## Unit Test 1

**Unit Test란 대상 코드에 대해 테스트하고자 개발자가 작성한 코드로서, 주로 특정 메소드를 실행해서 그 결과가 기대값과 일치하는지 확인하는 형태이며, Unit Test는 서로 독립적으로 수행되어야 함.**

❑ 효과

- 작성한 코드의 설계 개선 작업 시, 코드 품질에 대한 확신
- 코드 수정 시 버그를 쉽게 찾을 수 있게 해줌
- 자동화된 회귀 테스트 (Regression Test)를 가능하게 해주는 Source가 됨

❑ 작성 범위

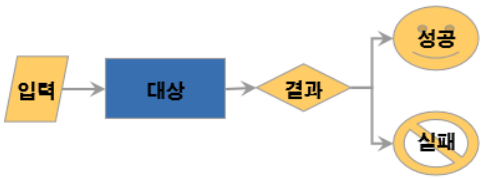
- 주요 흐름에 대한 테스트 (the happy path)
- 또 다른 주요 흐름에 대한 테스트 (the main alternative path)
- 경계 조건에 대한 테스트 (null 인자 체크 등)
- Exception 테스트 (Exception 발생하는 조건에 대한 테스트)

**각각의 테스트는 독립적으로 실행 되어야 한다. 그러기 위해 환경 일부를 리셋하거나 자원 정리가 필요.**
**테스트 환경 준비 설정 메소드**

## Unit Test 2

❑ 구성

- 테스트 프레임웍을 사용하는 Class
- 공용으로 사용하는 테스트 데이터 (test fixture)
- 테스트 데이터 준비 (Setup of test data)
- 테스트 메소드 (testXXX)
  - (테스트 별 준비)
  - 테스트 대상 메소드 실행
  - assert 문을 이용한 결과 확인 (assertTrue, assertEquals etc.)
- (내부 메소드)




```java
public class UserAdminTest {

    /* Class under test */
    private UserAdmin userAdmin;

    /* A simple test user */
    private User user;                // Test Data (Fixture)
    /* An administrator role */       //
    private Role adminRole;           //

    /**
     * Initializes the test fixture.
     */
    @Before
    public void setUp() throws Exception {
        userAdmin = new UserAdmin();
        user = new User("John", "Doe");    // Test Data Setup
        adminRole = new Role("Administrator");  //
    }

    /**
     * Test for method with ...
     */
    @Test
    public void testAddUser() {
        user.setAge(18);                   // Extra Test Setup
        userAdmin.addUser("jdoe", user, adminRole); // Use
Test Data

        User result = userAdmin.getUser("jdoe");
        assertEquals("John", result.getFirstName());
        assertEquals("Doe", result.getLastName());
    }
}
```

테스트코드작성시

1. 테스트에 필요한 모든 조건과 상황을 준비 설정한다.
   ==> 필요한 객체 모두 생성 한다.
2. 테스트 대상이 되는 메소드 호출한다.
3. 테스트 메소드가 원하는 대로 동작하는지 검증한다.
4. 실행이 끝난후 다른 코드에 영향을 주지 않도록 정리 한다.
5. 테스트코드는 실질적으로 배포 파일에서 제외 되어야 한다.

무엇을 테스트 해야 하는가?
결과가 옳은가?
경계조건을 확인?  버그는 경계조건에서 많이 발생한다.
역관계 확인?
교차 확인 다른 수단을 이용 체크?
에러 상황 확인?
성능 부분도 고려?