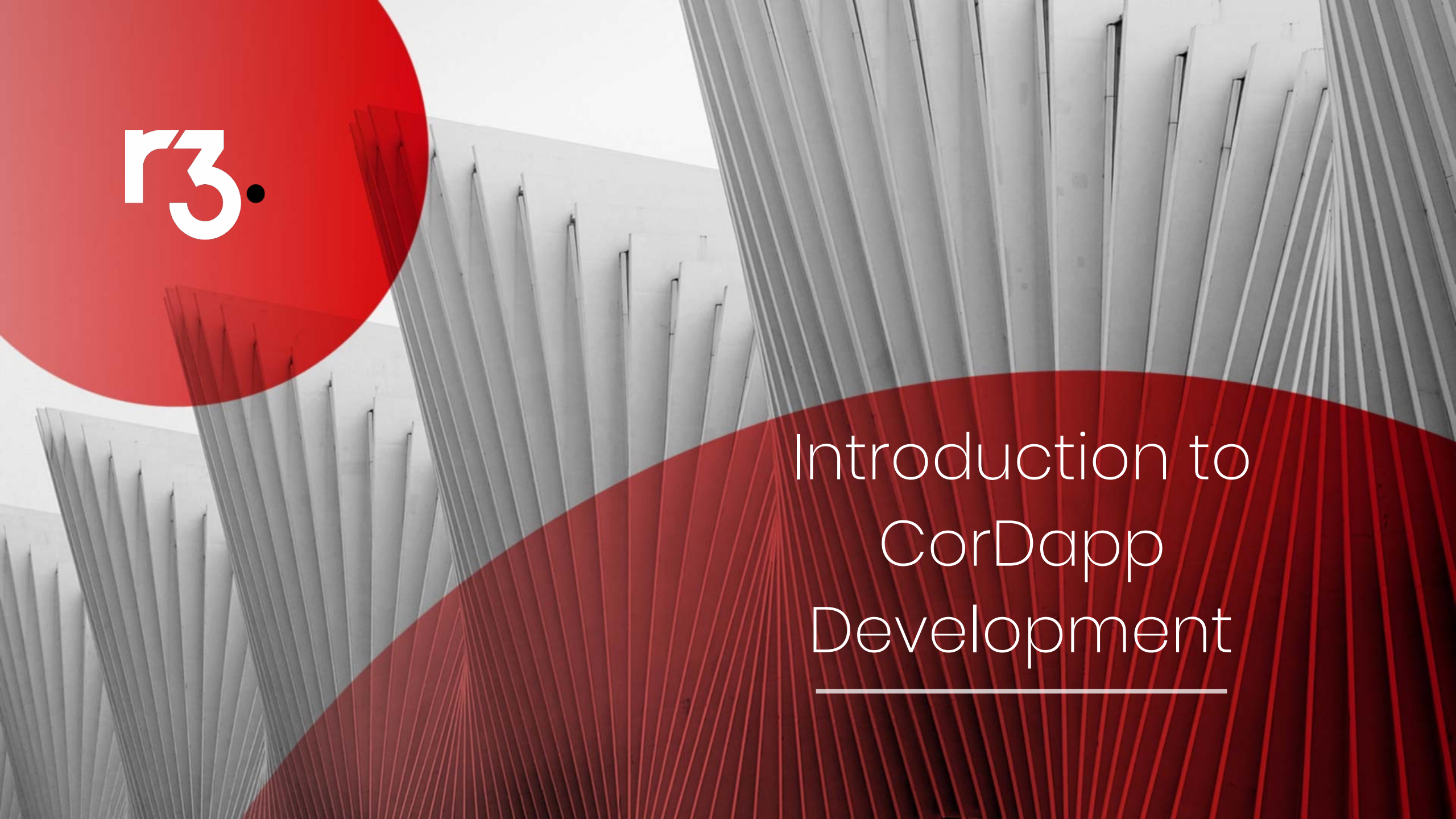
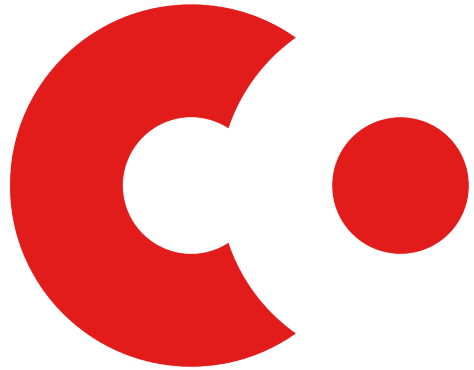


Introduction to CorDapp Development

Summary

- **Purpose**
  - Walk through building and deploying a simple CorDapp.
- **Assumptions**
  - Some programming experience ideally in Java or Kotlin.
  - Operating on a Mac or Windows machine.
- **Content**
  - All code will be shown in Java but there is also a Kotlin version available.
- **Goal**
  - By the end of this tutorial, you should have a good idea of how development on Corda works.



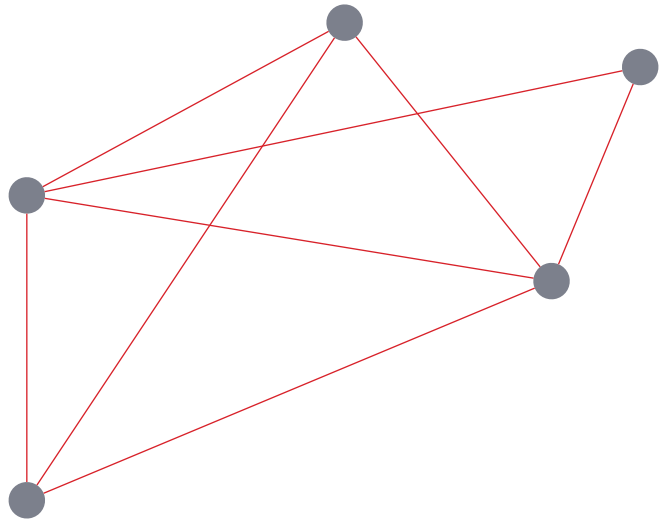
Content Overview


EXERCISE
1
Run a Corda Network

Run a Corda Network

- Before we even open up any Corda code, we are first going to do a simple CorDapp deployment locally on our machine.

- Then we are going to actually use this CorDapp and show how we can run Corda flows and vault queries.



- First, we need to install a few key technologies before we can deploy a Corda network.

- If you know you already have any of the following technologies installed on your machine then feel free to skip that step.

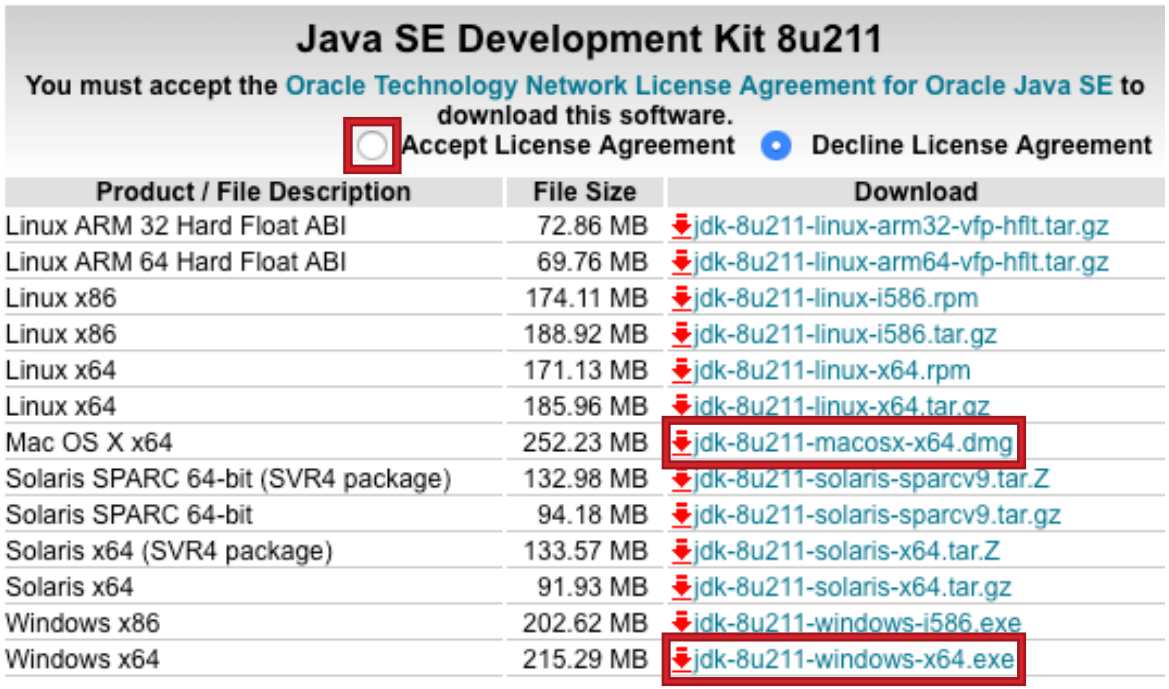
Key Technologies - Java 8 JDK

- The Java Development Kit is used to create software like CorDapps in the Java and Kotlin programming languages.

- Corda is written in the Java Virtual Machine (JVM). Therefore we will need a version of the JDK to write a CorDapp.

- It is very important that you download the correct version of Java as currently Corda is only compatible with Java 8.

- Navigate to the Java 8 JDK download page which can be found at https://www.oracle.com/technetwork/java/javase/downloads.
  - Accept the license agreement by clicking the option outlined in the image to the right.
  - Then click on the .dmg download for the appropriate machine.
  - If you are not logged into an Oracle account you may need to login or create an account for the download to begin.



Java SE Development Kit 8u211

You must accept the Oracle Technology Network License Agreement for Oracle Java SE to download this software.

○ **Accept License Agreement** ● **Decline License Agreement**

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	72.86 MB	jdk-8u211-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	69.76 MB	jdk-8u211-linux-arm64-vfp-hflt.tar.gz
Linux x86	174.11 MB	jdk-8u211-linux-i586.rpm
Linux x86	188.92 MB	jdk-8u211-linux-i586.tar.gz
Linux x64	171.13 MB	jdk-8u211-linux-x64.rpm
Linux x64	185.96 MB	jdk-8u211-linux-x64.tar.gz
Mac OS X x64	252.23 MB	jdk-8u211-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	132.98 MB	jdk-8u211-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	94.18 MB	jdk-8u211-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	133.57 MB	jdk-8u211-solaris-x64.tar.Z
Solaris x64	91.93 MB	jdk-8u211-solaris-x64.tar.gz
Windows x86	202.62 MB	jdk-8u211-windows-i586.exe
Windows x64	215.29 MB	jdk-8u211-windows-x64.exe

Key Technologies - Java 8 JDK

- Once the download is complete, double click on the file to begin installation.
  - The installation process is very straightforward but keep note of wherever you select for the install destination as you will need that later.

- To check to see if Java is installed correctly type `java –version` into a Terminal window
  - The terminal should display something similar to the image on the below.
  - You do not need this exact version but you should have a version number higher than 1.8.0_171 but less than 1.9.0_000.

```
~ myUsername$ java -version
java version "1.8.0_211"
Java(TM) SE Runtime Environment (build 1.8.0_211-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.211-b12, mixed mode)
```

Key Technologies - Gradle

- Gradle is a tool that provides build automation for CorDapps and allows for the easy management of project dependencies.

- Instructions for Gradle installation can be found at https://gradle.org/install/.
  - The manual installation process is quite straightforward but if you are using a Mac it may be easiest to use Homebrew for installation.

- To test if Gradle has been successfully installed on your machine run the command "*gradle -v*" in a Terminal window.

```
~ myUsername$ gradle -v

------------------------------------------------------------
Gradle 5.4.1
------------------------------------------------------------

Build time:   2019-04-26 08:14:42 UTC
Revision:     261d171646b36a6a28d5a19a69676cd098a4c19d

Kotlin:       1.3.21
Groovy:       2.5.4
Ant:          Apache Ant(TM) version 1.9.13 compiled on July 10 2018
JVM:          1.8.0_211 (Oracle Corporation 25.211-b12)
OS:           Mac OS X 10.14.5 x86_64
```

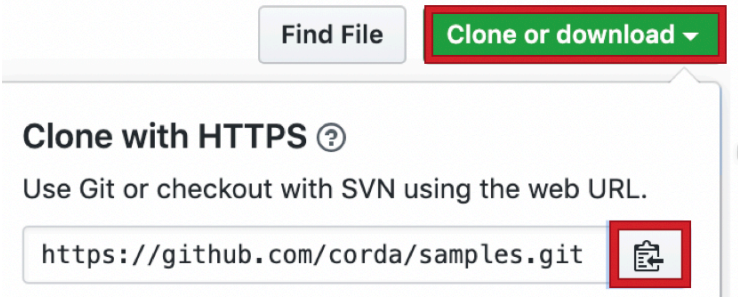
Key Technologies - Git

- Git is a version control tool for tracking changes in source code for software development.
- Git allows for the quick setup for any sample CorDapps or any other open-source CorDapps which use Git for version control.
- While git isn't actually necessary for the deployment of a CorDapp, it is a helpful tool to have if you are going to be developing in Corda.
- The link to download git can be found at https://git-scm.com/download.
- To check if git has been successfully installed on your machine type `git –version` into a Terminal window.
- In the next section, we will use git to access a whole folder of different CorDapp projects.

```
~ myUsername$ git --version
git version 2.22.0
```

Clone a CorDapp Project with Git

- Now we are going to clone the entire Corda sample Git repository onto our machine so we can use any of these sample CorDapps.
- Navigate to https://github.com/corda/samples and click the green “Clone or download” button followed by the button to the right of the link to copy it to your clipboard.
- Open Terminal on your machine and navigate to the directory where you want the project to be located.
- Now type `git clone` in the Terminal window and paste the link we copied previously.
- If successful you should see something similar to the image below.



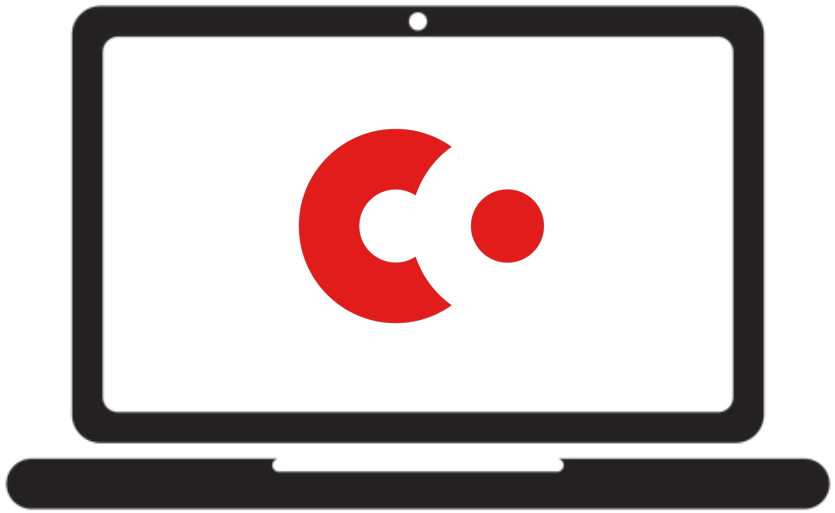

```
[~ myUsername$ git clone https://github.com/corda/samples.git
Cloning into 'samples'...
remote: Enumerating objects: 124, done.
remote: Counting objects: 100% (124/124), done.
remote: Compressing objects: 100% (92/92), done.
remote: Total 5599 (delta 41), reused 69 (delta 3), pack-reused 5475
Receiving objects: 100% (5599/5599), 3.11 MiB | 14.80 MiB/s, done.
Resolving deltas: 100% (2327/2327), done.
```

CorDapp Deployment

- As CorDapps and the technologies used in their development vary from project to project, so do their respective methods of deployment.

- Certain CorDapps contain web interfaces while some may only be used from Terminal.

- In the following slides we are going to walk through the local deployment of the CorDapp Example sample using the Terminal.



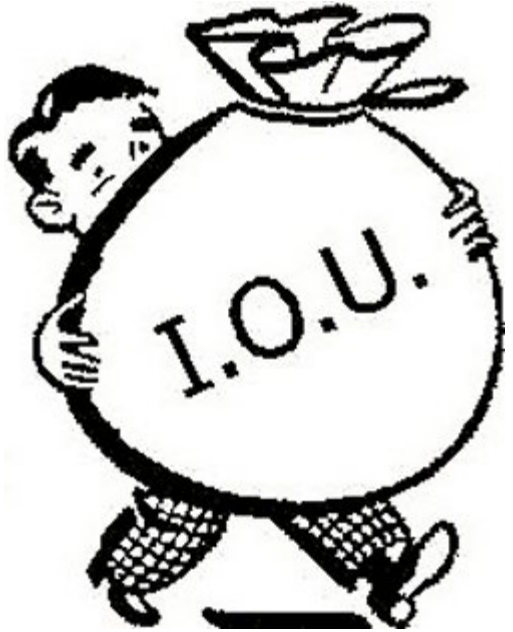
CorDapp Deployment

- We are going to do a local deployment of a sample CorDapp which allows nodes to create IOU's with one another.

- First we need to navigate to the root directory of the CorDapp in our Terminal window.

- If you are still in the Terminal window from the previous step, simply type `cd samples` followed by `cd cordapp-example` like seen in the image below.

```
~ myUsername$ cd samples
samples myUsername$ cd cordapp-example
cordapp-example myUsername$
```



- In the next step we are actually going to deploy our CorDapp.

CorDapp Deployment

- To deploy the CorDapp, run `./gradlew clean deployNodes` as seen in the image to the right.
  - On Windows this is done with the command `gradlew.bat clean deployNodes`
- This command generates all the necessary files that we need in order to run the network.
- This may take around a minute to execute so wait until you see the BUILD SUCCESSFUL message on the image to the right before proceeding.
- On the next slide we will actually deploy the network.

```
cordapp-example myUsername$ ./gradlew clean deployNodes

> Task :workflows-java:deployNodes
Running Cordform task
Deleting /Users/          /CorDapps/samples/cordapp-example/workflows-java/build/nodes
Bootstrapping local test network in /Users/          /CorDapps/samples/cordapp-example/workflows-java/build/nodes
Generating node directory for PartyA
Generating node directory for Notary
Generating node directory for PartyC
Generating node directory for PartyB
Waiting for all nodes to generate their node-info files...
Distributing all node-info files to all nodes
Loading existing network parameters... none found
Gathering notary identities
Generating contract implementations whitelist
New NetworkParameters {
      minimumPlatformVersion=4
      notaries=[NotaryInfo(identity=O=Notary, L=London, C=GB, validating=false)]
      maxMessageSize=10485760
      maxTransactionSize=524288000
      whitelistedContractImplementations {

      }
      eventHorizon=PT720H
      packageOwnership {

      }
      modifiedTime=2019-08-27T13:55:28.151Z
      epoch=1
  }
Bootstrapping complete!

> Task :workflows-kotlin:deployNodes
Running Cordform task
Deleting /Users/          /CorDapps/samples/cordapp-example/workflows-kotlin/build/nodes
Bootstrapping local test network in /Users/          /CorDapps/samples/cordapp-example/workflows-kotlin/build/nodes
Generating node directory for PartyA
Generating node directory for Notary
Generating node directory for PartyC
Generating node directory for PartyB
Waiting for all nodes to generate their node-info files...
Distributing all node-info files to all nodes
Loading existing network parameters... none found
Gathering notary identities
Generating contract implementations whitelist
New NetworkParameters {
      minimumPlatformVersion=4
      notaries=[NotaryInfo(identity=O=Notary, L=London, C=GB, validating=false)]
      maxMessageSize=10485760
      maxTransactionSize=524288000
      whitelistedContractImplementations {

      }
      eventHorizon=PT720H
      packageOwnership {

      }
      modifiedTime=2019-08-27T13:55:57.265Z
      epoch=1
  }
Bootstrapping complete!

BUILD SUCCESSFUL in 58s
24 actionable tasks: 23 executed, 1 up-to-date
```

CorDapp Deployment

- To run the nodes, enter the following command:

```
workflows-kotlin/build/nodes/runnodes
```

  - On Windows the equivalent command is:

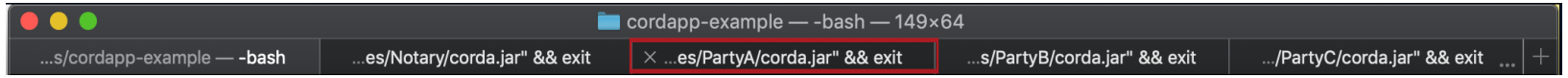
```
call workflows-kotlin\build\nodes\runnodes.bat
```

- The result should look similar to the image on the right.
- You should also see four additional Terminal windows open in new tabs.
- If you've made it this far, congrats! You have successfully deployed your first CorDapp.

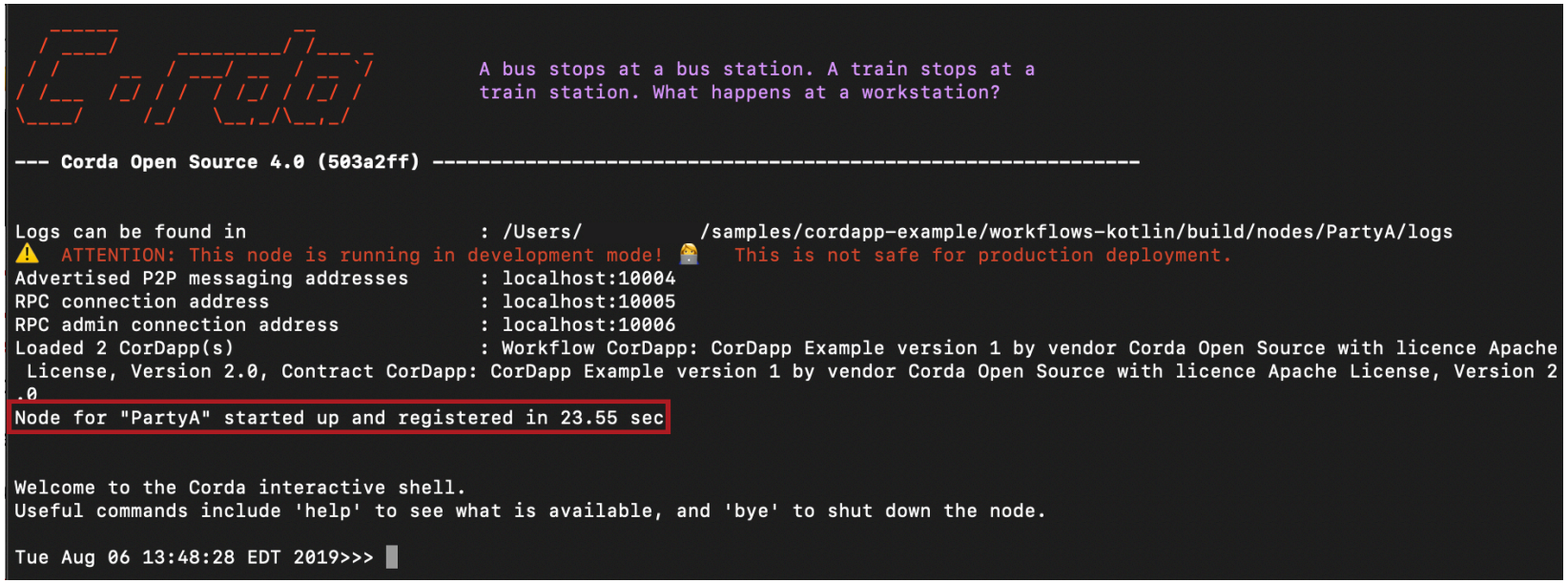
```
[cordapp-example myUsername$ workflows-kotlin/build/nodes/runnodes
Starting nodes in /Users/          /CorDapps/samples/cordapp-example/workflows-kotlin/build/nodes
Starting corda.jar in /Users/          /CorDapps/samples/cordapp-example/workflows-kotlin/build/nodes/Notary on debu
g port 5005
Node will expose jolokia monitoring port on 7005
Running command: osascript -e tell app "Terminal"
    activate
    delay 0.5
    tell app "System Events" to tell process "Terminal" to keystroke "t" using command down
    delay 0.5
    do script "bash -c 'cd \"/Users/          /CorDapps/samples/cordapp-example/workflows-kotlin/build/nodes/Notary\
" ; \"/Library/Java/JavaVirtualMachines/jdk1.8.0_211.jdk/Contents/Home/jre/bin/java\" \"-Dcapsule.jvm.args=-agentlib
:jdwp=transport=dt_socket,server=y,suspend=n,address=5005 -javaagent:drivers/jolokia-jvm-1.6.0-agent.jar=port=7005,l
ogHandlerClass=net.corda.node.JolokiaSlf4jAdapter\" \"-Dname=Notary\" \"-jar\" \"/Users/          /CorDapps/samples/
cordapp-example/workflows-kotlin/build/nodes/Notary/corda.jar\" && exit'" in selected tab of the front window
end tell
Starting corda.jar in /Users/          /CorDapps/samples/cordapp-example/workflows-kotlin/build/nodes/PartyA on debu
g port 5006
Node will expose jolokia monitoring port on 7006
Running command: osascript -e tell app "Terminal"
    activate
    delay 0.5
    tell app "System Events" to tell process "Terminal" to keystroke "t" using command down
    delay 0.5
    do script "bash -c 'cd \"/Users/          /CorDapps/samples/cordapp-example/workflows-kotlin/build/nodes/PartyA\
" ; \"/Library/Java/JavaVirtualMachines/jdk1.8.0_211.jdk/Contents/Home/jre/bin/java\" \"-Dcapsule.jvm.args=-agentlib
:jdwp=transport=dt_socket,server=y,suspend=n,address=5006 -javaagent:drivers/jolokia-jvm-1.6.0-agent.jar=port=7006,l
ogHandlerClass=net.corda.node.JolokiaSlf4jAdapter\" \"-Dname=PartyA\" \"-jar\" \"/Users/          /CorDapps/samples/
cordapp-example/workflows-kotlin/build/nodes/PartyA/corda.jar\" && exit'" in selected tab of the front window
end tell
No file corda.jar found in /Users/          /CorDapps/samples/cordapp-example/workflows-kotlin/build/nodes/.cache
Starting corda.jar in /Users/          /CorDapps/samples/cordapp-example/workflows-kotlin/build/nodes/PartyC on debu
g port 5007
Node will expose jolokia monitoring port on 7007
Running command: osascript -e tell app "Terminal"
    activate
    delay 0.5
    tell app "System Events" to tell process "Terminal" to keystroke "t" using command down
    delay 0.5
    do script "bash -c 'cd \"/Users/          /CorDapps/samples/cordapp-example/workflows-kotlin/build/nodes/PartyC\
" ; \"/Library/Java/JavaVirtualMachines/jdk1.8.0_211.jdk/Contents/Home/jre/bin/java\" \"-Dcapsule.jvm.args=-agentlib
:jdwp=transport=dt_socket,server=y,suspend=n,address=5007 -javaagent:drivers/jolokia-jvm-1.6.0-agent.jar=port=7007,l
ogHandlerClass=net.corda.node.JolokiaSlf4jAdapter\" \"-Dname=PartyC\" \"-jar\" \"/Users/          /CorDapps/samples/
cordapp-example/workflows-kotlin/build/nodes/PartyC/corda.jar\" && exit'" in selected tab of the front window
end tell
Starting corda.jar in /Users/          /CorDapps/samples/cordapp-example/workflows-kotlin/build/nodes/PartyB on debu
g port 5008
Node will expose jolokia monitoring port on 7008
Running command: osascript -e tell app "Terminal"
    activate
    delay 0.5
    tell app "System Events" to tell process "Terminal" to keystroke "t" using command down
    delay 0.5
    do script "bash -c 'cd \"/Users/          /CorDapps/samples/cordapp-example/workflows-kotlin/build/nodes/PartyB\
" ; \"/Library/Java/JavaVirtualMachines/jdk1.8.0_211.jdk/Contents/Home/jre/bin/java\" \"-Dcapsule.jvm.args=-agentlib
:jdwp=transport=dt_socket,server=y,suspend=n,address=5008 -javaagent:drivers/jolokia-jvm-1.6.0-agent.jar=port=7008,l
ogHandlerClass=net.corda.node.JolokiaSlf4jAdapter\" \"-Dname=PartyB\" \"-jar\" \"/Users/          /CorDapps/samples/
cordapp-example/workflows-kotlin/build/nodes/PartyB/corda.jar\" && exit'" in selected tab of the front window
end tell
Started 4 processes
Finished starting nodes
```

Run a Corda Flow

- Now we are going to actually use this CorDapp and run a Corda “flow”.
  - We will talk more about Corda flows later on but for now just know that they help coordinate ledger updates.
- First, navigate to the “PartyA” Terminal window like seen below.



- Now you should see something similar to the image below.



- Make sure to wait until it says the node has started up and been registered as it does in the image above.

Run a Corda Flow

- Now to run our flow, copy the following command.
  - `flow start ExampleFlow$Initiator iouValue: 50, otherParty: "O=PartyB,L=New York,C=US"`
- In this example, PartyA is creating an IOU which shows that PartyB owes PartyA an amount of 50.
- If the flow ran correctly, your Terminal window should look similar to the one below.

```
[Tue Aug 06 14:22:00 EDT 2019>>> flow start ExampleFlow$Initiator iouValue: 50, otherParty: "O=PartyB,L=New York,C=US"

  ✅   Starting
  ✅   Generating transaction based on new IOU.
  ✅   Verifying contract constraints.
  ✅   Signing transaction with our private key.
  ✅   Gathering the counterparty's signature.
      ✅   Collecting signatures from counterparties.
      ✅   Verifying collected signatures.
  ✅   Obtaining notary signature and recording transaction.
          Requesting signature by notary service
              Requesting signature by Notary service
              Validating response from Notary service
      ✅   Broadcasting transaction to participants
  ➡️   Done
Flow completed with result: SignedTransaction(id=388FEA783CE176D7C16A6DD453D0C760AFCE86C6A4504B2D1E06136A2B6570D6)
```

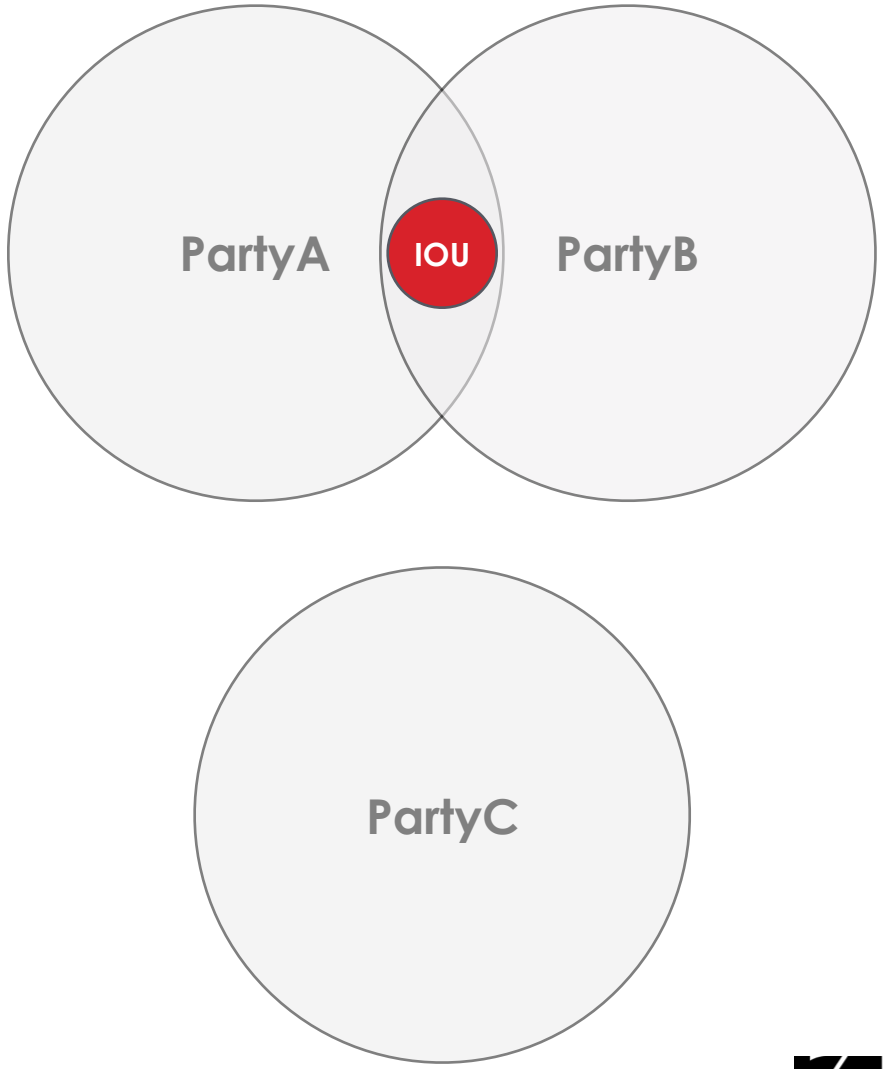
Run a Vault Query

- Next, we are going to run a vault query to check that the IOUState now exists.
  - A node's vault is really just a classic relational database that represents that specific node's view of the ledger.
- Run the following command to perform a vault query of all the IOUStates in this node's vault.
  - `run vaultQuery contractStateType: com.example.state.IOUState`
- If the vault query executes correctly, you should see the IOUState you created similarly to the image on the right.

```
Tue Aug 06 14:26:00 EDT 2019>>> run vaultQuery contractStateType: com.example.state.IOUState
states:
- state:
    data: !<com.example.state.IOUState>
      value: 50
      lender: "O=PartyA, L=London, C=GB"
      borrower: "O=PartyB, L=New York, C=US"
      linearId:
        externalId: null
        id: "9a53603b-ea94-4562-8fe9-f9cd7d2d8f01"
    contract: "com.example.contract.IOUContract"
    notary: "O=Notary, L=London, C=GB"
    encumbrance: null
    constraint: !<net.corda.core.contracts.SignatureAttachmentConstraint>
      key: "aSq9DsNNvGhYxYyqA9wd2eduEAZ5AXWgJTbTEw3G5d2maAq8vtLE4kZHgCs5jcB1N31cx1hpsLeqG2ngSysVHqcXhb
Nts6SkRWDaV7xNcr6MtcbufGUchxredBb6"
  ref:
    txhash: "388FEA783CE176D7C16A6DD453D0C760AFCE86C6A4504B2D1E06136A2B6570D6"
    index: 0
statesMetadata:
- ref:
    txhash: "388FEA783CE176D7C16A6DD453D0C760AFCE86C6A4504B2D1E06136A2B6570D6"
    index: 0
  contractStateClassName: "com.example.state.IOUState"
  recordedTime: "2019-08-06T18:22:45.866Z"
  consumedTime: null
  status: "UNCONSUMED"
  notary: "O=Notary, L=London, C=GB"
  lockId: null
  lockUpdateTime: null
  relevancyStatus: "RELEVANT"
  constraintInfo:
    constraint:
      key: "aSq9DsNNvGhYxYyqA9wd2eduEAZ5AXWgJTbTEw3G5d2maAq8vtLE4kZHgCs5jcB1N31cx1hpsLeqG2ngSysVHqcXhb
Nts6SkRWDaV7xNcr6MtcbufGUchxredBb6"
totalStatesAvailable: -1
stateTypes: "UNCONSUMED"
otherResults: []
```

Run a Vault Query

- Now try running the same exact command in the Terminal window for PartyB and see what happens.
- You should see the same exact content in PartyB's vault.
- Because PartyB was also a participant in this transaction, an identical IOUState is also stored in their vault.
- If you try this same command for PartyC, you will see this node's vault is empty.
- This demonstrates how Corda uses a peer-to-peer messaging system where data propagates around the network on a need to know basis.



EXERCISE

2

Setup your Development Environment

Clone our CorDapp Project with Git

- In this section you will be introduced to IntelliJ, the IDE that we will be using for our development environment.

- However, first we need to clone a new git repository.

- Just as you did in the previous exercise git clone the repository that can be found at https://github.com/corda/corda-training-template/tree/corda-training-template-v2.

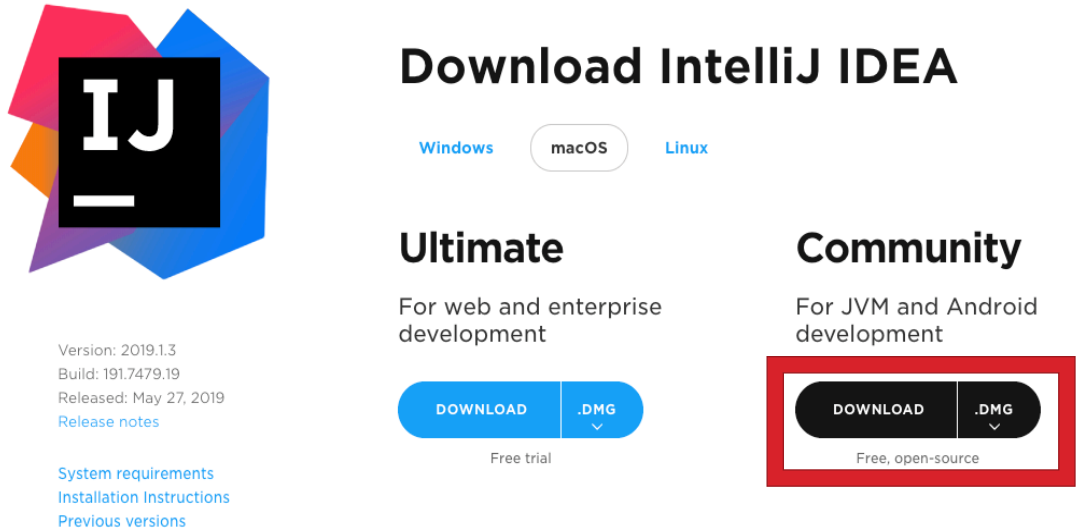
IntelliJ IDE

- Intellij is an integrated development environment (IDE) that was built specifically for Java (and Kotlin) development.

- Provides developers with an easy way to create and deploy CorDapps through a single graphical user interface.

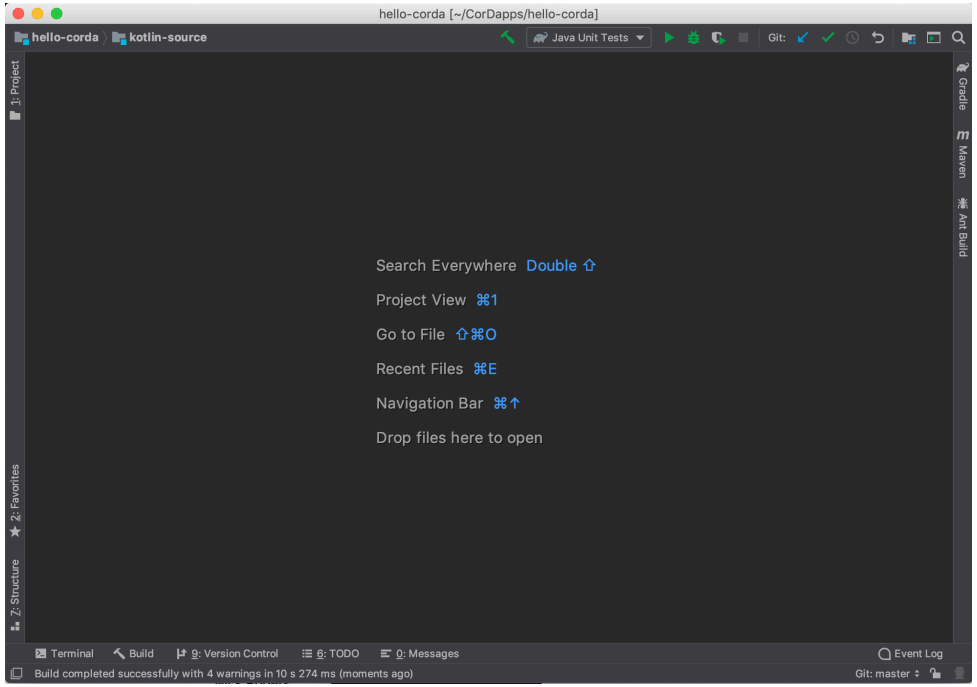
- Navigate to https://www.jetbrains.com/idea/download and download the community version by clicking the button specified by the image to the right.

- Follow the prompts to install the application. Use the default configuration options unless you are otherwise instructed by your system admin.



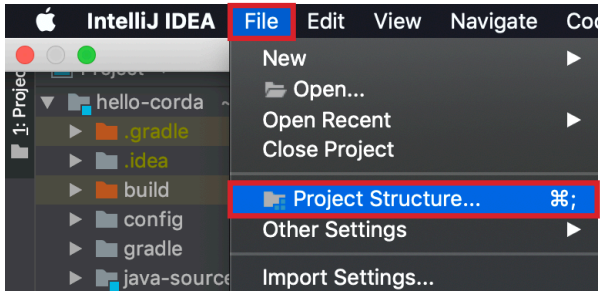
CorDapp Development Setup – Open Project

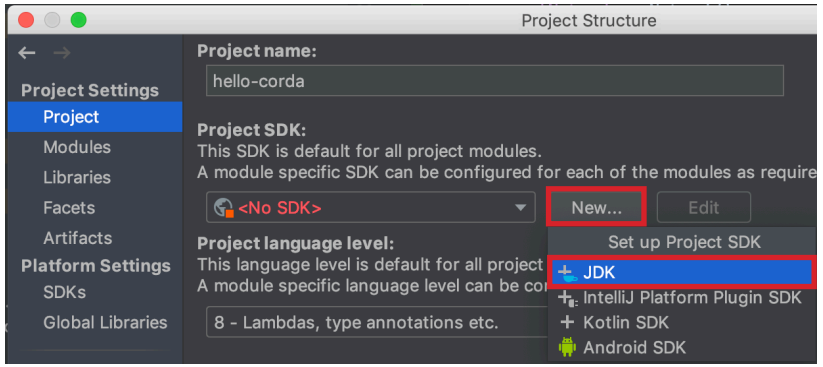
- Open IntelliJ on your machine and select “Open” from the start menu as seen on the image to the right.
- Navigate to the root directory of the CorDapp (hello-corda) and select it.
- IntelliJ should open the project and you should see something similar to the image below.

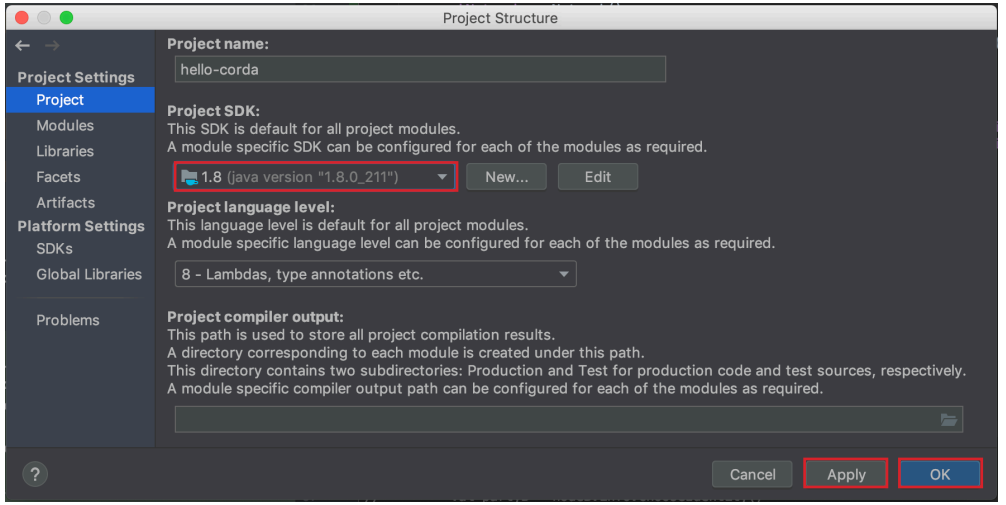


CorDapp Development Setup – Select SDK

- Now we must tell IntelliJ to use the version of the Java JDK that we downloaded from earlier.
- On the menu bar, click on “File” on the top right of the screen and then click on “Project Structure”.
- From the Project Structure window click on “New…” and then “JDK” as seen in the image on the right.
- Navigate to the location of where you installed your Java JDK.
  - If you don’t remember the path, the default is typically something similar to this */Library/Java/JavaVirtualMachines/jdk1.8.0_211.jdk/Contents/Home.*
- Now you should see your version of Java under ”Project SDK”.
  - Before closing this window, be sure to click ”Apply” and then “OK” in the bottom right corner or your changes may not save.

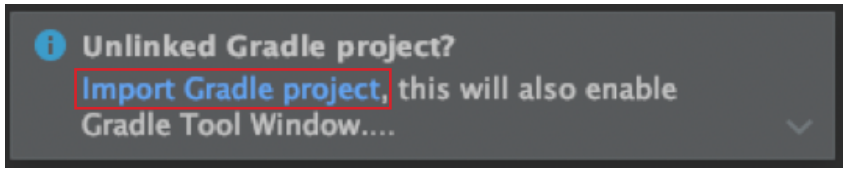


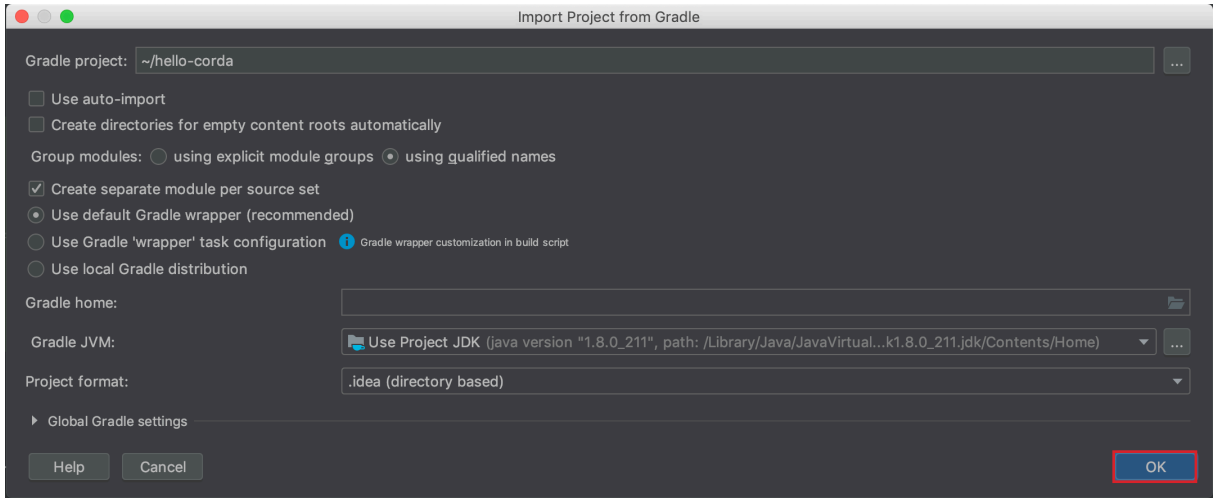


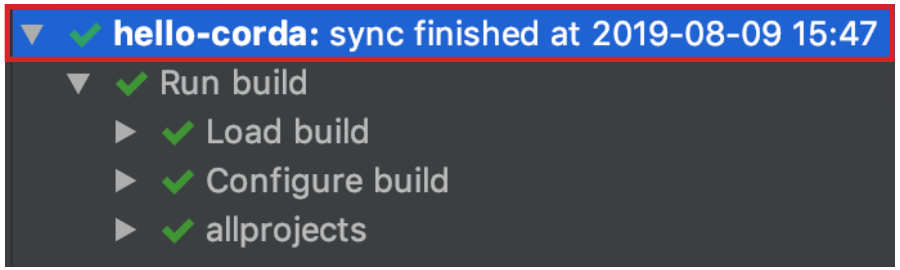


CorDapp Development Setup – Import Gradle

- Click "*Import Gradle project*" on the pop up that appears on the bottom right of the screen like the image to the right.
  - If this pop up does not appear, click on 1 Event Log in the bottom right corner of the screen.
  - Now click "Import Gradle Project" in the message that appears in the event log.
- A menu similar to the one on the right should appear on the screen.
  - Simply select "OK" and wait for the Gradle project to sync.
- This may take a few minutes but when completed you should see a message similar to the image on the right.





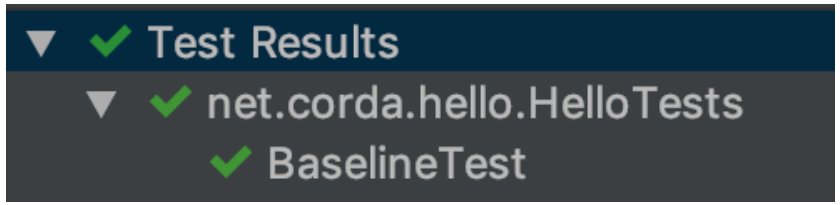


IntelliJ Run Configurations

- We are now going to show how to use IntelliJ run configurations.
- We've created a couple run configurations for this project but for now we are just going to use one of them to run a baseline test to make sure the IntelliJ project is setup properly.
- In the top right of your IntelliJ window you should see something similar to the image below.
  - Click on the drop down menu that is highlighted to show all saved run configurations.

- Then select "BaseLine Test" from the dropdown menu and click the run button.

- You should see the test run and pass on the bottom of the IntelliJ window.



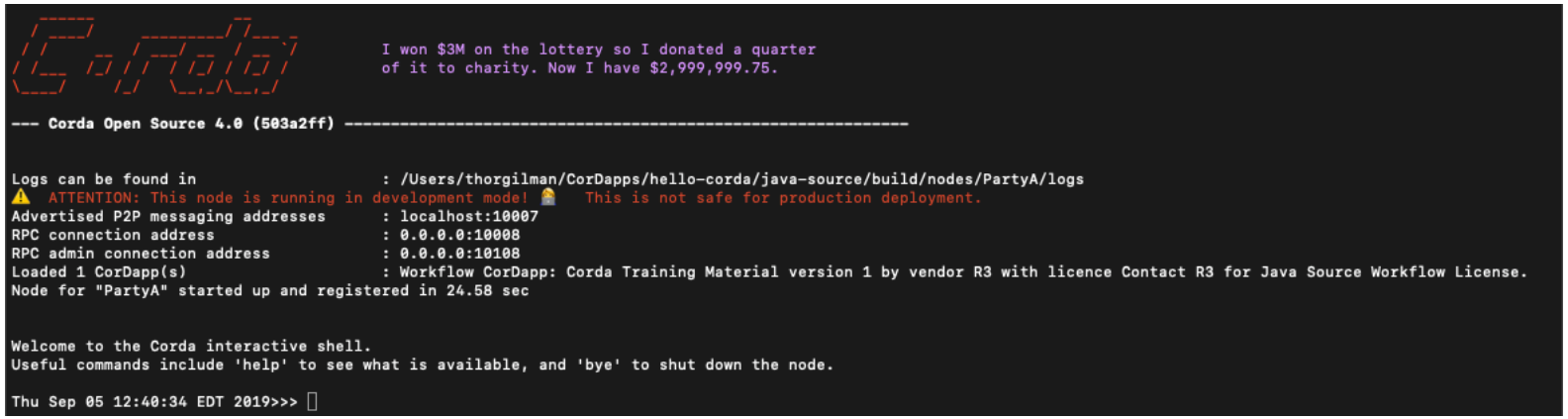

EXERCISE
3
Hello Corda!

Hello Corda!

- In the previous exercise we setup the HelloCorda CorDapp in the IntelliJ development environment.
- This CorDapp is an incomplete messaging application that is meant to allow nodes on a network to send messages to each other.
- In this exercise, we are going to make a couple changes to the code in order to make this a functioning messaging application.
- You aren't expected to fully understand what is going on in this code so don't worry if you are somewhat confused during this exercise.
- In future exercises, you will go into more depth but this exercise is simply meant to help familiarize yourself with some Corda code.
- We are first going to deploy the CorDapp as it is, then we will make some changes and deploy the application again.

Hello Corda! - Deployment

- This deployment process is going to follow very closely to how you deployed the sample CorDapp previously.
- From a Terminal window, navigate to the root directory of the HelloCorda CorDapp (hello-corda).
- Enter `./gradlew clean java-source:deployNodes` into your Terminal window.
  - You should see a BUILD SUCCESSFUL message if the nodes have been properly deployed.
- Next enter `java-source/build/nodes/runnodes` in order to start the network.
  - Like before, you should see three additional Terminal windows open.
- Navigate to the tab for PartyA and you should now see the Corda interactive shell.



Hello Corda! - Deployment

- Now we are going to run a flow & vault query just like before.
- Flow:
  - `start MessageFlow target: "O=PartyB,L=New York,C=US”`
- Vault Query:
  - `run vaultQuery contractStateType: net.corda.hello.MessageState`
- In its current form, this application essentially shares shares a state with another party but the only content within the state are the *origin* & *target* of the state itself.
- Next, we are going to add some functionality which will add a *content* field so messages can be shared between parties.

```
Thu Sep 05 12:40:34 EDT 2019>>> start MessageFlow target: "O=PartyB,L=New York,C=US"

 ✅   Starting
          Requesting signature by notary service
              Requesting signature by Notary service
[             Validating response from Notary service                                ]
     ✅   Broadcasting transaction to participants
➡️   Done
Flow completed with result: SignedTransaction(id=5572784AFC1958972A30DACD4F54BBFBB6D94B5ABC276A38D4D
D6A5014700292)

Thu Sep 05 12:43:28 EDT 2019>>> run vaultQuery contractStateType: net.corda.hello.MessageState
states:
- state:
    data: !<net.corda.hello.MessageState>
      origin: "O=PartyA, L=London, C=GB"
      target: "O=PartyB, L=New York, C=US"
    contract: "net.corda.hello.MessageContract"
    notary: "O=Notary, L=London, C=GB"
    encumbrance: null
    constraint: !<net.corda.core.contracts.SignatureAttachmentConstraint>
      key: "aSq9DsNNvGhYxYyqA9wd2eduEAZ5AXWgJTbTEw3G5d2maAq8vtLE4kZHgCs5jcB1N31cx1hpsLeqG2ngSysVHqcX
hbNts6SkRWDaV7xNcr6MtcbufGUchxredBb6"
  ref:
    txhash: "5572784AFC1958972A30DACD4F54BBFBB6D94B5ABC276A38D4DD6A5014700292"
    index: 0
statesMetadata:
- ref:
    txhash: "5572784AFC1958972A30DACD4F54BBFBB6D94B5ABC276A38D4DD6A5014700292"
    index: 0
  contractStateClassName: "net.corda.hello.MessageState"
  recordedTime: "2019-09-05T16:43:27.117Z"
  consumedTime: null
  status: "UNCONSUMED"
  notary: "O=Notary, L=London, C=GB"
  lockId: null
  lockUpdateTime: null
  relevancyStatus: "RELEVANT"
  constraintInfo:
    constraint:
      key: "aSq9DsNNvGhYxYyqA9wd2eduEAZ5AXWgJTbTEw3G5d2maAq8vtLE4kZHgCs5jcB1N31cx1hpsLeqG2ngSysVHqcX
hbNts6SkRWDaV7xNcr6MtcbufGUchxredBb6"
totalStatesAvailable: -1
stateTypes: "UNCONSUMED"
otherResults: []
```

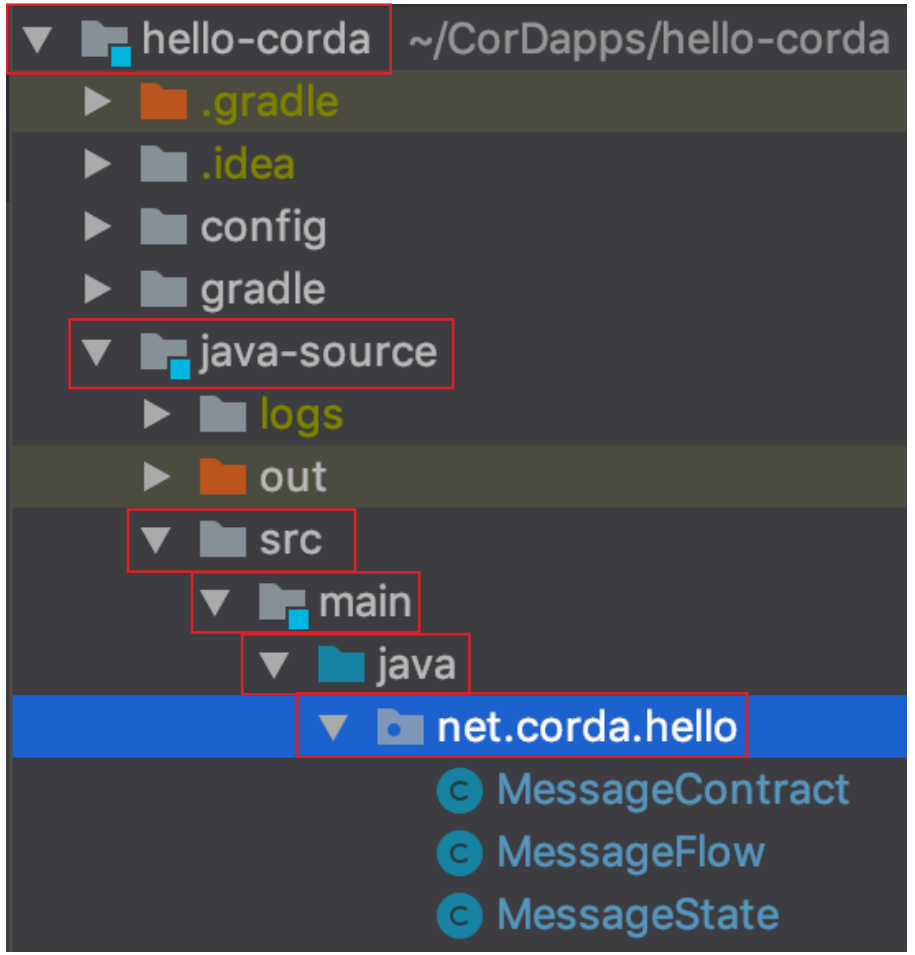
Hello Corda!

- Let's start by navigating to the *net.corda.hello* package that can be found at the following class path seen on the right.

- Here we have Java classes for a simple Corda State, Contract and Flow.

- In the following slides we are going to a small make change to each of these files.

- But first let's look at a quick refresher of some Corda key concepts on the next slide.



corda

Corda Platform Overview

- This is a brief overview of the different components that make up the Corda Platform.
- Next, we will take a look at the code for each and make a couple changes to the HelloCorda CorDapp.

States

States are immutable objects that represent (shared) facts such as a financial agreement or contract at a specific point in time

Transactions

Transactions consume input states and create output states. Newly created output states replace the input states which are marked as historic

Contracts

Contracts control the evolution of states by allowing relevant participants to check for transaction validity

Flows

Flows are light-weight processes used to coordinate interactions required for peers to create and agree transactions

r3.

Hello Corda! - State

- Navigate to the State class by double clicking on the MessageState Java file.
- In Corda, States represent on-ledger facts that may be shared with other parties on the network.
- Here we have a state that takes in two parameters *origin* & *target* of type Party.
- On lines 12-14 we override the *getParticipants()* method to pass in our *origin* & *target* parameters.
    - A state's *getParticipants()* method defines the list of Partys that will hold this state in their vault.

```
1  @BelongsToContract(MessageContract.class)
2  public class MessageState implements ContractState {
3      public final Party origin;
4      public final Party target;
5      public MessageState(Party origin, Party target) {
6          this.origin = origin;
7          this.target = target;
8      }
9      @Override
10     public List<AbstractParty> getParticipants() {
11         return ImmutableList.of(origin, target);
12     }
13 }
```

Hello Corda! - State (Task #1)

- Above the MessageState class you should see a comment that describes your task as the following:
  - **TODO(#1): Update this code so that the MessageState constructor takes an additional parameter 'content' of type String. This parameter should set a local parameter that is also named 'content'.**
- Model your answer after how the *origin* & *target* variables are created.
- Don't worry if other sections of the code seem to break after making this change, we will correct this in the next task.

Hello Corda! - Flow

- Next let's take a look at the Flow for this CorDapp.
- This class is more complex than the State but don't worry, you aren't expected to understand exactly what every line of code does.
- Here we have a Corda Flow that creates a MessageState and shares it with another node on the network.
- Flows often contain the most complex code within a CorDapp as they are responsible for the coordination of information around the network.
- Flows typically have two parts, an Initiator and a Responder
  - The initiating node's call function is executed when that node runs the flow and the responder is executed for each party the initiator is communicating with.
- Typically the initiating node will formulate a transaction that describes the States and Contracts involved.
- Then that transaction will be signed by various parties before it is committed to the ledger by running FinalityFlow.

```java
public class MessageFlow {

    @StartableByRPC
    @InitiatingFlow
    public static class Initiator extends FlowLogic<SignedTransaction> {

        private final Party target;
        public Initiator(Party target) {
            this.target = target;
        }

        @Suspendable
        @Override
        public SignedTransaction call() throws FlowException {
            // Step #1: Get initiating party & notary
            final Party origin = getServiceHub().getMyInfo().getLegalIdentities().get(0);
            final Party notary = getServiceHub().getNetworkMapCache().getNotaryIdentities().get(0);

            // Step #2: Create transaction items
            final MessageState state = new MessageState(origin, target);
            final Command<SendMessage> command = new Command<>(new SendMessage(), ImmutableList.of(origin.getOwningKey()));
            final StateAndContract stateAndContract = new StateAndContract(state, MessageContract.ID);

            // Step #3: Create & sign transaction
            final TransactionBuilder builder = new TransactionBuilder(notary).withItems(stateAndContract, command);
            builder.verify(getServiceHub());
            final SignedTransaction stx = getServiceHub().signInitialTransaction(builder);

            // Step #4: Send transaction to other party and finalize
            final FlowSession targetSession = initiateFlow(target);
            return subFlow(new FinalityFlow(stx, Arrays.asList(targetSession)));
        }
    }

    @InitiatedBy(Initiator.class)
    public static class Responder extends FlowLogic<SignedTransaction> {

        private final FlowSession otherPartySession;
        public Responder(FlowSession otherPartySession) {
            this.otherPartySession = otherPartySession;
        }

        @Suspendable
        @Override
        public SignedTransaction call() throws FlowException {
            return subFlow(new ReceiveFinalityFlow(otherPartySession));
        }
    }
}
```

Hello Corda! - Flow (Task #2)

- Above the flow class definition, you should see another task that is copied below.
  - **TODO(#2): Update this code so that the constructor of the Initiating flow takes an additional parameter 'content' of type String. Then the constructor of the Initiating flow should set the 'content' parameter to a local variable of the same name. Lastly, pass in 'content' as an additional parameter when we create our MessageState object.'**

- This is going to complete the additional functionality to our flow that allows us to send customized messages instead of the default "Hello Corda!" message.

- Once this change is complete, you should no longer see any compile errors in your state or flow class.

Hello Corda! - Contract

- Lastly, we have our Contract code.
- In Corda, Contracts provide rules that govern how States can evolve over time.
- On lines 3-5 we define one command called *SendMessage.*
- On lines 9-16, the verify method defines the rules for how the *SendMessage* command can modify our *MessageState.*
- Currently, there are only three conditions for the *SendMessage* command.
  - *"There should be no input state."*
  - *"There should be one output state."*
  - *"The party sending the message must sign the SendMessage transaction."*

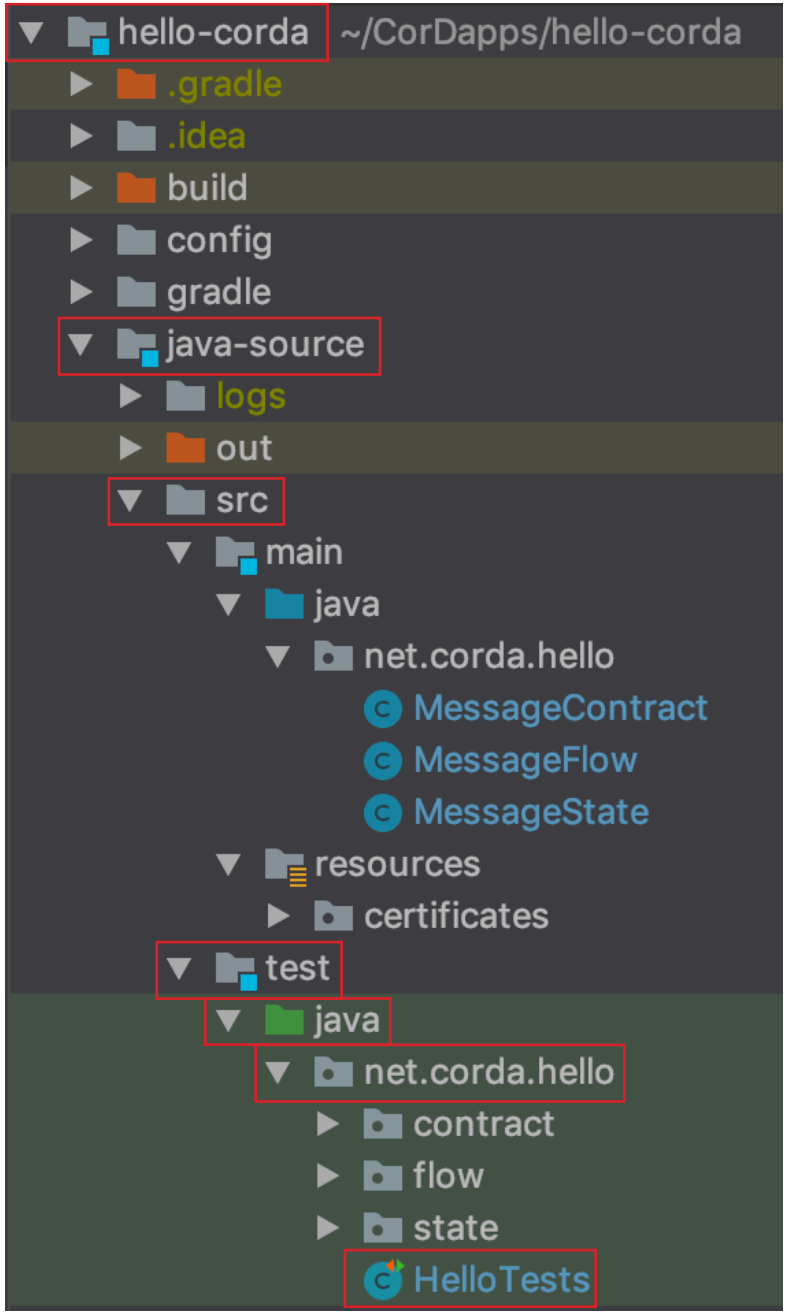
```
1  public class MessageContract implements Contract {
2      static final String ID = "net.corda.hello.MessageContract";
3      public interface Commands extends CommandData {
4          class SendMessage extends TypeOnlyCommandData implements Commands{}
5      }
6      @Override
7      public void verify(LedgerTransaction tx) {
8          final CommandWithParties<SendMessage> command = requireSingleCommand(tx.getCommands(), SendMessage.class);
9          requireThat( require -> {
10             require.using("There should be no input state.", tx.getInputStates().isEmpty());
11             require.using("There should one output state.", tx.getOutputStates().size() == 1);
12             final MessageState outputState = tx.outputsOfType(MessageState.class).get(0);
13             require.using("The party sending the message must sign the SendMessage transaction.",
14                     (command.getSigners().equals(Collections.singletonList(outputState.origin.getOwningKey()))));
15             return null;
16         });
17     }
18 }
```

Hello Corda! - Contract (Task #3)

- Now we are going to add an additional condition to MessageContract as described below.
  - **TODO(#3): Add an additional requirement to this contract so that the 'content' variable in a MessageState cannot be an empty String. Make sure the message in the require statement is as follows "*The content cannot be an empty String.*".**
- Use the existing conditions in order to formulate your answer.
- Make sure that the require statement message is exactly as stated, otherwise the solutions test we are going to run next will not pass.

Hello Corda! - Test Your Changes

- Remember earlier when we ran that baseline test through run configurations?
- Now we are going to navigate to the file where the code for that test is held.
- Because we made changes to our code, we need to comment out that test so it doesn't cause a syntax error.
- Then we are going to run a new test in order to test our changes.
- Once you have navigated to the file shown in the image to the left, comment out the baseline test and then uncomment the solution test.
- Now you can run the solution test from either the run configurations or by clicking on the green arrow to the right of the test.
- If the test passes, congrats! Your solution should be working. Now let's test it out for real.



Hello Corda! - Deployment

- We are going to follow a similar deployment process as before expect now we will pass in the content parameter when we run our flow.
- Start your network with the following two commands like before.
  - `./gradlew clean java-source:deployNodes`
  - `java-source/build/nodes/runnodes`
- Then navigate to the tab for PartyA like before.
- Now we will run the same flow as before but with an additional parameter.
  - `start MessageFlow target: "O=PartyB,L=New York,C=US", content: "Hello Corda!”`
  - Feel free to add any message you want! There’s no need for it to be “Hello Corda!”.

```
[Thu Sep 05 15:46:20 EDT 2019>>> start MessageFlow target: "O=PartyB,L=New York,C=US", content: "Hello Corda!"

  ✅   Starting
           Requesting signature by notary service
               Requesting signature by Notary service
               Validating response from Notary service
       ✅   Broadcasting transaction to participants
➡️   Done
Flow completed with result: SignedTransaction(id=C73895EDFF95986A4465F71BE26C26F45D7DE61EE51A7DAD8B39748B49F97691)
```

Hello Corda! - Deployment

- Finally, let's do another vault query like before.
  - `run vaultQuery contractStateType: net.corda.hello.MessageState`

- Now you should be able to see your message appear in the vault query like seen in the image on the right.

- If you wish, you can also run the same vault query for PartyB and you should see an identical result.

```
Thu Sep 05 16:01:12 EDT 2019>>> run vaultQuery contractStateType: net.corda.hello.MessageState
states:
- state:
    data: !<net.corda.hello.MessageState>
      origin: "O=PartyA, L=London, C=GB"
      target: "O=PartyB, L=New York, C=US"
      content: "Hello Corda!"
    contract: "net.corda.hello.MessageContract"
    notary: "O=Notary, L=London, C=GB"
    encumbrance: null
    constraint: !<net.corda.core.contracts.SignatureAttachmentConstraint>
      key: "aSq9DsNNvGhYxYyqA9wd2eduEAZ5AXWgJTbTEw3G5d2maAq8vtLE4kZHgCs5jcB1N31cx1hpsLeqG2ngSysVHq
cXhbNts6SkRWDaV7xNcr6MtcbufGUchxredBb6"
  ref:
    txhash: "92CC8A6E7FE040E6DE4072CB63EA67EC5BBFCD400C79FD229C541C9DCDE85722"
    index: 0
statesMetadata:
- ref:
    txhash: "92CC8A6E7FE040E6DE4072CB63EA67EC5BBFCD400C79FD229C541C9DCDE85722"
    index: 0
  contractStateClassName: "net.corda.hello.MessageState"
  recordedTime: "2019-09-05T20:01:11.373Z"
  consumedTime: null
  status: "UNCONSUMED"
  notary: "O=Notary, L=London, C=GB"
  lockId: null
  lockUpdateTime: null
  relevancyStatus: "RELEVANT"
  constraintInfo:
    constraint:
      key: "aSq9DsNNvGhYxYyqA9wd2eduEAZ5AXWgJTbTEw3G5d2maAq8vtLE4kZHgCs5jcB1N31cx1hpsLeqG2ngSysVHq
cXhbNts6SkRWDaV7xNcr6MtcbufGUchxredBb6"
totalStatesAvailable: -1
stateTypes: "UNCONSUMED"
otherResults: []
```

Hello Corda!

- Congrats! You have completed this part of the Corda Developer Training!

- Hopefully you now have a better idea how Corda works and are comfortable looking at CorDapp code.

Thank you

r3.com

New York
11 West 42nd Street, 8th Floor
New York, NY 10036

London
2 London Wall Place,
London, EC2Y 5AU

Singapore
80 Robinson Road, #09-04
Singapore, 068898