1. TDD; not insufferable nit-picker insists 100% coverage
2. Not lazy; only leave out tests hard to write
3. Only most complex parts where bugs most likely uncovered
4. Don’t waste time writing test unless minimal value

A_assertNotNull.java
- 1 – What we found in the codebase
- 25 – Bad test
- 50 – Could have driven this
- 75 – Wrapper
- 100 – Good test
- 125 – What we actually drive...but not really
- 150 – Because we could also have driven this
- 175 – Always need a test like this in such cases

B_parameters.java
- 1 – What we want to drive. Explain terms.
- 25 – Penetrable test. Easy, good, unfortunate.
- 50 – Observable test. Excellent.
- 75 – Impenetrable test. Problem.
- 100 – What we want to drive now.
- 125 – How we drive it.

C_missingAssertions.java
- 1 – Bad test: explain onEditorActionListener
- 25 – Good test: extract callback
- 50 – Two separate tests for extracted callback

D_sameValues.java
- 1 – What we want to drive
- 25 – Bad test
- 50 – What we could just as easily have driven: what’s bad?
- 75 – Good test

E_otherSideOfIf.java
- 1 – What we want to drive

25 – First test
50 – How not to write production code
75 – Right way to write production code
100 – Second test
100 – Driven code
125 – Alternative—superior—intermediate version
150 – Even better intermediate version

F_smartWrapFact.java

1 – Annoying collaborator; cannot be constructed in unit test
25 – What we want to drive
50 – But we need to drive code with a factory
75 – ...and everything's cool.
100 – But what if we have to do it many places? No.
125 – But both the code and the test would be easier.
150 – Separation of responsibilities
175 – Isolating untestable code allows testing what's testable
200 – Example test is trivially different

G_realValues.java

1 – Casting: ugh. Explain Android
25 – Test-drive a utility method to encapsulate cast
50 – Method that was driven. Anything wrong?
75 – Use a value that isn't real in the testable
100 – Prod code would need to be ridiculous for false positive

H_specifyOnce.java

1 – What we're driving
25 – Really Bad test
50 – Could have driven this
75 – Bad test, but one that will fail
100 – Good test

I_importantValues.java

1 – SALT will be used on stored data in customer devices
25 – Accidentally changed into this

50 – If the value is important, write a test

J_unrolled.java
- 1 – POJO full of sensitive data
- 25 – What we're driving
- 50 – Bad test
- 75 – Now we add another field; logout now false positive
- 100 – Better test for reflective language
- 125 – Non-reflective language can use alternate design
- 150 – Same concept for the test

K_classChecks.java
- 1 – Bus events
- 25 – What we want to drive
- 50 – Good success test
- 75 – Bad failure test
- 100 – Could have driven this instead
- 125 – Good failure test

L_quarantine.java
- 1 – What we want to drive
- 25 – Clumsy test
- 50 – Quarantine pattern
- 75 – One hand washes the other

M_noBefore.java
- 1 – Bad test: even with good prod, one will always fail
- 28 – Good test: mock created in @Before

N_tooManyMock.java – Mockito specific
- 1 – What we want to drive
- 25 – Bad test: explain @Mock, @InjectMocks, and injectMocks
- 50 – Could have driven this instead
- 75 – Can't use @Mock on generated instance fields

O_anyClass.java – Mockito specific

1 – What we want to drive
25 – Bad tests: always pass
51 – Could have driven this instead
75 – Good test
100 – Better design
125 – Easier test