class CashRegisterTests: XCTestCase {  
    ...  
    func testInitAvailableFunds_setsAvailableFunds() {  
        // given  
        let availableFunds = Decimal(100)  
        // when  
        let sut = CashRegister(availableFunds: availableFunds)  
        // then  
        XCTAssertEqual(sut.availableFunds, availableFunds)  
    }  
  
    func testAddItem_oneItem_addsCostToTransactionTotal() {  
        // given  
        let availableFunds = Decimal(100)  
        let sut = CashRegister(availableFunds: availableFunds)  
        let itemCost = Decimal(42)  
        // when  
        sut.addItem(itemCost)  
        // then  
        XCTAssertEqual(sut.transactionTotal, itemCost)  
    }  
  
    CashRegisterTests.defaultTestSuite.run()  
  
    class CashRegister {  
        var availableFunds: Decimal  
        var transactionTotal: Decimal = 0  
  
        init(availableFunds: Decimal) {  
            self.availableFunds = availableFunds  
        }  
  
        func addItem(_ cost: Decimal) {  
            transactionTotal += cost  
        }  
    }  
}
```

리팩토링 할게 있을까?

The TDD Cycle

반복 - Refactor: 클린 코드 작성

```
func testInitAvailableFunds_setsAvailableFunds() {  
    // given  
    let availableFunds = Decimal(100)  
    // when  
    let sut = CashRegister(availableFunds: availableFunds)  
    // then  
    XCTAssertEqual(sut.availableFunds, availableFunds)  
}  
  
func testAddItem_oneItem_addsCostToTransactionTotal() {  
    // given  
    let availableFunds = Decimal(100)  
    let sut = CashRegister(availableFunds: availableFunds)  
    let itemCost = Decimal(42)  
    // when  
    sut.addItem(itemCost)  
    // then  
    XCTAssertEqual(sut.transactionTotal, itemCost)  
}
```

The TDD Cycle

반복 - Refactor: 클린 코드 작성

- 테스트 코드에 중복적으로 사용되는 프로퍼티를 정의 합니다.

```
class CashRegisterTests: XCTestCase {  
    var availableFunds: Decimal!  
    var sut: CashRegister!  
  
    override func setUp() {  
        super.setUp()  
        availableFunds = 100  
        sut = CashRegister(availableFunds: availableFunds)  
    }  
  
    override func tearDown() {  
        super.tearDown()  
        availableFunds = nil  
        sut = nil  
    }  
}
```

The TDD Cycle

반복 - Refactor: 클린 코드 작성

- `setup()`은 각 테스트 method가 실행 되기 직전에 호출됩니다.
- 여기서 선언해 놓은 프로퍼티에 값을 대입하거나 인스턴스를 넣습니다.

```
class CashRegisterTests: XCTestCase {  
    ...  
    var availableFunds: Decimal!  
    var sut: CashRegister!  
    ...  
    override func setUp() {  
        super.setUp()  
        availableFunds = 100  
        sut = CashRegister(availableFunds: availableFunds)  
    }  
    ...  
    override func tearDown() {  
        super.tearDown()  
        availableFunds = nil  
        sut = nil  
    }  
}
```

The TDD Cycle

반복 - Refactor: 클린 코드 작성

- `tearDown()`은 각 테스트 method가 종료 된 직후 호출 됩니다.
- 다음 테스트를 위해서 프로퍼티를 초기화 해줘야 합니다.
- XCTest 프레임워크는 작동 할 때 테스트 대상 내에서 XCTestCase의 하위 클래스를 인스턴스화 하고 모든 테스트 사례가 실행 될때까지 해제하지 않기 때문에 테스트가 많은 경우 과하게 메모리를 사용하여 성능 문제가 발생 할 수 있습니다. 그래서 의도적으로 `Nil`로 설정합니다.

```
class CashRegisterTests: XCTestCase {  
    ...  
    var availableFunds: Decimal!  
    var sut: CashRegister!  
  
    override func setUp() {  
        super.setUp()  
        availableFunds = 100  
        sut = CashRegister(availableFunds: availableFunds)  
    }  
  
    override func tearDown() {  
        super.tearDown()  
        availableFunds = nil  
        sut = nil  
    }  
}
```

The TDD Cycle

반복 - Refactor: 클린 코드 작성

```
func testInitAvailableFunds_setsAvailableFunds() {  
    // given  
    let availableFunds = Decimal(100)  
    // when  
    let sut = CashRegister(availableFunds: availableFunds)  
    // then  
    XCTAssertEqual(sut.availableFunds, availableFunds)  
}
```

```
func testInitAvailableFunds_setsAvailableFunds() {  
    // given  
    // when  
    // then  
    XCTAssertEqual(sut.availableFunds, availableFunds)  
}
```

The TDD Cycle

반복 - Refactor: 클린 코드 작성

```
func testAddItem_oneItem_addsCostToTransactionTotal() {  
    // given  
    let availableFunds = Decimal(100)  
    let sut = CashRegister(availableFunds: availableFunds)  
    let itemCost = Decimal(42)  
    // when  
    sut.addItem(itemCost)  
    // then  
    XCTAssertEqual(sut.transactionTotal, itemCost)  
}
```

```
func testAddItem_oneItem_addsCostToTransactionTotal() {  
    // given  
    let itemCost = Decimal(42)  
    // when  
    sut.addItem(itemCost)  
    // then  
    XCTAssertEqual(sut.transactionTotal, itemCost)  
}
```

The TDD Cycle

요구사항

- ~~ChshRegister를 만들려고 한다.~~
- 사용 가능한 자금(availableFunds)을 승인하는 이니셜 라이저 작성
- 매매(transaction)를 위해 금액을 추가(addItem)가 되어야 함
- 금액 추가가 여러명이 되어야 함
- 결재를 승인 하는 기능이 있어야 함(acceptPayment method)

The TDD Cycle

반복 - Red: 실패 테스트 케이스 작성

- 금액 추가가 여러명이 되어야 함

```
func testAddItem_twoItems_addsCostsToTransactionTotal() {
    // given
    let itemCost = Decimal(42)
    let itemCost2 = Decimal(20)
    let expectedTotal = itemCost + itemCost2
    기존의 코드를 사용하면 테스트 실패
    sut.addItem(itemCost)
    sut.addItem(itemCost2)
    // then
    XCTAssertEqual(sut.transactionTotal, expectedTotal)
}
```

```
Test Suite 'CashRegisterTests' started at 2020-07-01 23:13:45.391
Test Case '-[__lldb_expr_14.CashRegisterTests testAddItem_oneItem_addsCostToTransactionTotal]' started.
Test Case '-[__lldb_expr_14.CashRegisterTests testAddItem_oneItem_addsCostToTransactionTotal]' passed (0.114 seconds).
Test Case '-[__lldb_expr_14.CashRegisterTests testAddItem_twoItems_addsCostsToTransactionTotal]' started.
CashRegister.playground:71: error: -[__lldb_expr_14.CashRegisterTests testAddItem_twoItems_addsCostsToTransactionTotal] : XCTAssertEqual failed: ("20") is not equal to ("62")
Test Case '-[__lldb_expr_14.CashRegisterTests testAddItem_twoItems_addsCostsToTransactionTotal]' failed (0.006 seconds).
Test Case '-[__lldb_expr_14.CashRegisterTests testInitAvailableFunds_setsAvailableFunds]' started.
Test Case '-[__lldb_expr_14.CashRegisterTests testInitAvailableFunds_setsAvailableFunds]' passed (0.002 seconds).
Test Suite 'CashRegisterTests' failed at 2020-07-01 23:13:45.514.
Executed 3 tests, with 1 failure (0 unexpected) in 0.122 (0.123) seconds
```

The TDD Cycle

반복 - Green: 테스트 통과 케이스 작성

```
class CashRegister {  
    ...  
    var availableFunds: Decimal  
    var transactionTotal: Decimal = 0  
  
    ...  
    init(availableFunds: Decimal) {  
        self.availableFunds = availableFunds  
    }  
    ...  
    func addItem(_ cost: Decimal) {  
        transactionTotal += cost  
    }  
}
```

```
class CashRegister {  
    ...  
    var availableFunds: Decimal  
    var transactionTotal: Decimal = 0  
  
    ...  
    init(availableFunds: Decimal) {  
        self.availableFunds = availableFunds  
    }  
    ...  
    func addItem(_ cost: Decimal) {  
        transactionTotal += cost  
    }  
}
```

The TDD Cycle

반복 - Green: 테스트 통과 케이스 작성

```
Test Suite 'CashRegisterTests' started at 2020-07-01 23:16:32.552
Test Case '-[__lldb_expr_16.CashRegisterTests testAddItem_oneItem_addsCostToTransactionTotal]' started.
Test Case '-[__lldb_expr_16.CashRegisterTests testAddItem_oneItem_addsCostToTransactionTotal]' passed (0.124 seconds).
Test Case '-[__lldb_expr_16.CashRegisterTests testAddItem_twoItems_addsCostsToTransactionTotal]' started.
Test Case '-[__lldb_expr_16.CashRegisterTests testAddItem_twoItems_addsCostsToTransactionTotal]' passed (0.002 seconds).
Test Case '-[__lldb_expr_16.CashRegisterTests testInitAvailableFunds_setsAvailableFunds]' started.
Test Case '-[__lldb_expr_16.CashRegisterTests testInitAvailableFunds_setsAvailableFunds]' passed (0.001 seconds).
Test Suite 'CashRegisterTests' passed at 2020-07-01 23:16:32.681.
Executed 3 tests, with 0 failures (0 unexpected) in 0.127 (0.128) seconds
```

The TDD Cycle

반복 - Refactor: 클린 코드 작성

```
class CashRegisterTests: XCTestCase {  
    ...  
    var availableFunds: Decimal!  
    var sut: CashRegister!  
    ...  
    override func setUp() {  
        super.setUp()  
        availableFunds = 100  
        sut = CashRegister(availableFunds: availableFunds)  
    }  
    ...  
    override func tearDown() {  
        super.tearDown()  
        availableFunds = nil  
        sut = nil  
    }  
    ...  
    func testInitAvailableFunds_setsAvailableFunds() {  
        XCTAssertEqual(sut.availableFunds, availableFunds)  
    }  
    ...  
    func testAddItem_oneItem_addsCostToTransactionTotal() {  
        // given  
        let itemCost = Decimal(42)  
        // when  
        sut.addItem(itemCost)  
        // then  
        XCTAssertEqual(sut.transactionTotal, itemCost)  
    }  
    ...  
    func testAddItem_twoItems_addsCostsToTransactionTotal() {  
        // given  
        let itemCost = Decimal(42)  
        let itemCost2 = Decimal(20)  
        let expectedTotal = itemCost + itemCost2  
        // when  
        sut.addItem(itemCost)  
        sut.addItem(itemCost2)  
        // then  
        XCTAssertEqual(sut.transactionTotal, expectedTotal)  
    }  
}
```

```
class CashRegisterTests: XCTestCase {  
    ...  
    var availableFunds: Decimal!  
    var sut: CashRegister!  
    var itemCost: Decimal!  
    ...  
    override func setUp() {  
        super.setUp()  
        availableFunds = 100  
        sut = CashRegister(availableFunds: availableFunds)  
        itemCost = 42  
    }  
    ...  
    override func tearDown() {  
        super.tearDown()  
        availableFunds = nil  
        sut = nil  
        itemCost = nil  
    }  
    ...  
    func testInitAvailableFunds_setsAvailableFunds() {  
        XCTAssertEqual(sut.availableFunds, availableFunds)  
    }  
    ...  
    func testAddItem_oneItem_addsCostToTransactionTotal() {  
        // when  
        sut.addItem(itemCost)  
        // then  
        XCTAssertEqual(sut.transactionTotal, itemCost)  
    }  
    ...  
    func testAddItem_twoItems_addsCostsToTransactionTotal() {  
        // given  
        let itemCost2 = Decimal(20)  
        let expectedTotal = itemCost + itemCost2  
        // when  
        sut.addItem(itemCost)  
        sut.addItem(itemCost2)  
        // then  
        XCTAssertEqual(sut.transactionTotal, expectedTotal)  
    }  
}
```

The TDD Cycle

반복 - Refactor: 클린 코드 작성

```
class CashRegisterTests: XCTestCase {  
    ...  
    var availableFunds: Decimal!  
    var sut: CashRegister!  
    var itemCost: Decimal!  
  
    override func setUp() {  
        super.setUp()  
        availableFunds = 100  
        sut = CashRegister(availableFunds: availableFunds)  
        itemCost = 42  
    }  
  
    override func tearDown() {  
        super.tearDown()  
        availableFunds = nil  
        sut = nil  
        itemCost = nil  
    }  
  
    func testInitAvailableFunds_setsAvailableFunds() {  
        XCTAssertEqual(sut.availableFunds, availableFunds)  
    }  
  
    func testAddItem_oneItem_addsCostToTransactionTotal() {  
        // when  
        sut.addItem(itemCost)  
        // then  
        XCTAssertEqual(sut.transactionTotal, itemCost)  
    }  
  
    func testAddItem_twoItems_addsCostsToTransactionTotal() {  
        // given  
        let itemCost2 = Decimal(20)  
        let expectedTotal = itemCost + itemCost2  
        // when  
        sut.addItem(itemCost)  
        sut.addItem(itemCost2)  
        // then  
        XCTAssertEqual(sut.transactionTotal, expectedTotal)  
    }  
}
```

The TDD Cycle

반복 - Refactor: 클린 코드 작성

- addItem(_)를 setup()에 넣게되면 매번 테스트가 호출 될때마다 실행이 되기 때문에 addItem을 하지 않아야 하는 테스트에서 결과가 이상하게 나 올 수 있습니다.

The TDD Cycle

요구사항

- ~~ChshRegister를 만들려고 한다.~~
- 사용 가능한 자금(availableFunds)을 승인하는 이니셜 라이저 작성
- 매매(transaction)를 위해 금액을 추가(addItem)가 되어야 함
- 금액 추가가 여러명이 되어야 함
- 결재를 승인 하는 기능이 있어야 함(acceptPayment method)

The TDD Cycle

Challenge

- 결재를 승인 하는 기능이 있어야 함(acceptPayment method)
- 결재가 승인이 나면 내야하는 전체 금액에서 결재 금액을 뺀 금액과 남은 전체 금액이 같아야 함
- 결재가 승인이 나면 결재 금액과 가능 금액의 합과 가능금액이 같아야 함.

The TDD Cycle

KeyPoint

- TDD 주기 : Red -> Green -> Refactor -> 반복