**Explain why unit testing**

* Use example
* Real life example/s
* R720

**Types of tests**

* Behaviour
  + Were all the correct methods/services called
  + Why is this good?
* State
  + We are checking the result being returned
  + Why is this good
* Performance
  + Runs code repeatedly to make sure it is fast enough
* Integration
  + Test the whole system and only mock external services; all internal components are run and checked

**How do I start**

* Install via
  + Gradle
    - dependencies {
    - testCompile 'junit:junit:4.12'
    - }
  + SBT
    - libraryDependencies += "junit" % "junit" % "4.12" % "test"
    - libraryDependencies += "com.novocode" % "junit-interface" % "0.11" % "test"
  + Maven
    - Dependencies, dependencyManagement
      * Scope=test
      * Version 4.11
      * >4.7, multi-threaded, limit threads! Gcc
* No test class constructor
  + Constructor exception is hidden in stack trace
* Do assume order of tests
* No hardcoded loading of data from FS
* Namespace should match between test and main class
* Prepend test to your class and method
  + Think about your name a bit to describe what you are doing
* Don’t hardcode your dates to within sysdate
* If possible, try and do thread safe tests

**Junit General**

* Asserts –not always enabled by default and illegal argument can be thrown (derived from runtime exception) so not always a good idea to enable it for non unit test classes;
* Static imports can work with junit 4 as it does not extend TestCase anymore
* @Before and @After each @Test

EX1:

* Common scenario in live is where there is a bug
* A unit test can be written to expose the bug and fix it
* Write some unit tests to test the card numbers provided and fix the bug

**Advanced Junit Usage**

* @Runwith
  + BlockJUnit4Runner is the default basically just doing a simple implementation of Runner
  + There is also Suite which groups a bunch of classes into a suite
* You also get Parameterized as Runner which allows you to provide parameters for the class
  + You get to declare your own class variables using a constructor and also a public static method with @Parameters
    - The list of array elements are then passed in sequence to the constructor executing the tests in the class X amount of time where X is the amount of test records
    - TestNG is better as you provide the parameters per method
* @Rule allows you to reduce a bit a duplication. Instead of supplying timeout for each test method, declare a timeout with class scope and @Rule. Junit will apply it to each method in the class using AOP
* Categories in @RunWith allows you to group your tests so that you can exclude or include them as you wish. Slow tests for example can be excluded.
  + Use a marker interface to designate your categories (FastTests, SlowTests)
  + Each test method or class can then be thrown into that category
  + Then you create your test suite with Categories runner and include and exclude the categories as needed.
  + Maven can then be configured to not run your slow/gui tests

**Mocking – how do I start**- Easymock vs Mockito

* In one package, you can mock interfaces; separate in EasyMock
* Mockito supports spying where verify(read) is used as opposed to EasyMock.expect which writes as well which means less code with Mockito
* You don’t have to do anything for void though you can verify it.
* Error messages in Mockito are a bit more descriptive
* Less expectations and plain verify; more verify what is needed
* Explain difference between stubs and mocks

@todo @inject

@todo research TDD