Hedera Hashgraph Training Materials

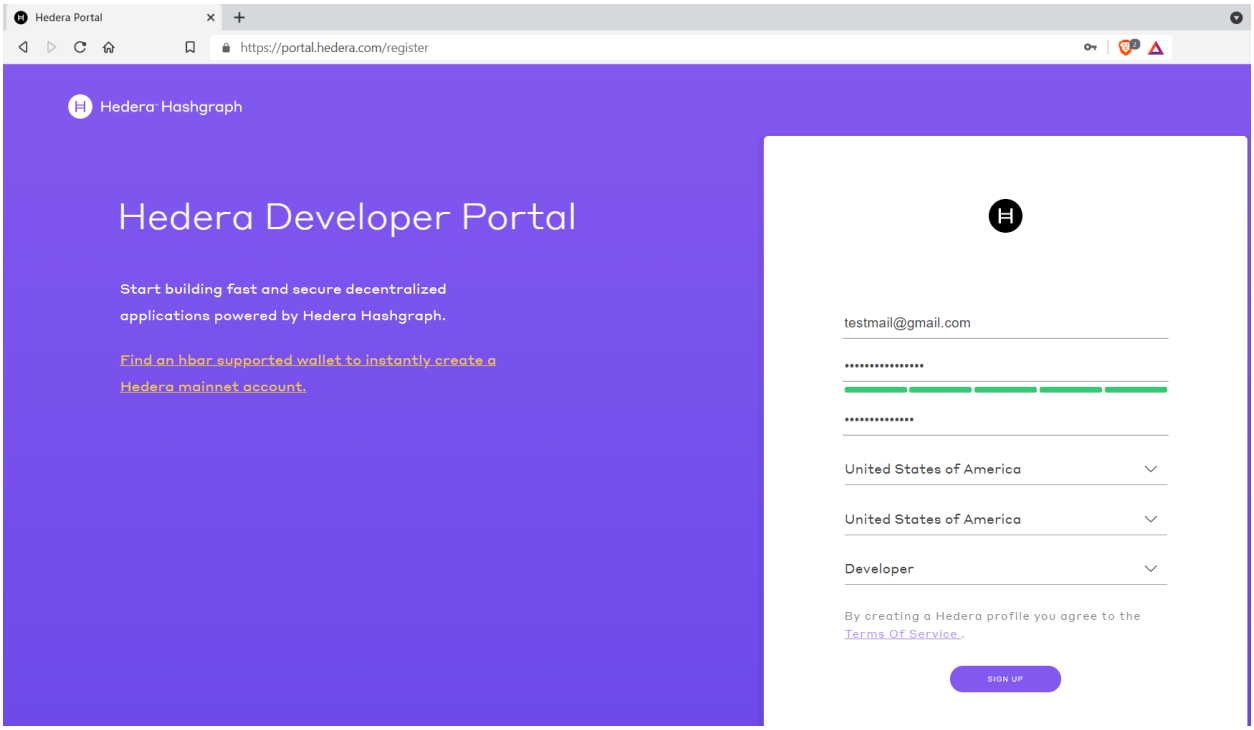
Prerequisites

Get a Header Testnet account.

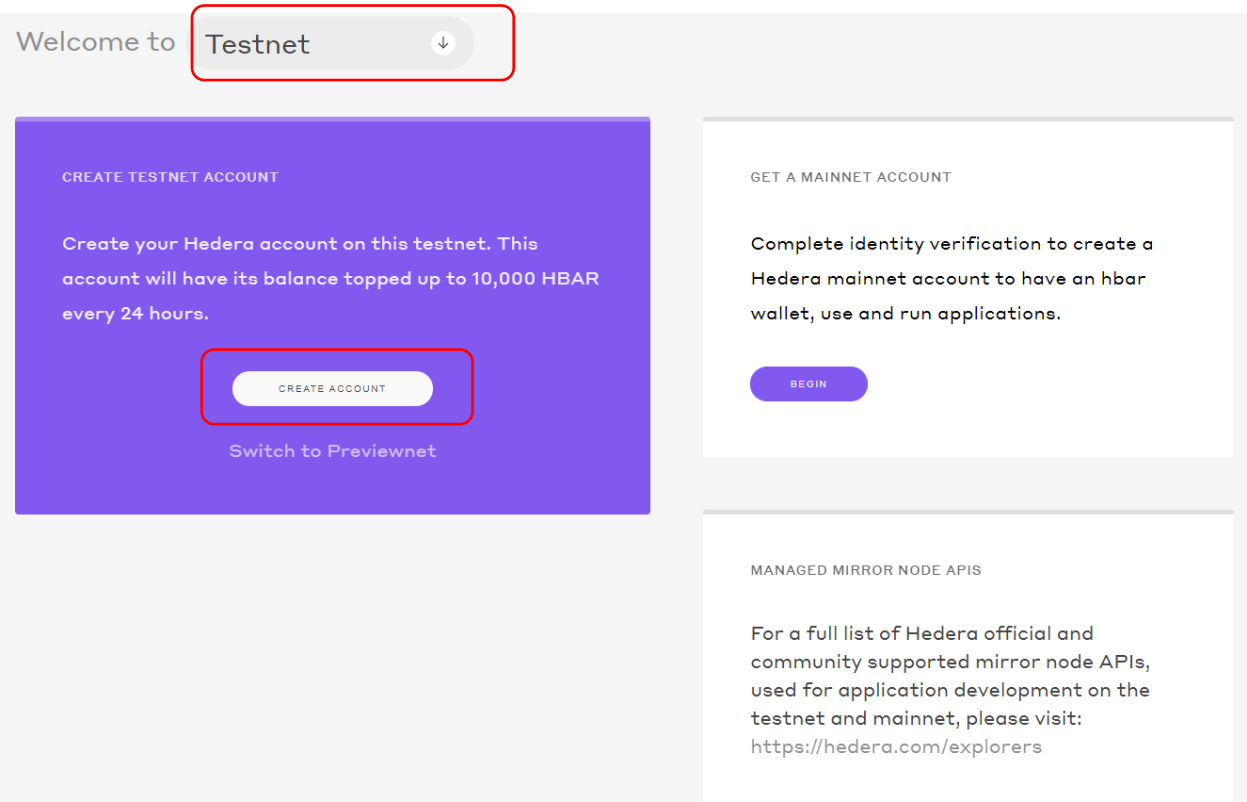
Create a Hedera Training directory, called hederaTraining.

Navigate to https://portal.hedera.com

Complete the account form and click Sign Up



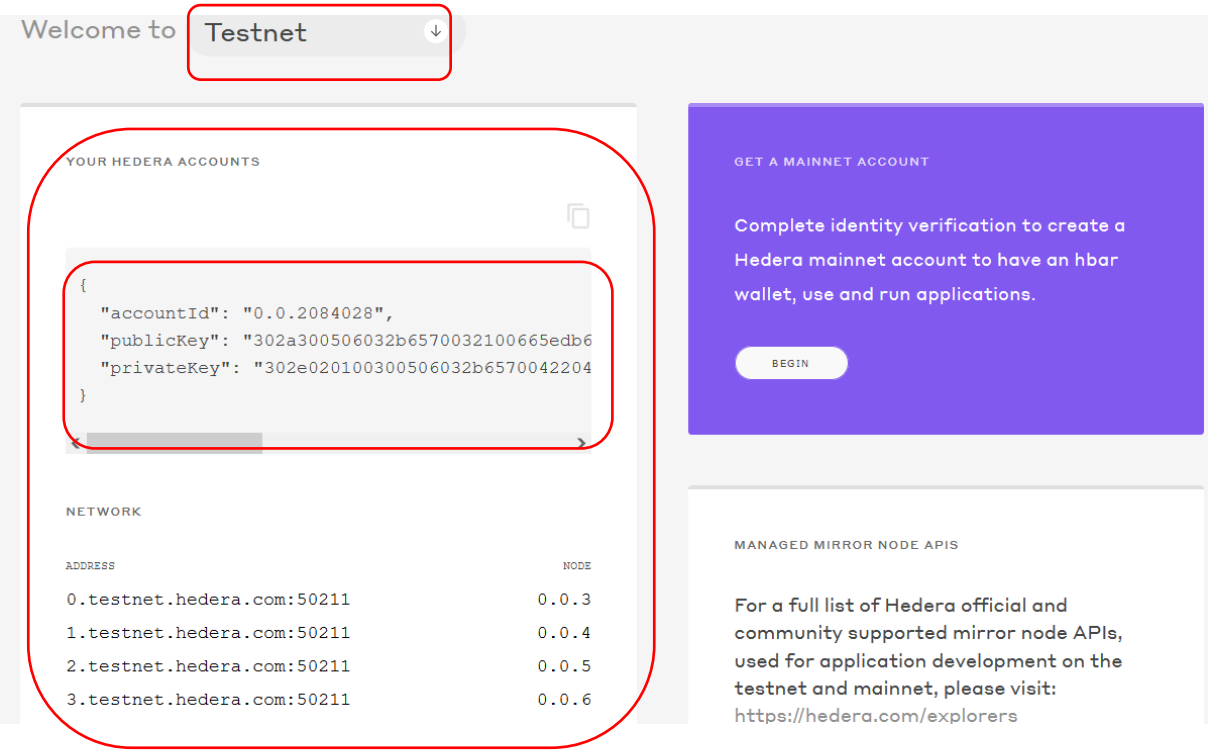
Click **Create Account**. For now, select Tested. A Mainnet account will need to be created. However, for now the **Testnet** will be used.



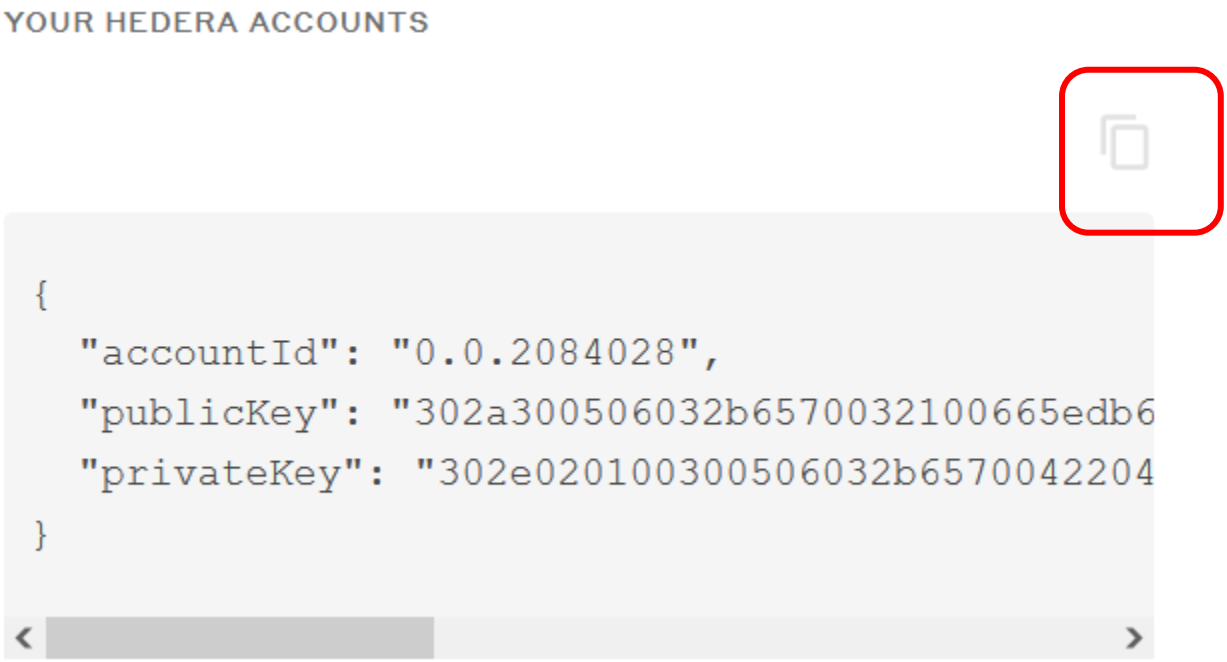
Testnet account credentials will be returned:

- account Id
- public key
- private key

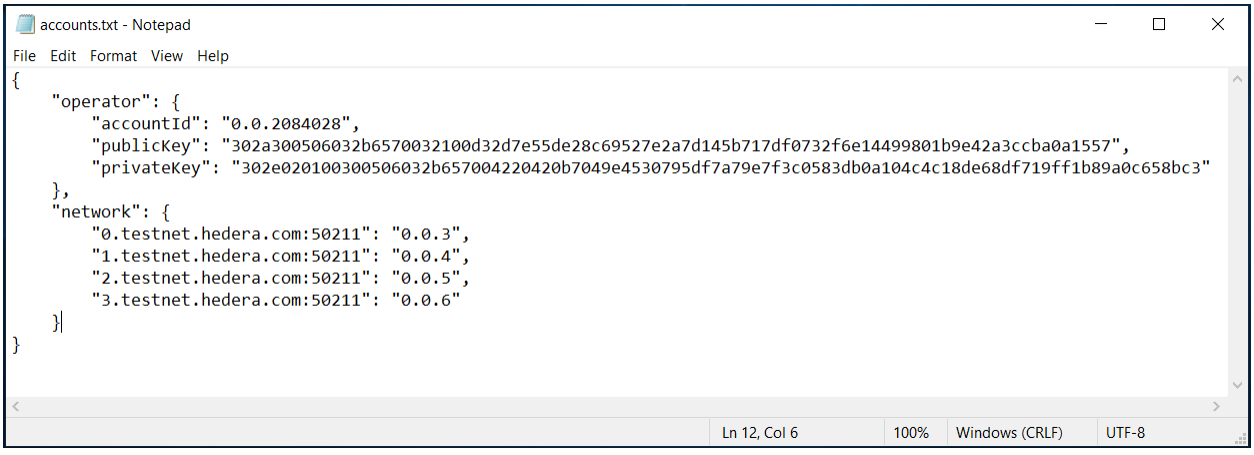
Testnet node addresses will also be returned.



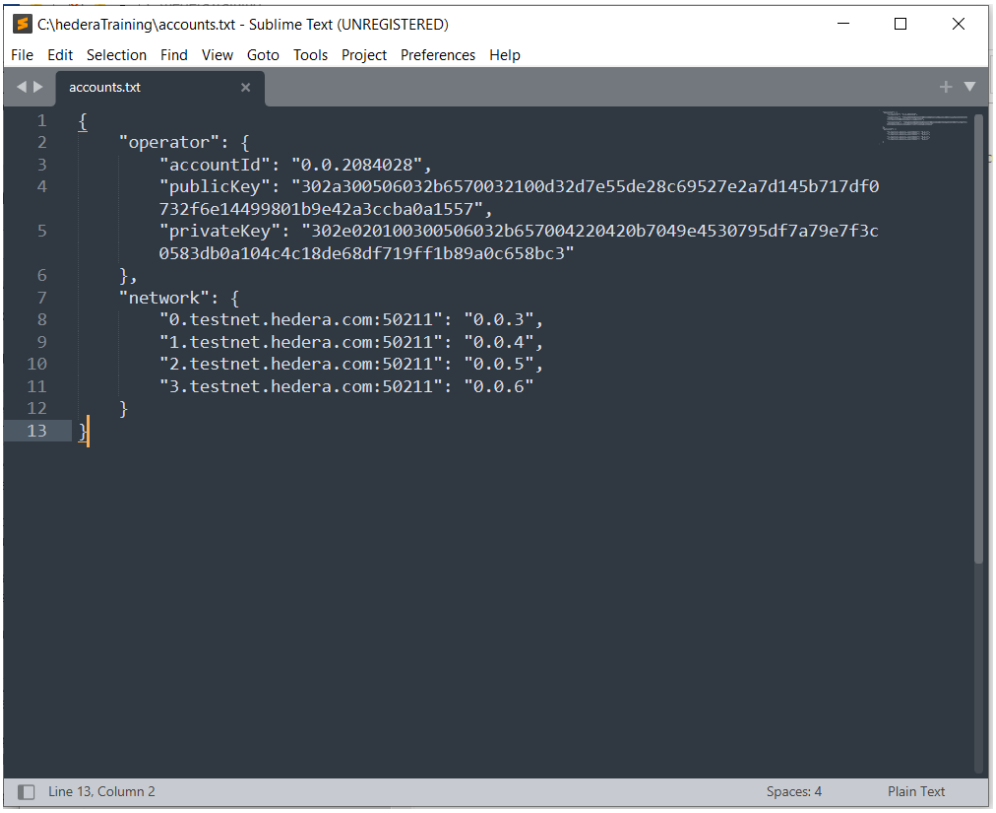
Copy account information to the buffer. Click the copy button

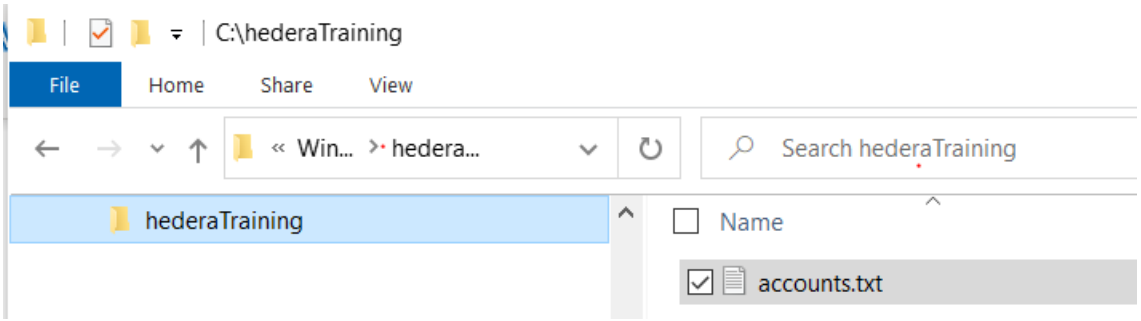


Paste the Testnet account information to a text file and save it in the hederaTraining directory.



There are advantages to keeping your account data in a text file opened with Sublime, Notepad++ or similar high feature text editor application. This will allow quick access to data with many user features.





Install IntelliJ

Step 1.

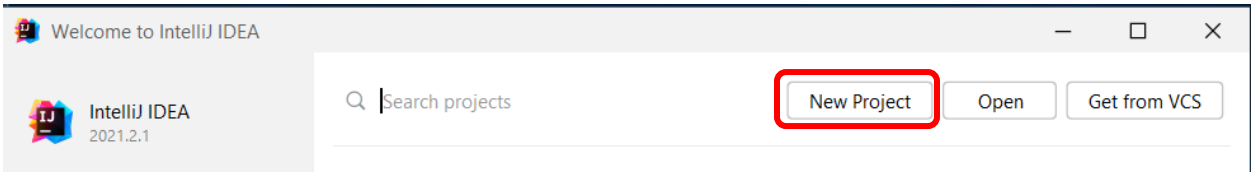
Download IntelliJ: https://www.jetbrains.com/idea/download

Step2

Install Java version 11, https://www.oracle.com/java/technologies. Java Version 11, is the minimal version

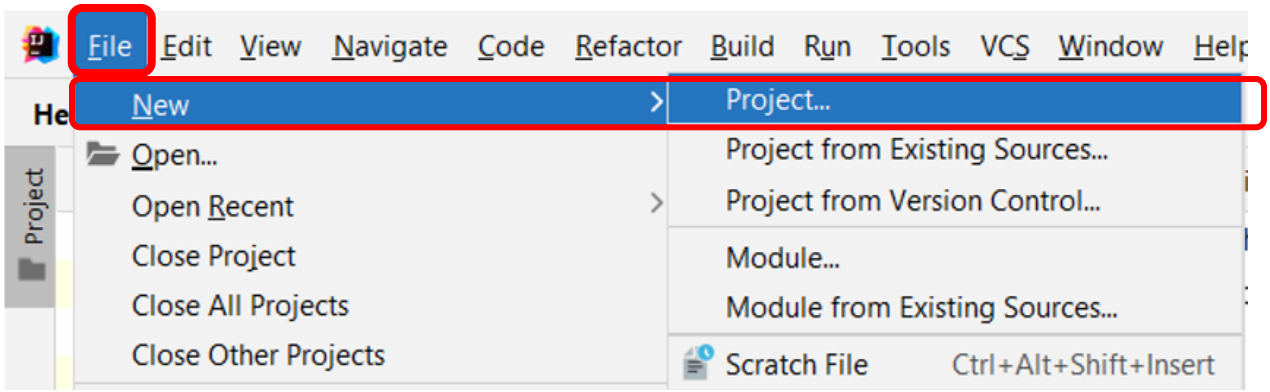
IntelliJ Step 1.

Start ItelliJ and click New Project



or

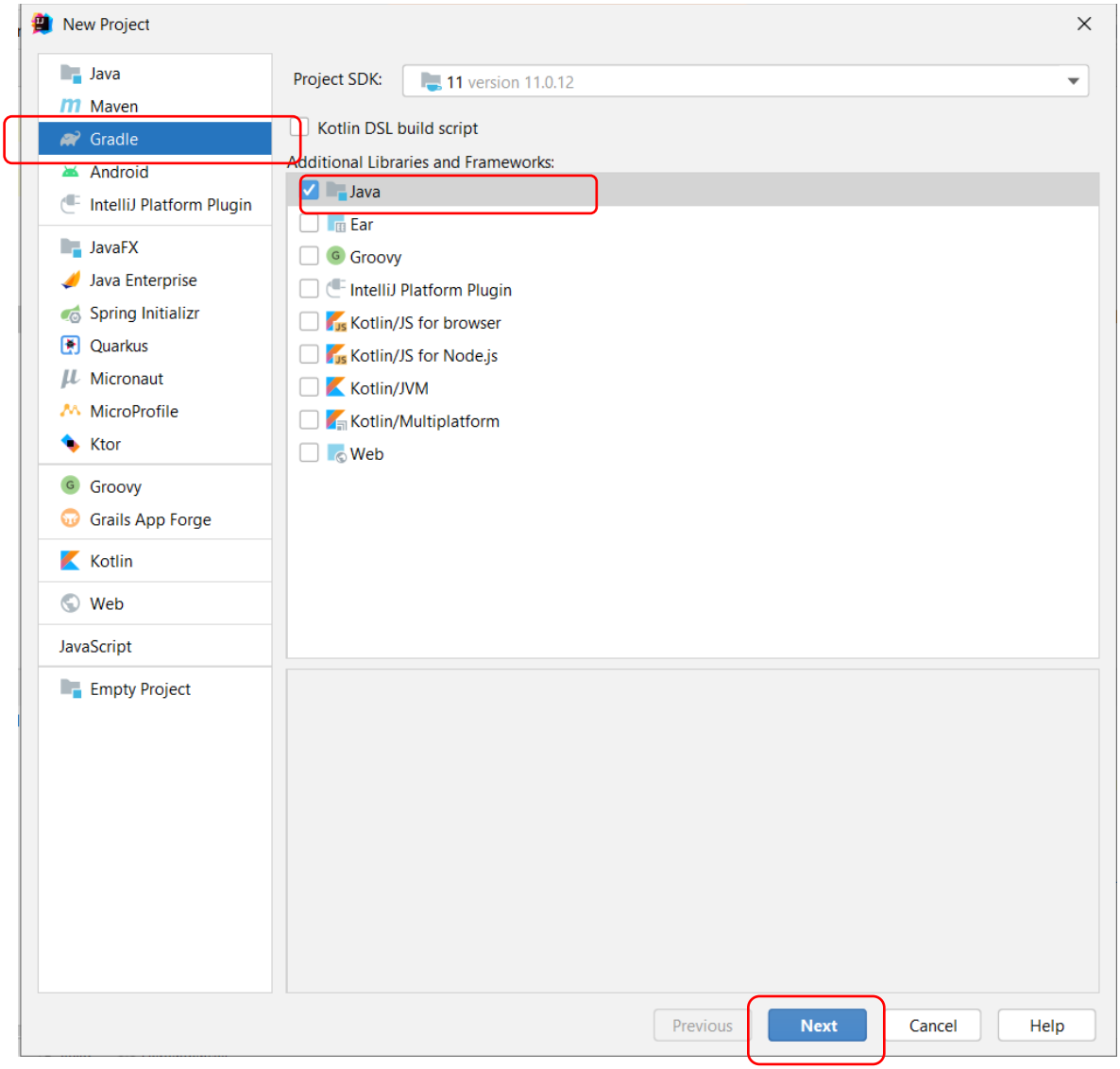
Create a new Project. Click **File** -> **New** -> **Project** …



Select **Gradle**

In **Additional Libraries and Frameworks**, check **Java**

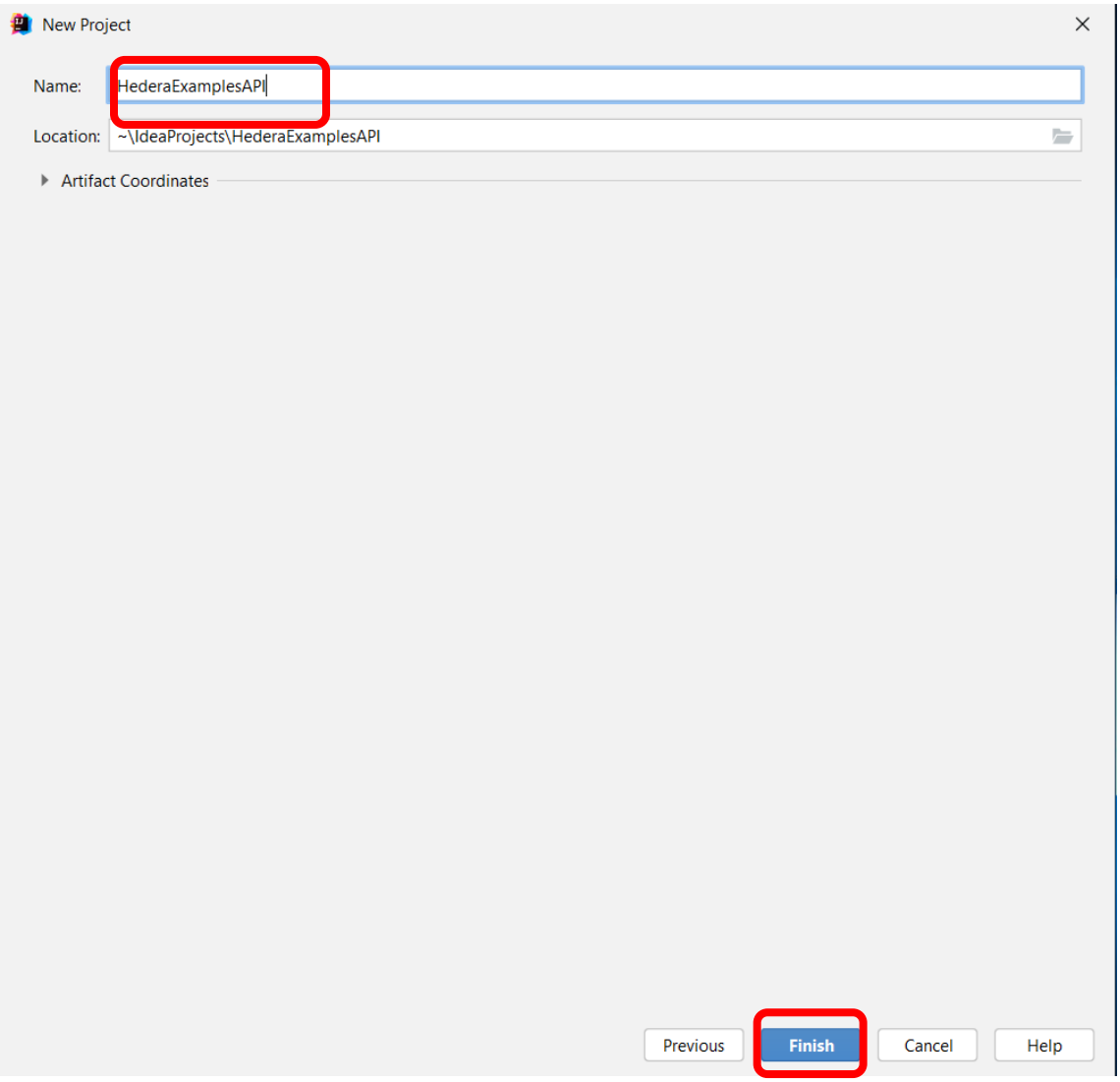
In **Project SDK**, select at least version 11.



Click **Next**

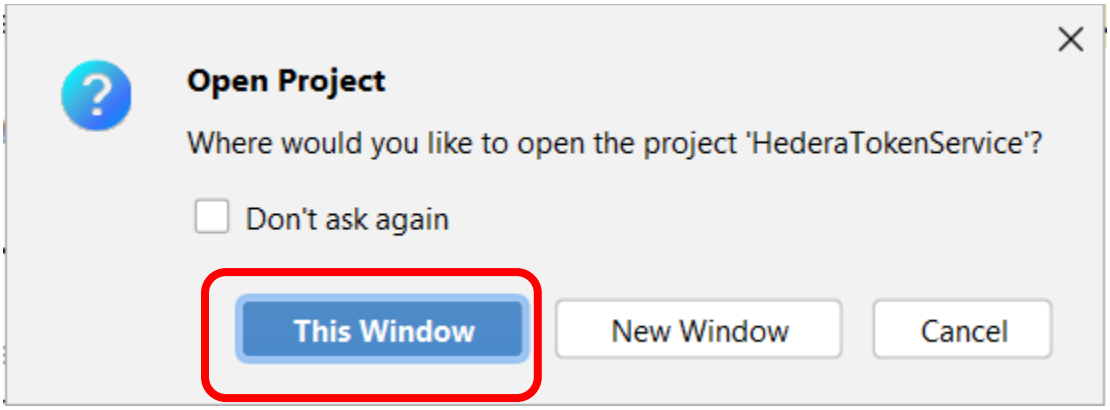
For **Name**, enter **HederaExamplesAPI**

Click **Finish**.

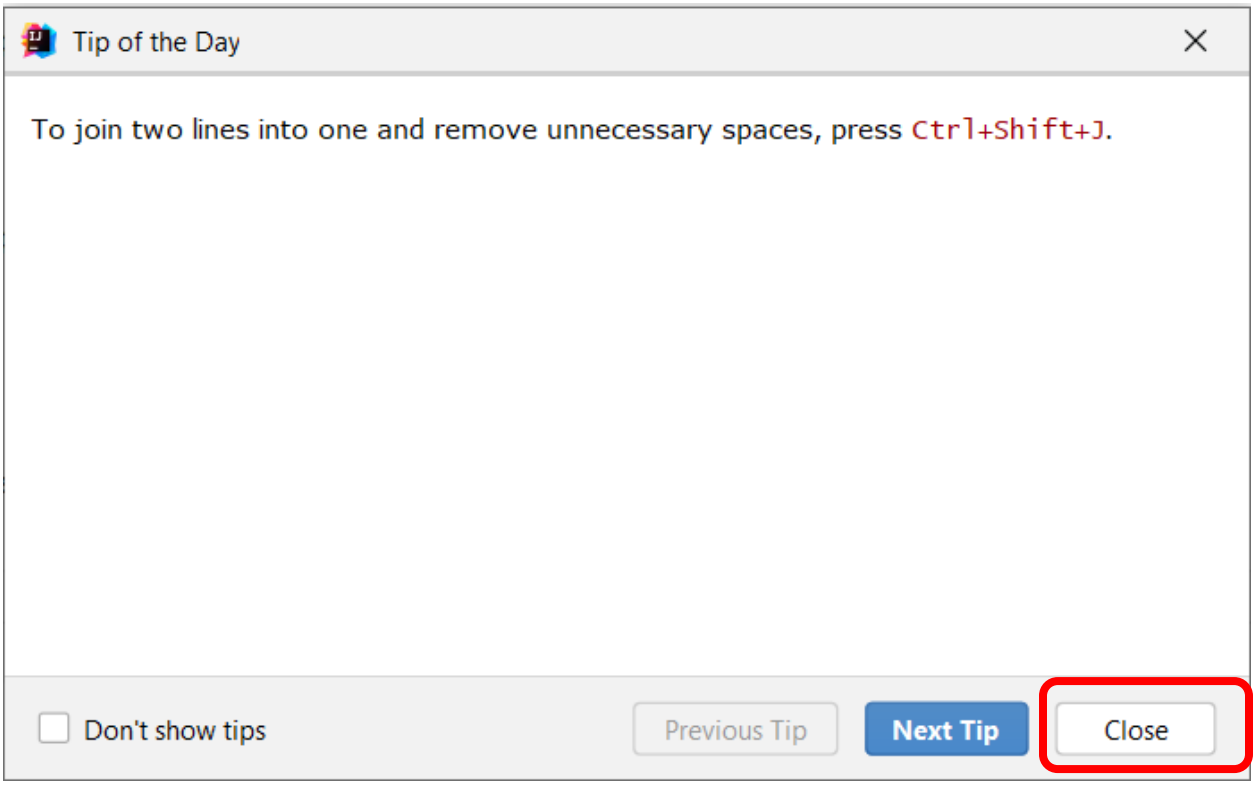


IntelliJ Step 2.

If prompted, for this exercise, select **This Window**

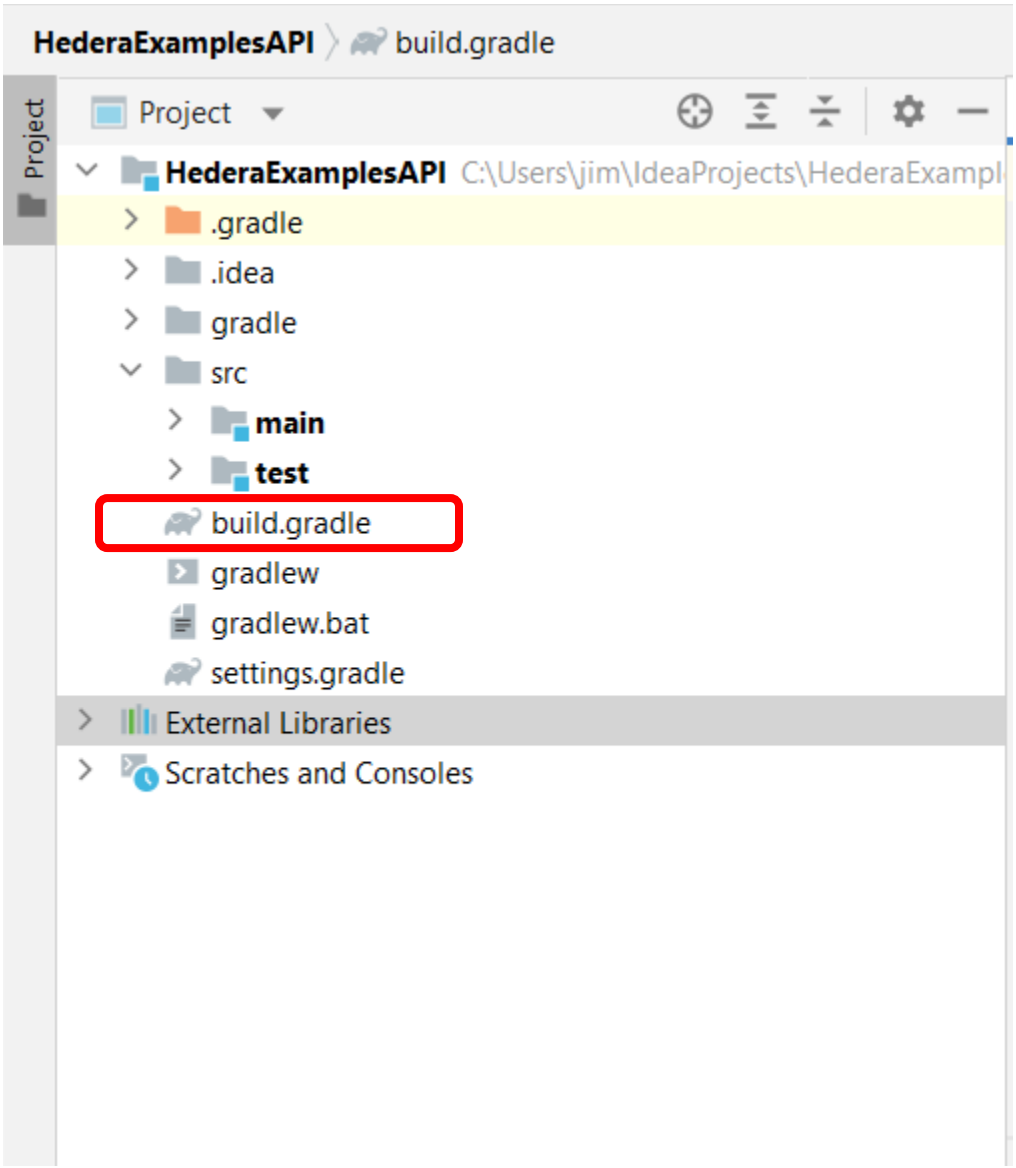


If the **Tip of the Day** opens, click **Close**



IntelliJ Step 3.

Double click on and open the project level build.gradle file.



IntelliJ Step 4.

Copy-Paste all source code from GitHub: build.gradle
https://github.com/theblockchainacademy/Hedera/blob/main/build.gradle

The build.gradle file may not correctly paste into the IntelliJ project. As stated, use the source from GitHub.

build.gradle file:

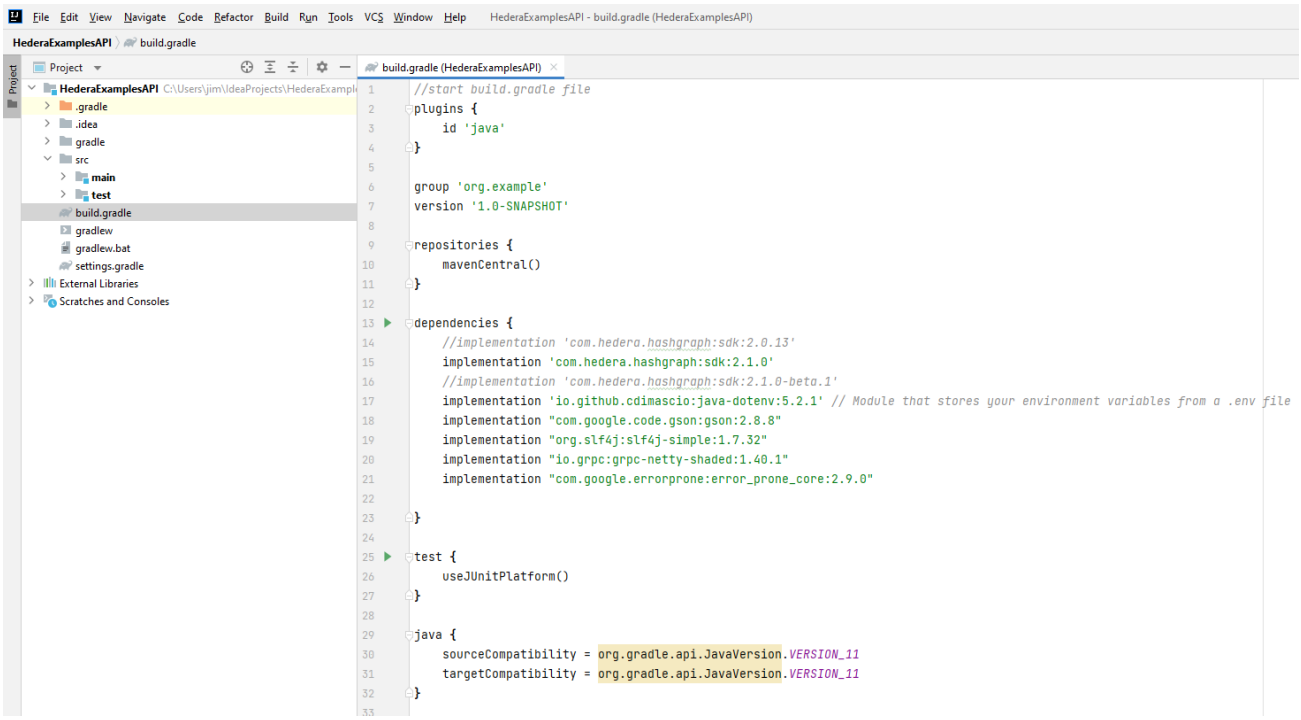
```
//start build.gradle file

plugins {

    id 'java'

}


group 'org.example'

version '1.0-SNAPSHOT'


repositories {

    mavenCentral()

}


dependencies {

    //implementation 'com.hedera.hashgraph:sdk:2.0.13'

    implementation 'com.hedera.hashgraph:sdk:2.1.0'

    //implementation 'com.hedera.hashgraph:sdk:2.1.0-beta.1'

    implementation 'io.github.cdimascio:java-dotenv:5.2.1' // Module
that stores your environment variables from a .env file

    implementation "com.google.code.gson:gson:2.8.8"

    implementation "org.slf4j:slf4j-simple:1.7.32"

    implementation "io.grpc:grpc-netty-shaded:1.40.1"

    implementation "com.google.errorprone:error_prone_core:2.9.0"


}
```

```
test {
    useJUnitPlatform()
}

java {
    sourceCompatibility = org.gradle.api.JavaVersion.VERSION_11
    targetCompatibility = org.gradle.api.JavaVersion.VERSION_11
}

tasks.addRule("Pattern: run<Example>: Runs an example.") { String
taskName ->
    if (taskName.startsWith("run")) {
        task(taskName, type: JavaExec) {
            classpath = sourceSets.main.runtimeClasspath
            main = (taskName - "run") + "Example"
            standardInput(System.in)

            // NOTE: Uncomment to enable trace logs in the SDK during
the examples
            // jvmArgs "-
Dorg.slf4j.simpleLogger.log.com.hedera.hashgraph=trace"
        }
    }
}
//end build.gradle file
```

Paste the dependency text into the project's ***build.gradle*** file using the source from GitHub.

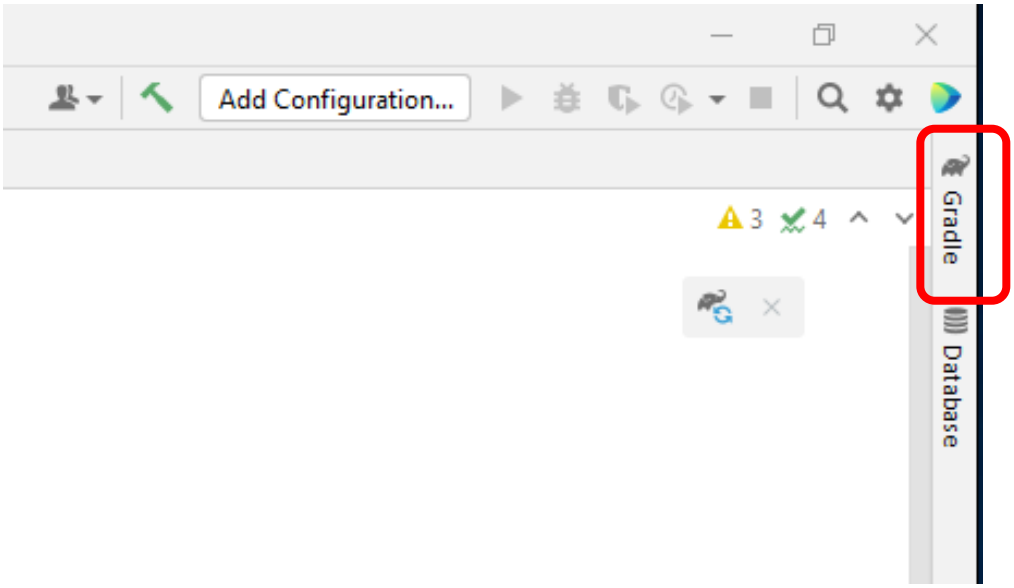
IntelliJ Step 5.

Click **File** -> **Save All** menu, or tap **ctrl + s** to save the project,



Update the dependencies using Gradle

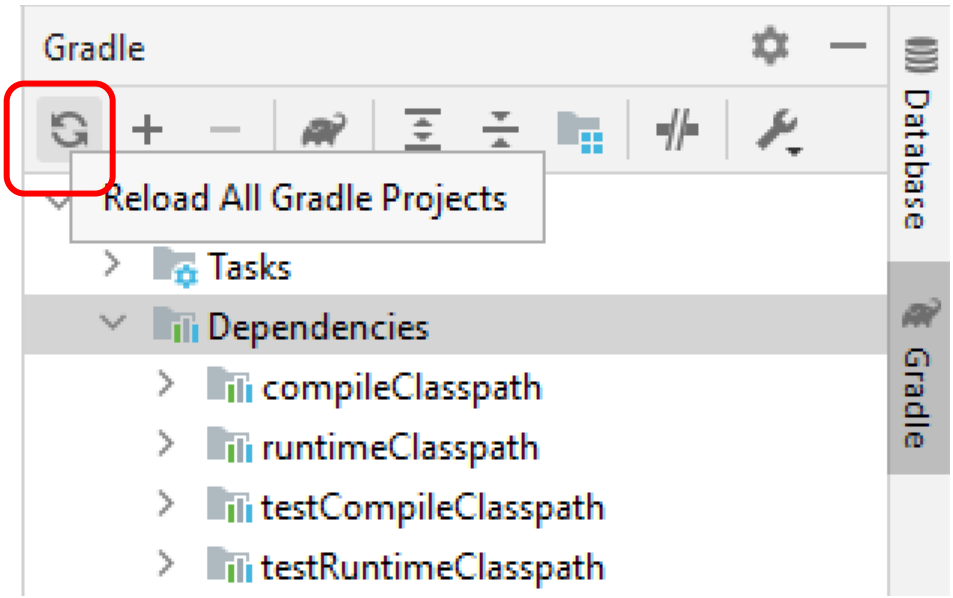
In IntelliJ's upper right corner, look for the **Gradle** tab button and click it.



IntelliJ Step 6.

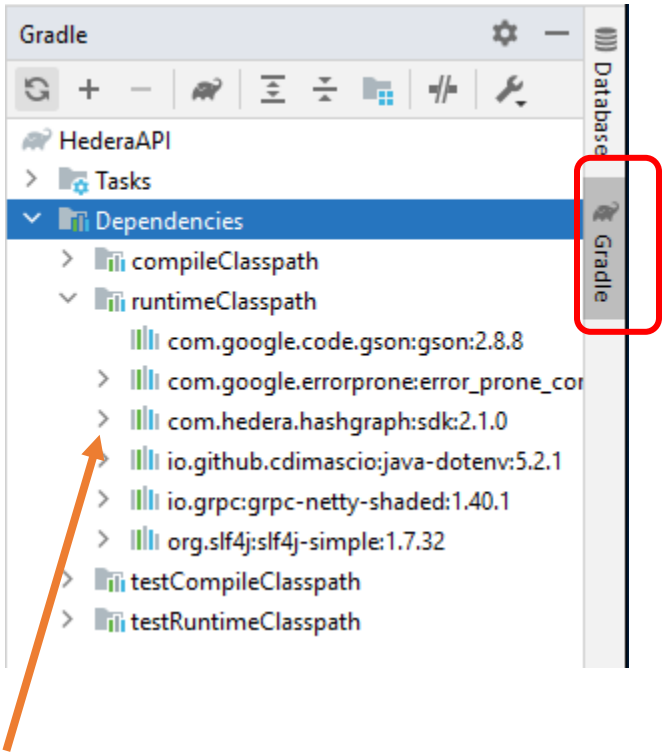
The Gradle tools will open.

Click the Reload All Objects button



Once the updated is completed, you can expand the Dependencies and view the class paths.

Click the Gradle tab button to close the Gradle tools.

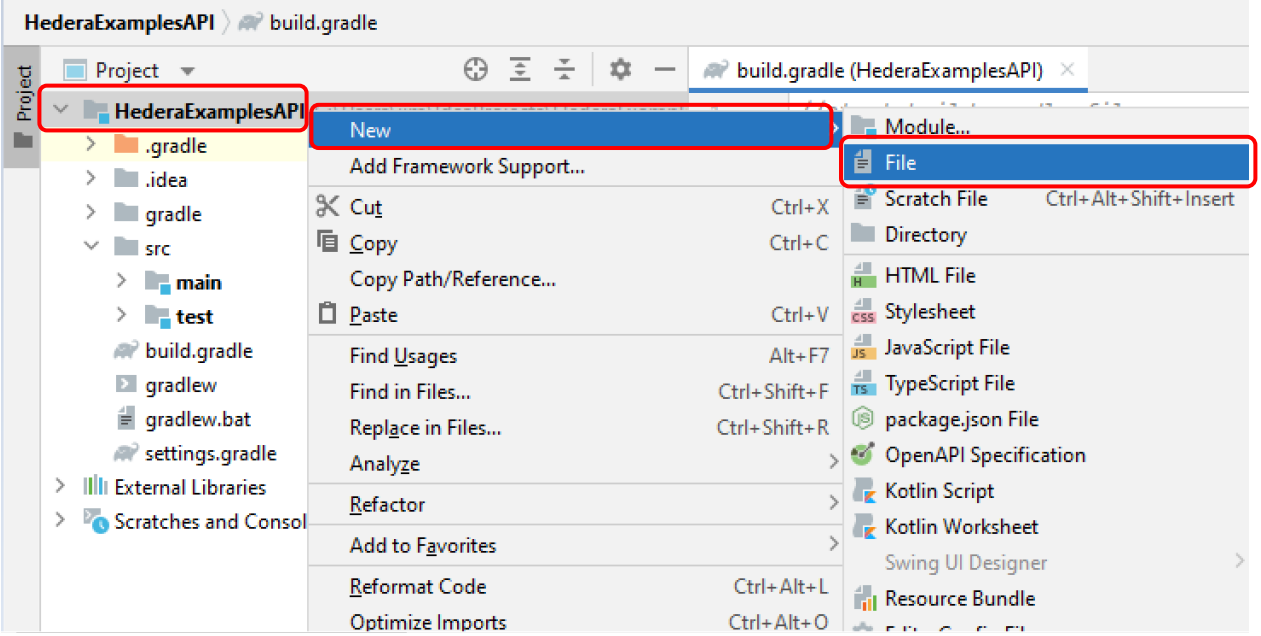


The Hedera SDK 2.1.X will appear in the dependencies, along with other dependencies.

IntelliJ Step 7.

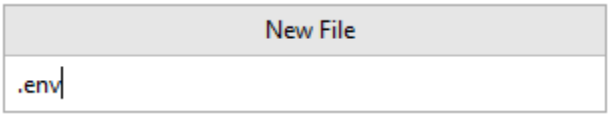
Create an environment file.

Right click on the project level, click **New** -> **File**

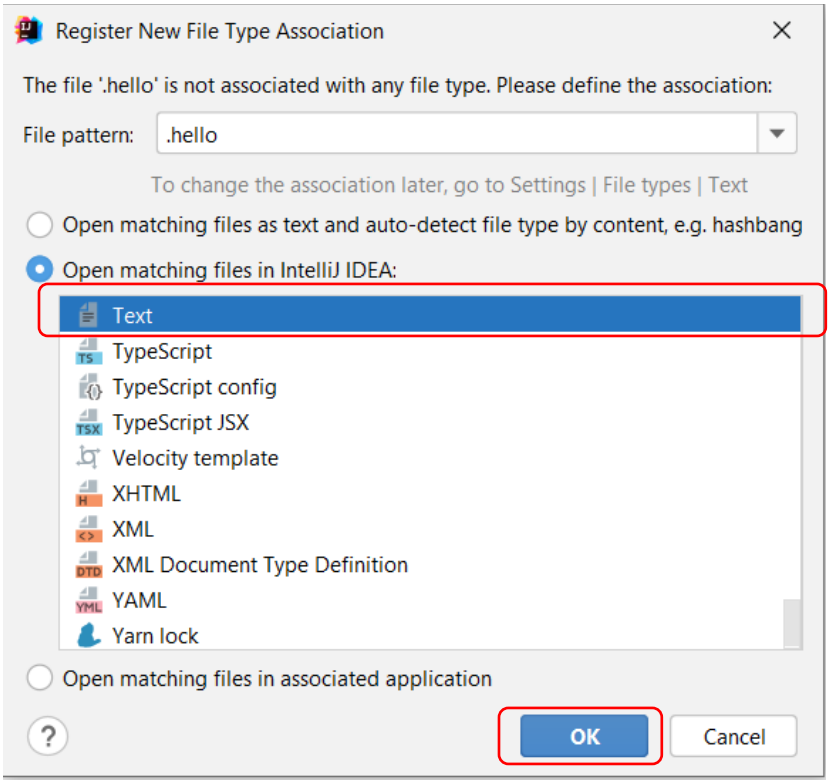


The ***New File*** dialog box will open, enter .env into the ***Name field*** and press the Enter key.

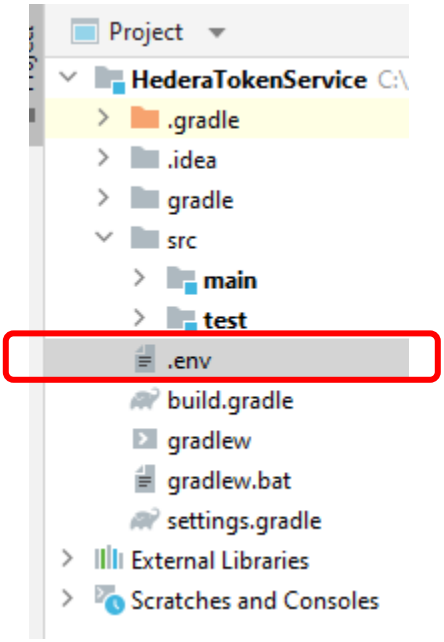
Note: This same .env file is used in the upcoming exercises, 1 -6



If prompted, select **Text** for the file type, and click **OK**.



The **.env** file will open in the editor view and appear listed at the same level, the root level, as the **build.gradle** file in the Project view.



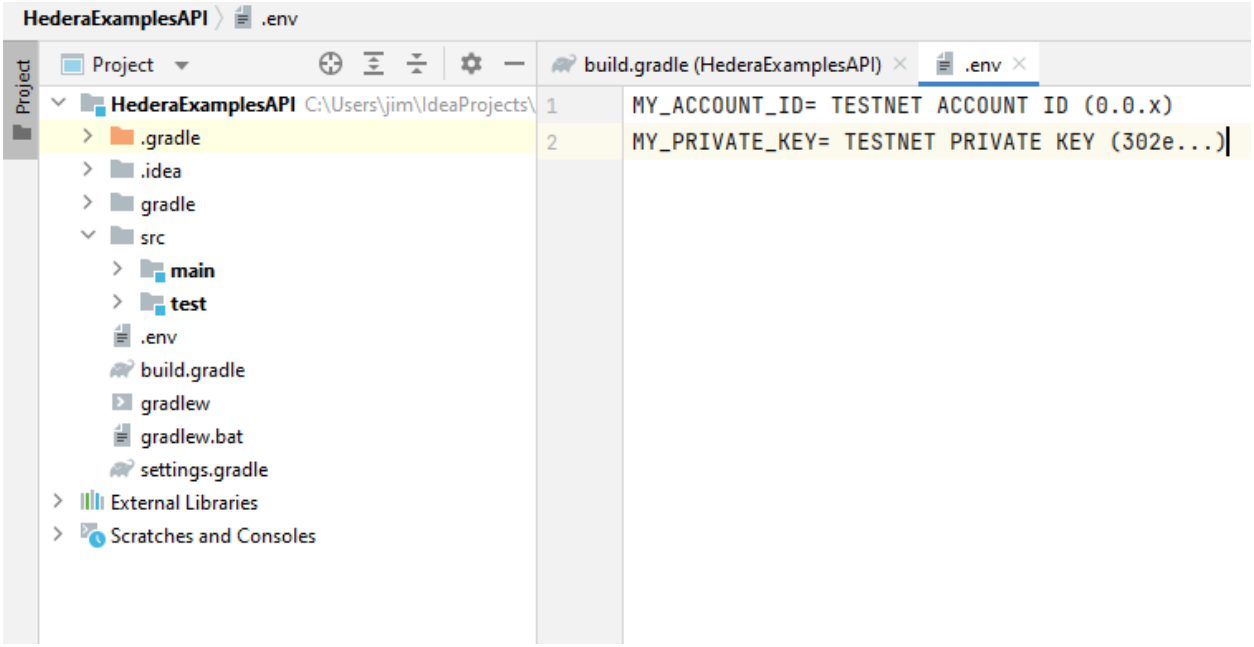
Using Environmental Setup section of the documentation, look for step 2. The template for the .env file is shown. You may copy the text, but it must be updated with your information.

Click on the copy button 

.env

```
MY_ACCOUNT_ID= TESTNET ACCOUNT ID (0.0.x)
MY_PRIVATE_KEY= TESTNET PRIVATE KEY (302e...)
```

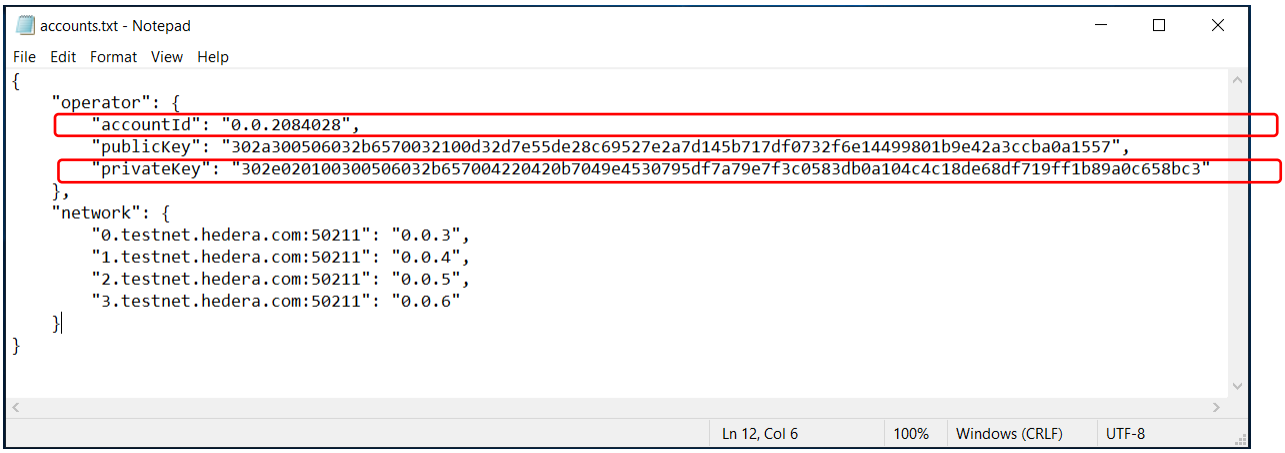
Go back to the Hedera testnet account ID and private key from your Hedera portal profile and enter them in the MY_ACCOUNT_ID and MY_PRIVATE KEY fields.



Use the data in your Testnet account to complete the .env file. This data is saved in the accounts.txt file in your hedera training directory.

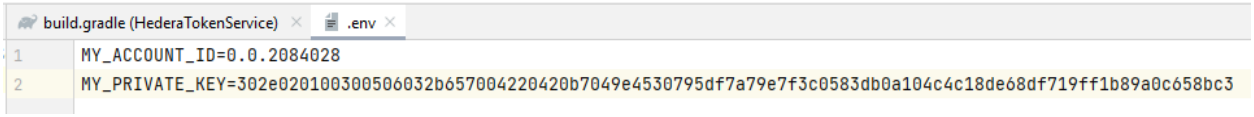
For line “MY_ACCOUNT_ID”, enter your Testnet account ID to the right of the equals, =.

For the line MY_PRIVATE_KEY, enter your Testnet private key to the right of the equals, =.

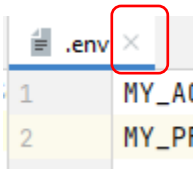


IntelliJ Step 8.

Your .env file will resemble the example below

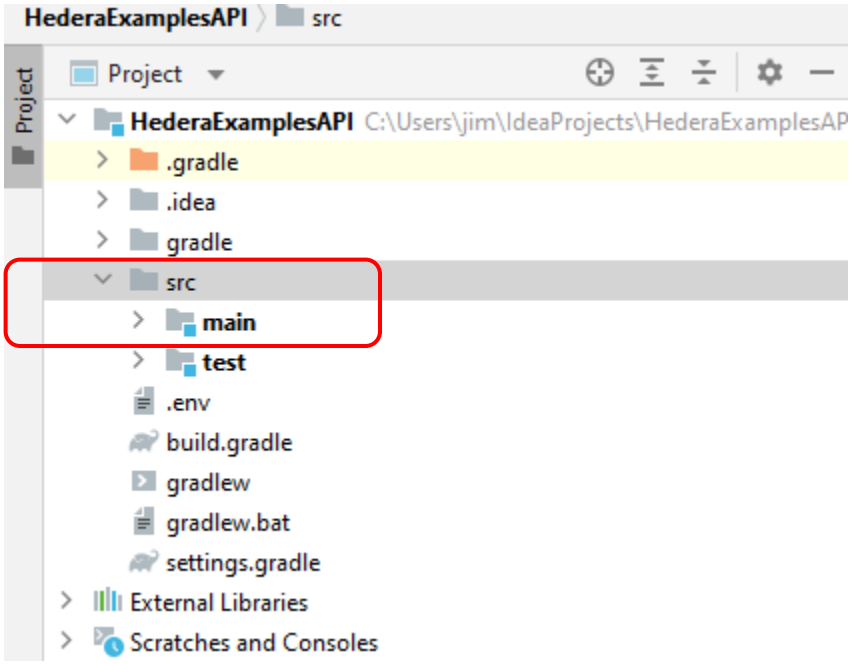


Save the project and close the build. gradle and the .env files. One way to close the file is to click the **X** on the file tabs.



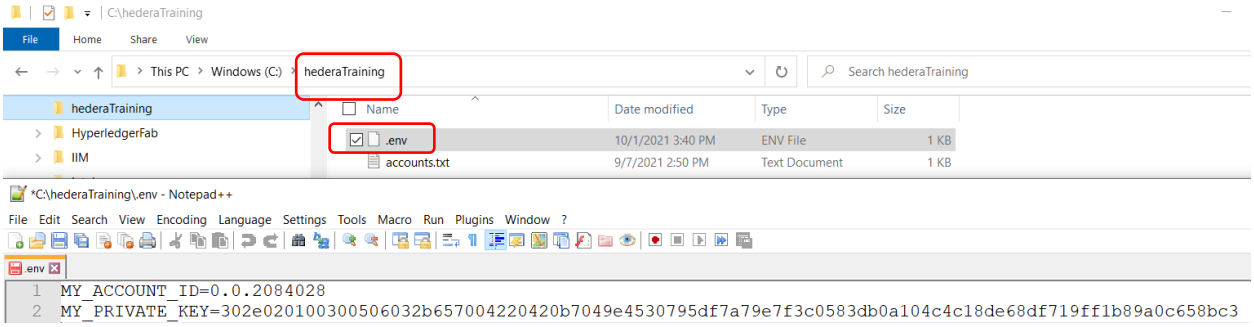
You will be creating Java classes in **src/main** folder.

Expand the projects **src/main** folder.

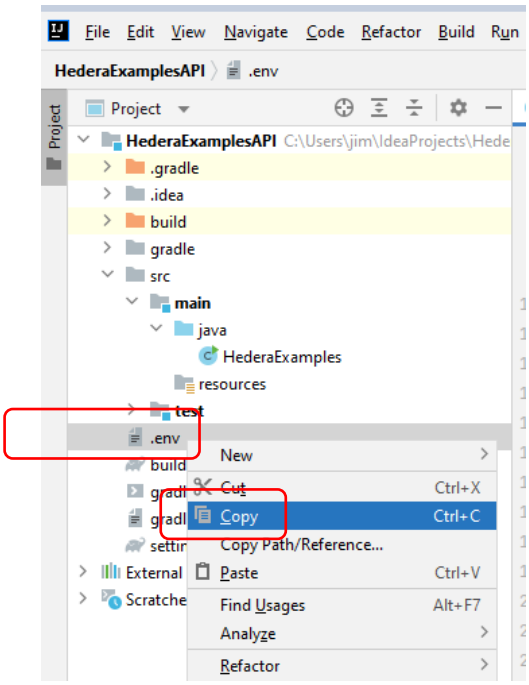


Save the .env file and the data to your herderaTraining folder.

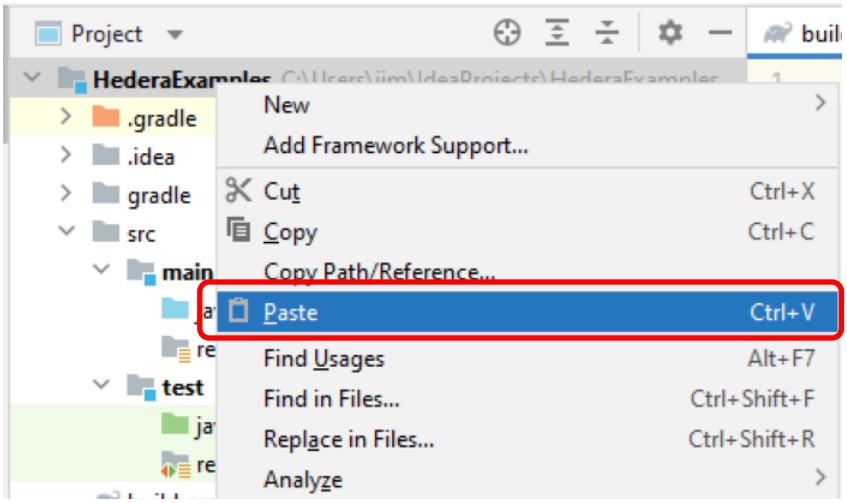
A sample of the .env file and its contents is shown below. The .env files are also in the GitHub reposities.



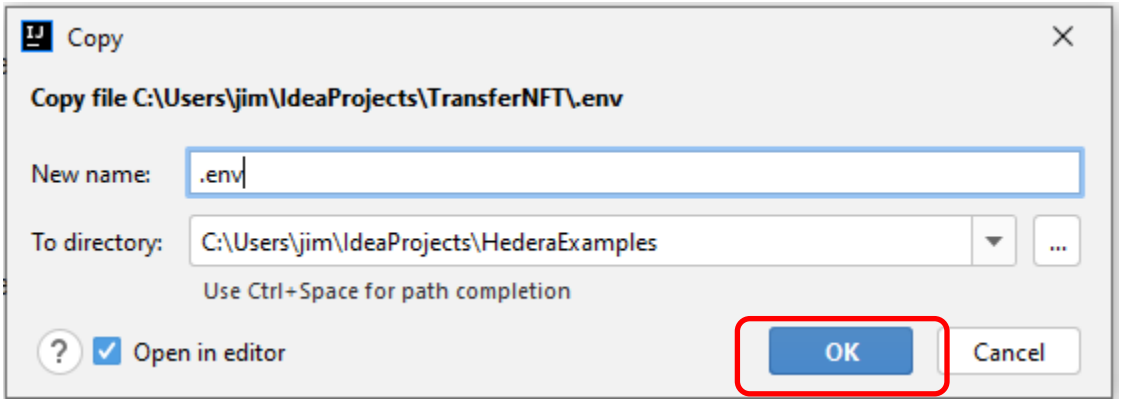
You may copy the .env file from your IntelliJ project to the herderaTraining folder.



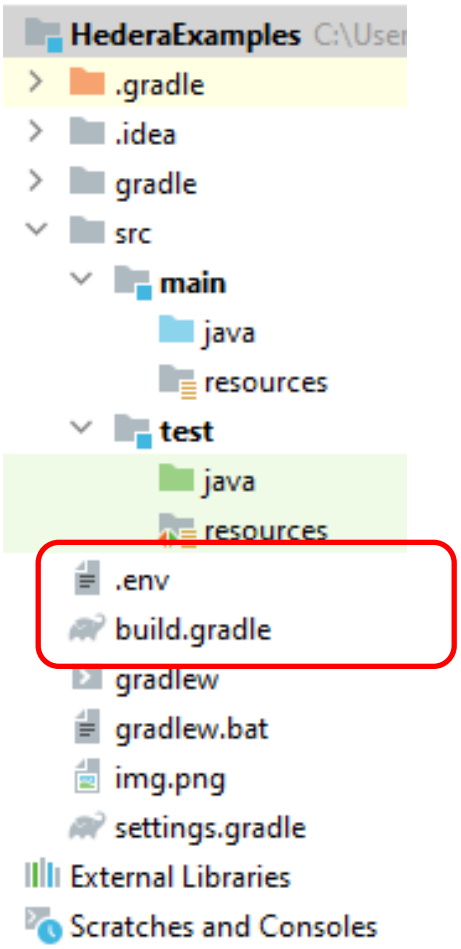
You may copy-paste in the .env files and build.gradle files from the GitHub repositories. Paste these files at the project level as shown in the image.



Click OK to paste the file



The .env and build.gradle files must appear as show below in the image.



Proceed to the exercises. The next step, step 9 will be completed in the exercises themselves.

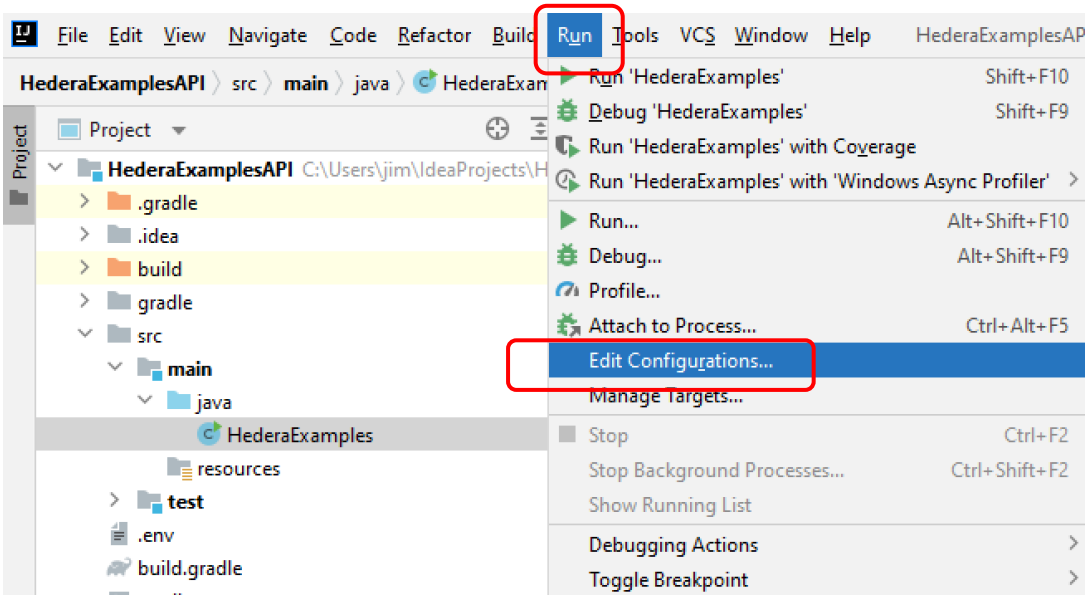
End Environmental Setup

ItelliJ Step 9

Project configurations may need to be entered, or updated.
If so, validate them, if not configure using the steps below.

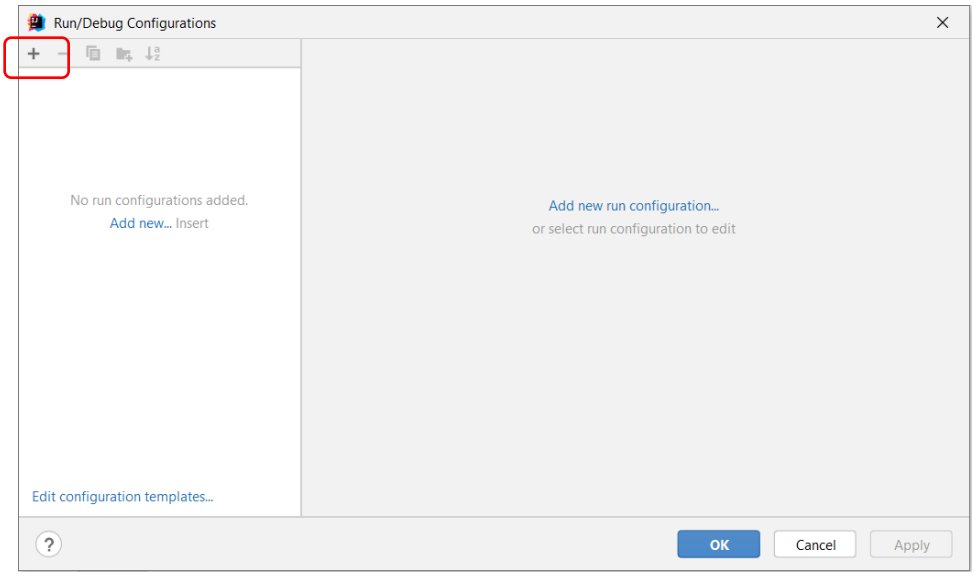
Edit the project configurations

Click **Run** -> **Edit Configurations**



If the Configurations are empty, follow the rest of this step, step 9. NOTE: The configurations may already be completed and entered.

If needed, click the + sign to add configuration information.



Follow the pattern of the HederaExamples class and application shown below

Name: < *Class Name* >, HederaExamples for this example

Run on: Local Machine

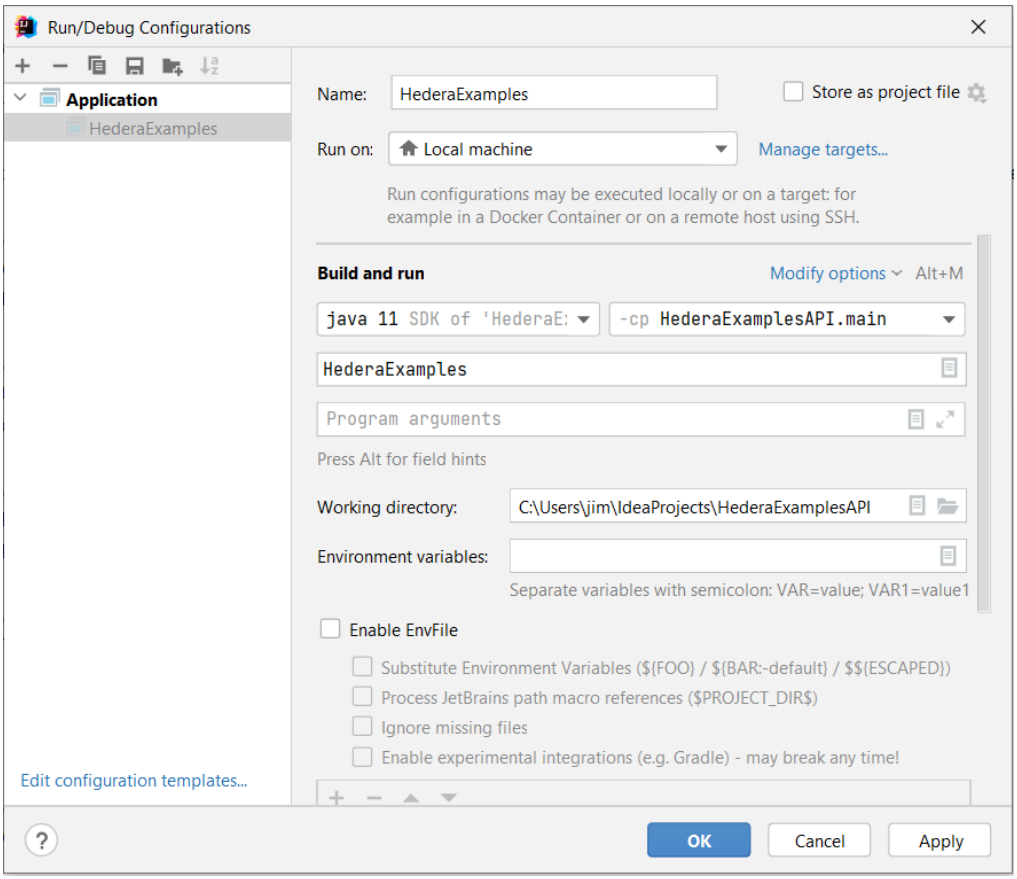
Under Build and Run:

Select Java 11.X (other Java versions are supported) and <Project name>.main,

HederaExamplesAPI.main for this example

In the Main Class field enter: HederaExamples

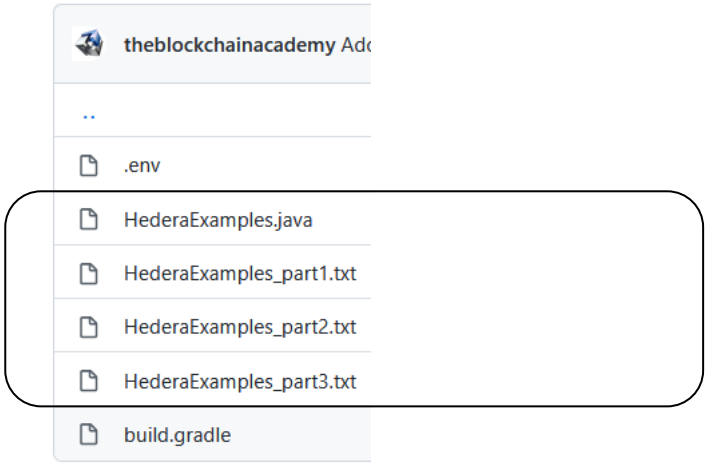
Click Apply and OK



Click **File** -> **Save All**

Interactive Exercise 1, Base Hedera API Examples

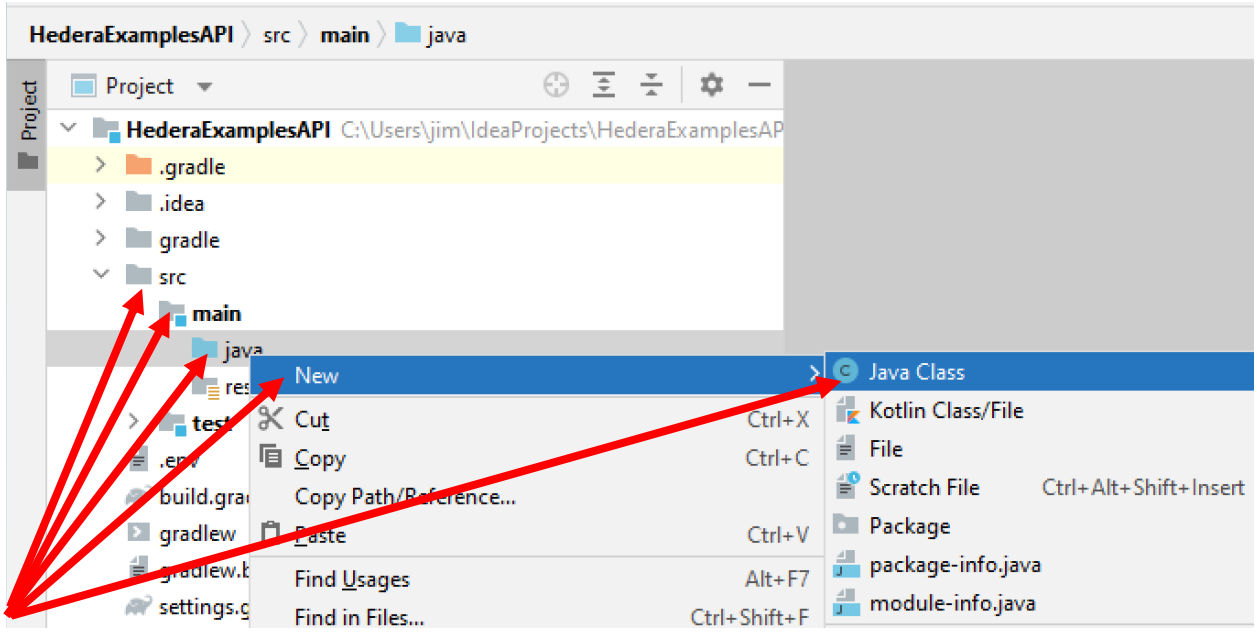
Use code in:
https://github.com/theblockchainacademy/Hedera/tree/main/Hedera_1_ExampleExercise1 and complete the following steps. Iteratively complete the exercise using files part1 to part3.



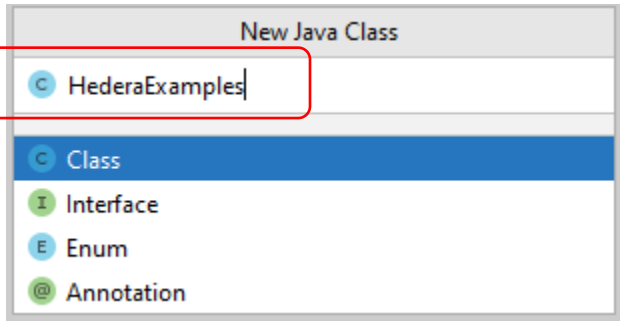
Make sure you have completed steps 1 – 8, Environmental Setup.

Make sure you update the build.gradle file is updated from this GitHub repository. Use your own .env file with your own account ID and private key. An example is included in this GitHub repository.

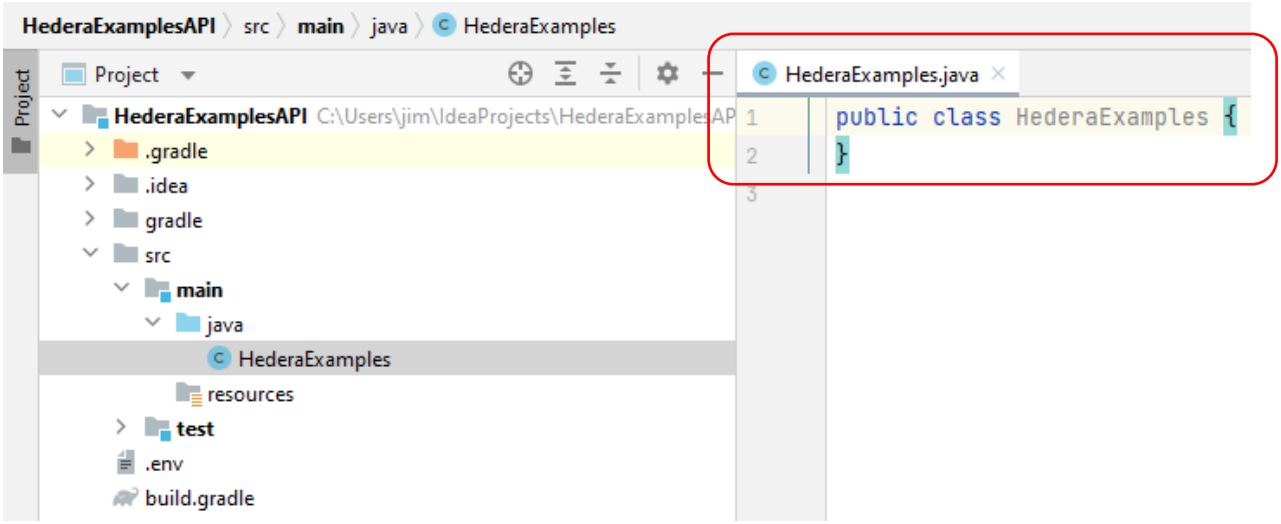
With all Environmental Configurations complete, right click on the java folder, click **New** -> **Java Class**



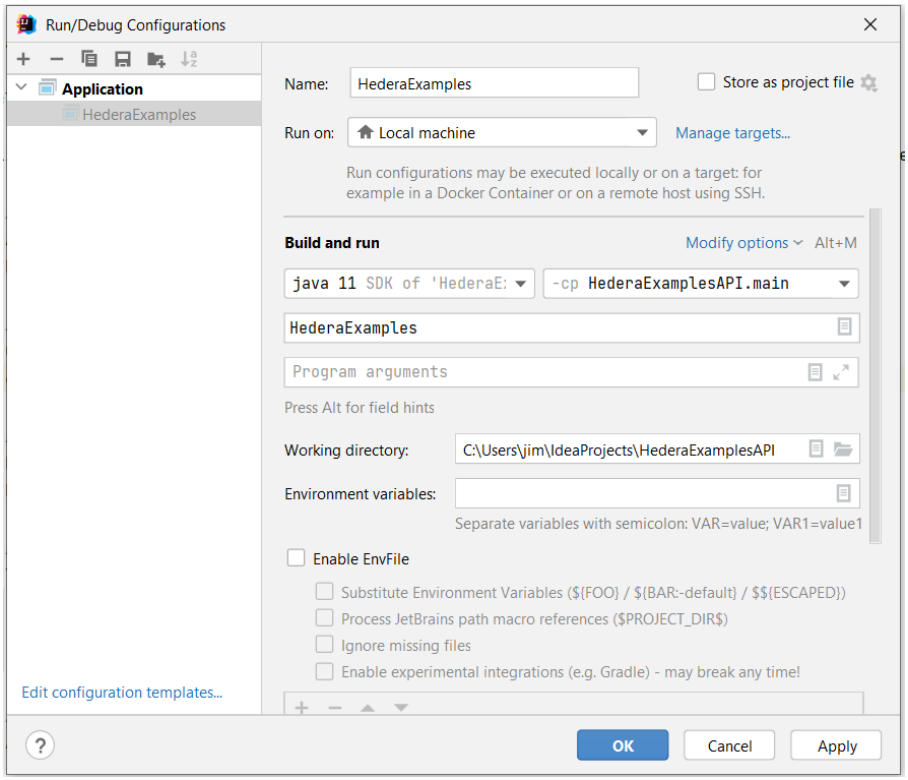
The **New Java Class** dialog box will open. Enter the class name: HederaExamples, and press the Enter key.



The class file opens. The class file is named **HederaExamples.java**



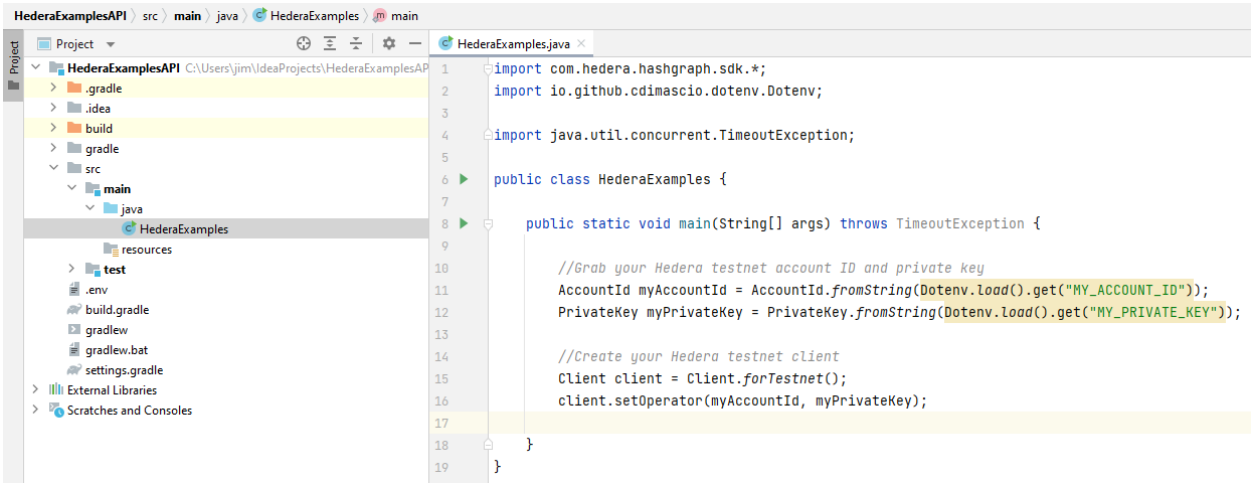
Validate project configurations, see **IntelliJ Step 9**. Project configurations will resemble the image below.



Enter the source code below into the HederaExamples class file. This is file HederaExamples in GitHub: [HederaExamples_part1.txt,](https://github.com/theblockchainacademy/Hedera/blob/main/Hedera_1_ExampleExercise1/HederaExamples_part1.txt) https://github.com/theblockchainacademy/Hedera/blob/main/Hedera_1_ExampleExercise1/HederaExamples_part1.txt

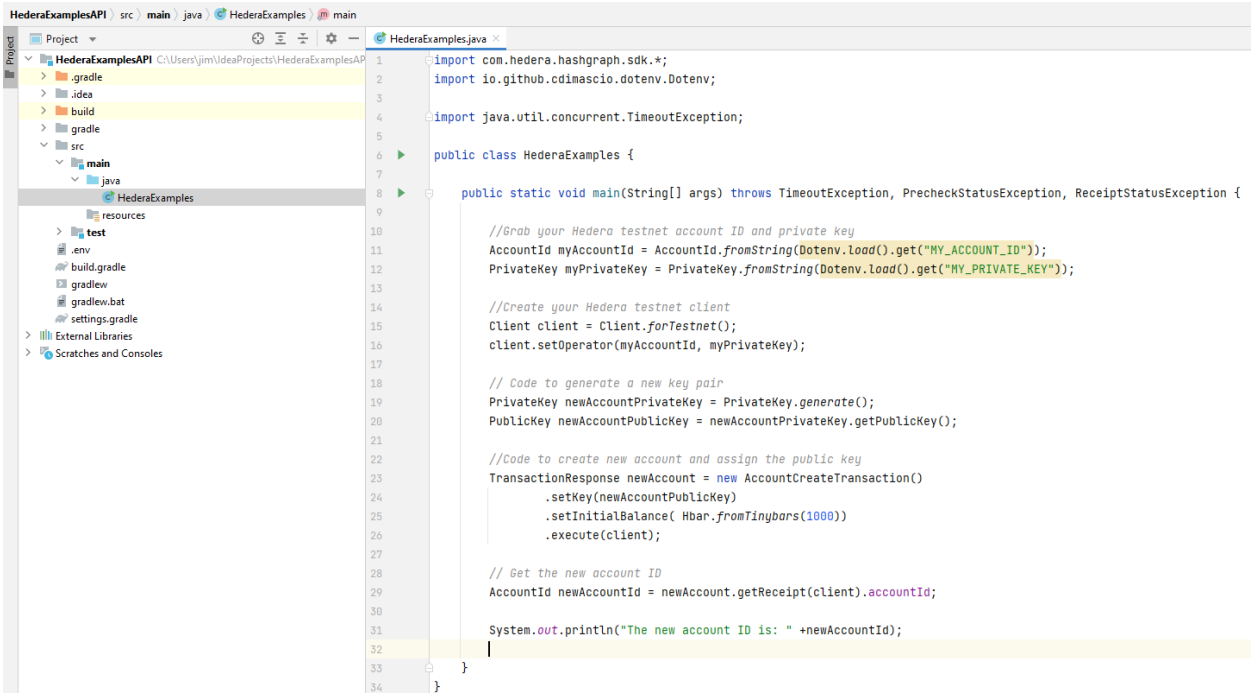
```
import com.hedera.hashgraph.sdk.*;
import io.github.cdimascio.dotenv.Dotenv;

import java.util.concurrent.TimeoutException;

public class HederaExamples {

    public static void main(String[] args) throws TimeoutException,
PrecheckStatusException, ReceiptStatusException {

        //Grab your Hedera testnet account ID and private key
        AccountId myAccountId =
AccountId.fromString(Dotenv.load().get("MY_ACCOUNT_ID"));
        PrivateKey myPrivateKey =
PrivateKey.fromString(Dotenv.load().get("MY_PRIVATE_KEY"));

        //Create your Hedera testnet client
```

```
        Client client = Client.forTestnet();
        client.setOperator(myAccountId, myPrivateKey);


    }
}
```



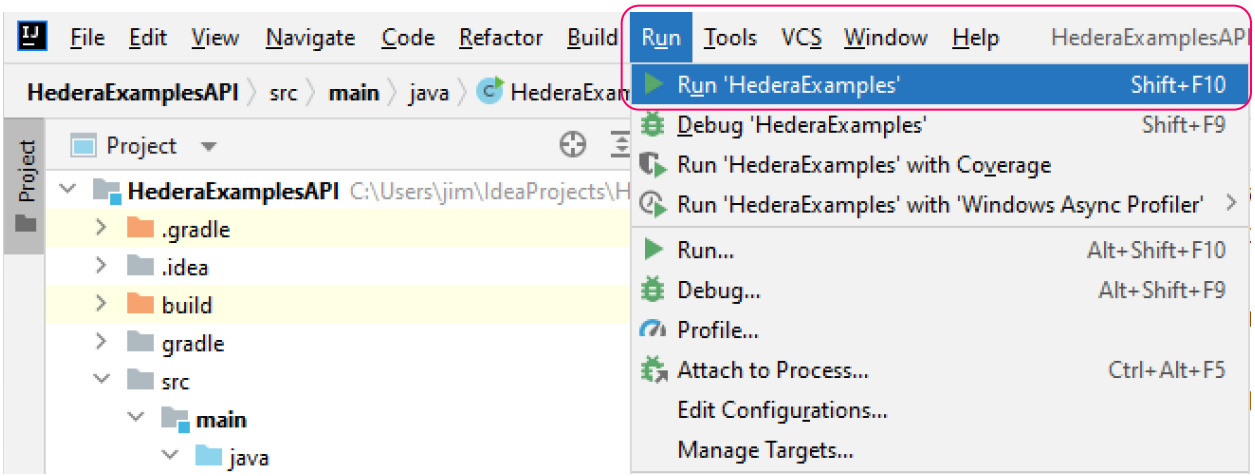
Append the source code below to the class. The source below is in the GitHub file: [HederaExamples_part1.txt](), https://github.com/theblockchainacademy/Hedera/blob/main/Hedera_1_ExampleExercise1/HederaExamples_part1.txt

```
// Code to generate a new key pair
PrivateKey newAccountPrivateKey = PrivateKey.generate();
PublicKey newAccountPublicKey = newAccountPrivateKey.getPublicKey();


//Code to create new account and assign the public key
TransactionResponse newAccount = new AccountCreateTransaction()
    .setKey(newAccountPublicKey)
    .setInitialBalance( Hbar.fromTinybars(1000))
    .execute(client);
// Get the new account ID
    AccountId newAccountId = newAccount.getReceipt(client).accountId;


    System.out.println("The new account ID is: " +newAccountId);
```



```
import com.hedera.hashgraph.sdk.*;
import io.github.cdimascio.dotenv.Dotenv;

import java.util.concurrent.TimeoutException;

public class HederaExamples {

    public static void main(String[] args) throws TimeoutException, PrecheckStatusException, ReceiptStatusException {

        //Grab your Hedera testnet account ID and private key
        AccountId myAccountId = AccountId.fromString(Dotenv.load().get("MY_ACCOUNT_ID"));
        PrivateKey myPrivateKey = PrivateKey.fromString(Dotenv.load().get("MY_PRIVATE_KEY"));

        //Create your Hedera testnet client
        Client client = Client.forTestnet();
        client.setOperator(myAccountId, myPrivateKey);

        // Code to generate a new key pair
        PrivateKey newAccountPrivateKey = PrivateKey.generate();
        PublicKey newAccountPublicKey = newAccountPrivateKey.getPublicKey();

        //Code to create new account and assign the public key
        TransactionResponse newAccount = new AccountCreateTransaction()
                .setKey(newAccountPublicKey)
                .setInitialBalance( Hbar.fromTinybars(1000))
                .execute(client);

        // Get the new account ID
        AccountId newAccountId = newAccount.getReceipt(client).accountId;

        System.out.println("The new account ID is: " +newAccountId);

    }
}
```

Click **Run** -> **Run 'Hedera Examples'**



The Run Console will open. Disregard the `[hedera-sdk] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery` warnings. These are warnings not errors

Look for **the new account ID** output.

You should see the sharded account ID. Your account ID results will be different.

Append the solution source code below to the class. The source below is in the GitHub file: HederaExamples_part2.txt, https://github.com/theblockchainacademy/Hedera/blob/main/Hedera_1_ExampleExercise1/HederaExamples_part2.txt

Add the source code below is the same Java class.

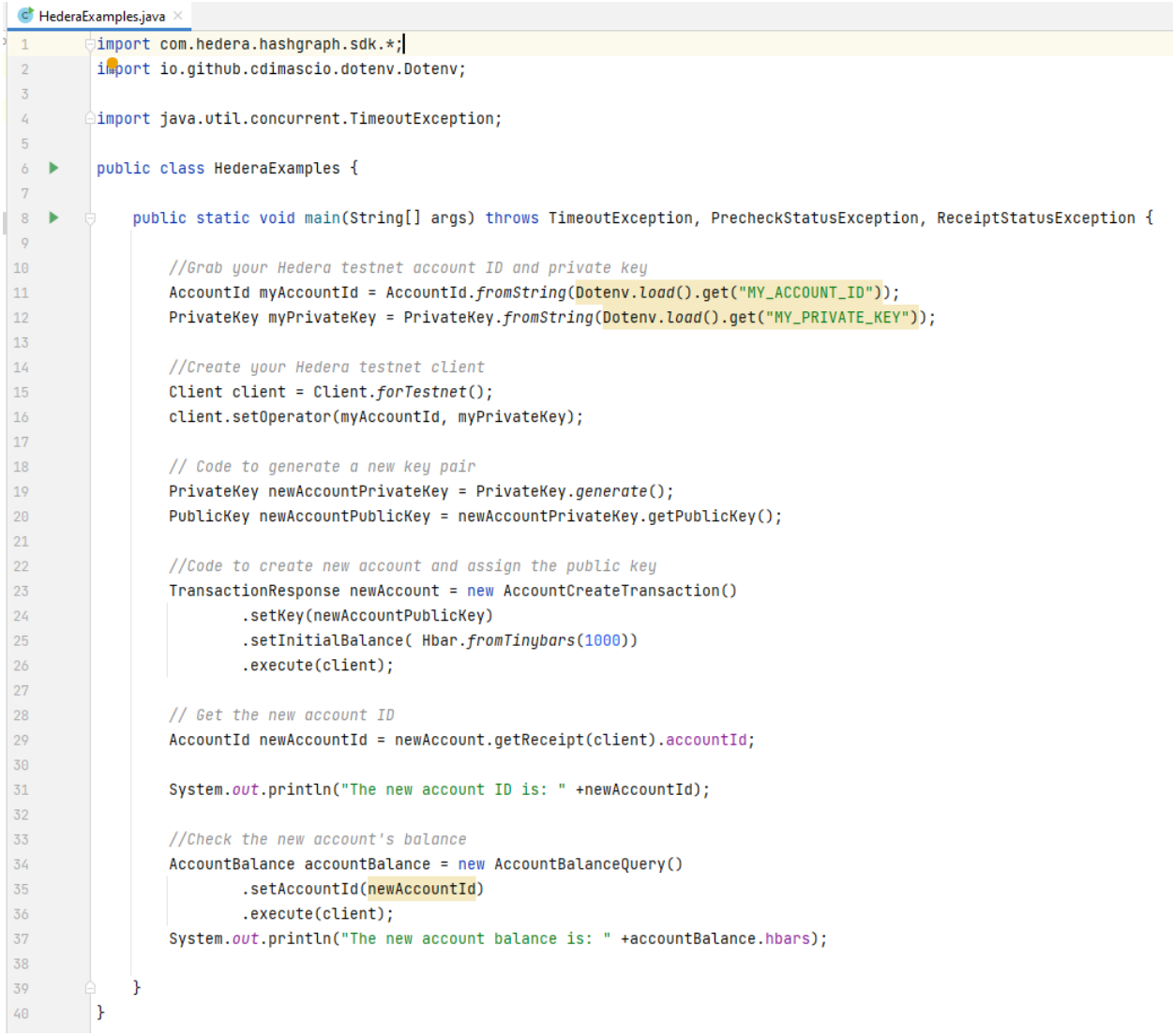
//Check the new account's balance

AccountBalance accountBalance = new AccountBalanceQuery()

.setAccountId(newAccountId)

.execute(client);

System.out.println("The new account balance is: " +accountBalance.hbars);



```java
import com.hedera.hashgraph.sdk.*;
import io.github.cdimascio.dotenv.Dotenv;

import java.util.concurrent.TimeoutException;

public class HederaExamples {

    public static void main(String[] args) throws TimeoutException, PrecheckStatusException, ReceiptStatusException {

        //Grab your Hedera testnet account ID and private key
        AccountId myAccountId = AccountId.fromString(Dotenv.load().get("MY_ACCOUNT_ID"));
        PrivateKey myPrivateKey = PrivateKey.fromString(Dotenv.load().get("MY_PRIVATE_KEY"));

        //Create your Hedera testnet client
        Client client = Client.forTestnet();
        client.setOperator(myAccountId, myPrivateKey);

        // Code to generate a new key pair
        PrivateKey newAccountPrivateKey = PrivateKey.generate();
        PublicKey newAccountPublicKey = newAccountPrivateKey.getPublicKey();

        //Code to create new account and assign the public key
        TransactionResponse newAccount = new AccountCreateTransaction()
                .setKey(newAccountPublicKey)
                .setInitialBalance( Hbar.fromTinybars(1000))
                .execute(client);

        // Get the new account ID
        AccountId newAccountId = newAccount.getReceipt(client).accountId;

        System.out.println("The new account ID is: " +newAccountId);

        //Check the new account's balance
        AccountBalance accountBalance = new AccountBalanceQuery()
                .setAccountId(newAccountId)
                .execute(client);
        System.out.println("The new account balance is: " +accountBalance.hbars);

    }
}
```

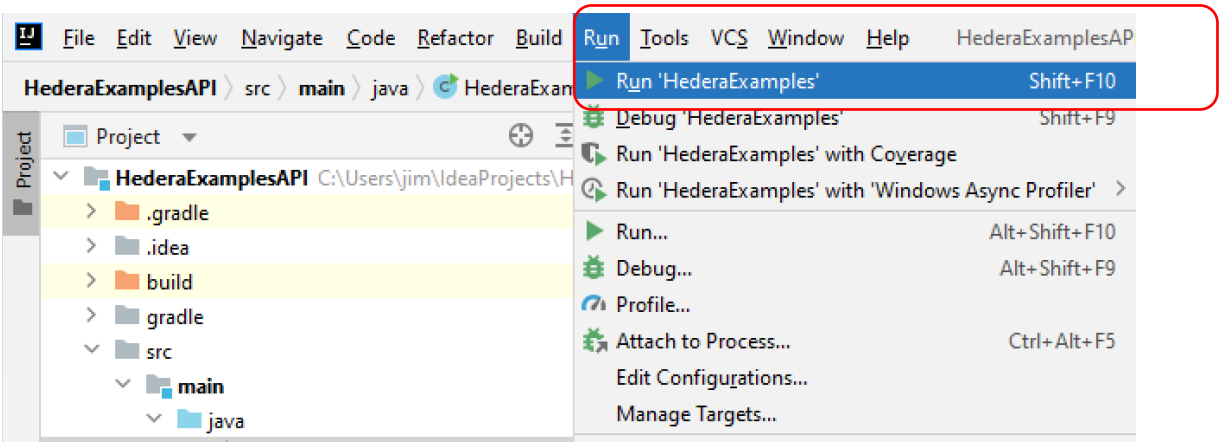
Append the solution source code below to the class. The source below is in the GitHub file: [HederaExamples_part3.txt](), https://github.com/theblockchainacademy/Hedera/blob/main/Hedera_1_ExampleExercise1/HederaExamples_part3.txt

The whole application source code is below, also on GitHub repository file: [HederaExamples.java](), https://github.com/theblockchainacademy/Hedera/blob/main/Hedera_1_ExampleExercise1/HederaExamples.java

```java
import com.hedera.hashgraph.sdk.*;
import io.github.cdimascio.dotenv.Dotenv;

import java.util.concurrent.TimeoutException;

public class HederaExamples {
```

```
    public static void main(String[] args) throws TimeoutException,
PrecheckStatusException, ReceiptStatusException {

        //Grab your Hedera testnet account ID and private key
        AccountId myAccountId =
AccountId.fromString(Dotenv.load().get("MY_ACCOUNT_ID"));
        PrivateKey myPrivateKey =
PrivateKey.fromString(Dotenv.load().get("MY_PRIVATE_KEY"));

        //Create your Hedera testnet client
        Client client = Client.forTestnet();
        client.setOperator(myAccountId, myPrivateKey);

        // Code to generate a new key pair
        PrivateKey newAccountPrivateKey = PrivateKey.generate();
        PublicKey newAccountPublicKey = newAccountPrivateKey.getPublicKey();

        //Code to create new account and assign the public key
        TransactionResponse newAccount = new AccountCreateTransaction()
                .setKey(newAccountPublicKey)
                .setInitialBalance( Hbar.fromTinybars(1000))
                .execute(client);

        // Get the new account ID
        AccountId newAccountId = newAccount.getReceipt(client).accountId;

        System.out.println("The new account ID is: " +newAccountId);

        //Check the new account's balance
        AccountBalance accountBalance = new AccountBalanceQuery()
                .setAccountId(newAccountId)
                .execute(client);
        System.out.println("The new account balance is: "
+accountBalance.hbars);

    }
}
```

Click **Run** -> **Run 'Hedera Examples'**



The Run Console will open. Disregard the [hedera-sdk-7] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery warnings. These are warnings and not errors.

Look for the new account ID output, and the account balance

You should see the sharded account ID, and the account balance.

The fonts for hbars, ℏ, cannot be rendered in the console. It returns “t?” for hbars.

```
4:39:27 PM: Executing task ':HederaExamples.main()'...

> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE

> Task :HederaExamples.main()
[hedera-sdk-7] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.6 in 250 ms after failure during attempt #1: OK
[hedera-sdk-2] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.6 in 500 ms after failure during attempt #2: OK
[hedera-sdk-6] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.6 in 1000 ms after failure during attempt #3: OK
The new account ID is: 0.0.2700983
The new account balance is: 1000 t?

Deprecated Gradle features were used in this build, making it incompatible with Gradle 8.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

See https://docs.gradle.org/7.1/userguide/command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 5s
2 actionable tasks: 1 executed, 1 up-to-date
4:39:33 PM: Task execution finished ':HederaExamples.main()'.
```

Interactive Exercise 2

Describe how Hedera Hashgraph Consensus resolves issues that other blockchains struggle with.

1. Describe: Proof-of-Stake consensus.

2. Describe Asynchronous Byzantine Fault Tolerance. What is the significance of the mathematical proof?

3. How about decentralization?

Interactive Exercise 3

How does Hedera Hashgraph Governance Model benefit users.

The Governing Council advices on what?

The Governing Council comes from a representative set of industries.

Do they make decisions?

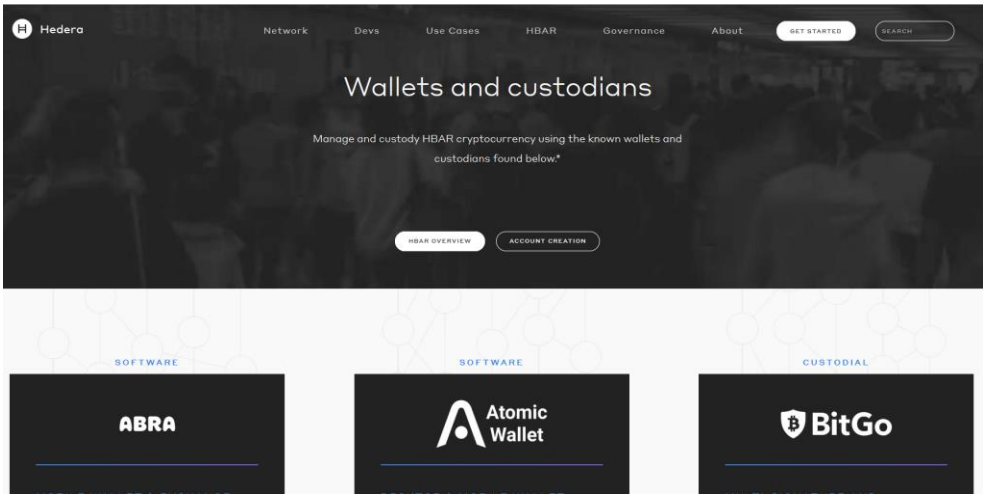
What else?

Interactive Exercise 4

Before completing this exercise, watch the steps for this interactive exercise included in the video lessons.

Navigate to https://hedera.com/buying-guide and https://docs.hedera.com/guides/mainnet/mainnet-access

Select a wallet and follow the steps for mainnet account creation.



Mainnet Access

You will need a Hedera mainnet account to interact with and pay for any of the network services (cryptocurrency, consensus, tokens, files and smart contracts). Your Hedera account is what holds a balance of hbar to be used for transfers to other accounts or payments for network services.

Create free mainnet accounts by visiting any of these wallet providers:

Wallet	Private Key Viewable	SDK-compatible Passphrase
Atomic	☑	☐
Coinomi	☐	☑
Exodus (Desktop)	☑	☐
Guarda	☑	☐
Hashpack	☑	☑
Xact	☑	☑
Wallawallet	☑	☑

After you receive your account ID and private keys, you can create additional mainnet accounts using the SDKs. When using the SDKs to build your Hedera client, select `client.forMainnet()`. A mainnet node will be picked at random to submit the transaction to.

Interactive Exercise 5

Mirrornet Queries

1. Before completing Start with base MirrorNet URL
   https://mainnet-public.mirrornode.hedera.com
2. Append text to URL /api/v1

   https://mainnet-public.mirrornode.hedera.com/api/v1
3. Add text for queries: transactions, accounts, tokens etc

   https://mainnet-public.mirrornode.hedera.com/api/v1/transactions

Mirrornet for Mainnet URLs:

https://mainnet-public.mirrornode.hedera.com/api/v1/transactions

https://mainnet-public.mirrornode.hedera.com/api/v1/accounts

Mirrornet for Testnet URLs

https://testnet.mirrornode.hedera.com/api/v1/transactions/?account.id=0.0.29589707

https://testnet.mirrornode.hedera.com/api/v1/accounts?account.id=0.0.29589707

https://testnet.mirrornode.hedera.com/api/v1/tokens?account.id=0.0.29589707

Run queries:

https://mainnet-public.mirrornode.hedera.com/api/v1/transactions

https://mainnet-public.mirrornode.hedera.com/api/v1/accounts

Add an ascending and descending filter.

https://testnet.mirrornode.hedera.com/api/v1/transactions

https://testnet.mirrornode.hedera.com/api/v1/accounts

https://testnet.mirrornode.hedera.com/api/v1/tokens

Run the queries below for your testnet account id

https://testnet.mirrornode.hedera.com/api/v1/transactions

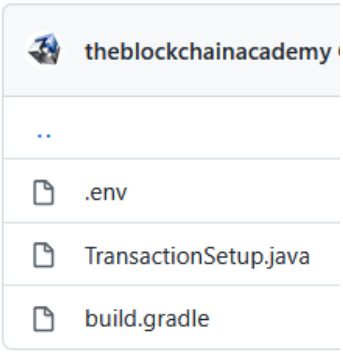
https://testnet.mirrornode.hedera.com/api/v1/accounts

https://testnet.mirrornode.hedera.com/api/v1/tokens

Interactive Exercise 6

Before completing this exercise, watch the steps for this interactive exercise included in the video lessons.

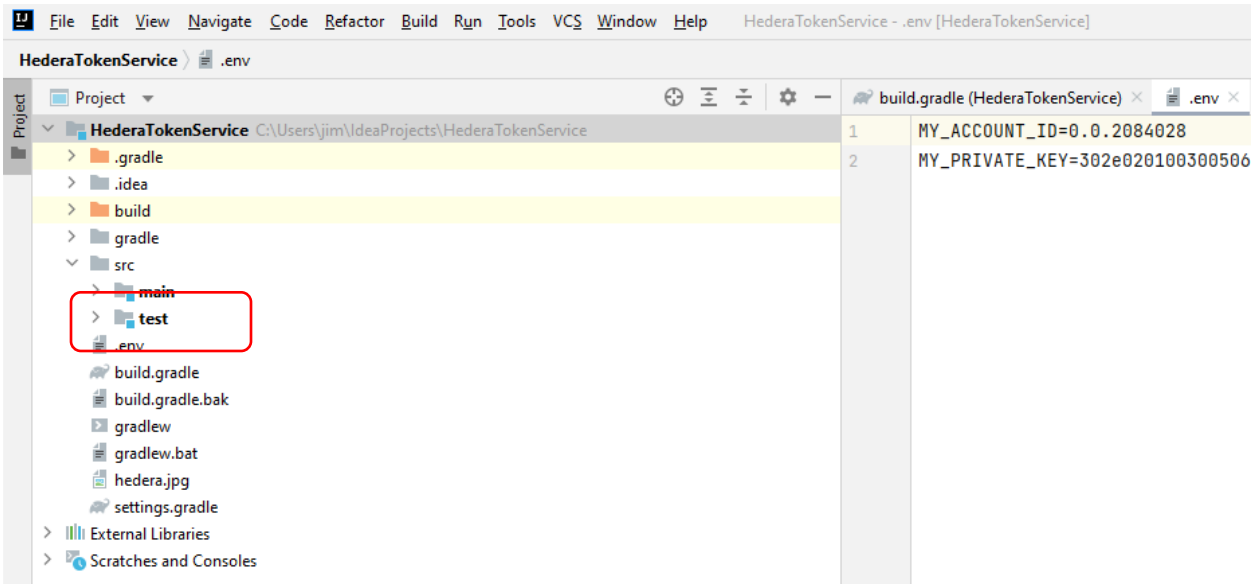
Use code in:
https://github.com/theblockchainacademy/Hedera/tree/main/Hedera_8_TransactionSetupExercise8
and complete the exercise using the code in the repository.



Start **IntelliJ** and click New Project. Name the new project HederaTokenServiceSetup.

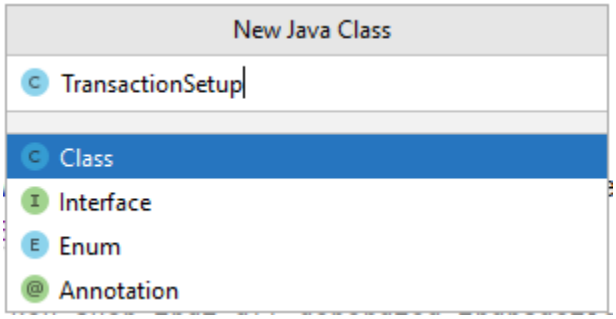
Follow the Environmental Configurations, IntelliJ Steps 1 – 8, from the beginning of this document.

Make sure you update the build.gradle file is updated from this GitHub repository. Use your own .env file with your own account ID and private key. An example is included in this GitHub repository.



Right click on folder **src/main/java**. Click **New** -> **Java Class**.

Enter **TransactionSetup**



Enter code from the repository:
https://github.com/theblockchainacademy/Hedera/tree/main/Hedera_8_TransactionSetupExercise8

TransactionSetup.java

```java
import com.hedera.hashgraph.sdk.*;
import io.github.cdimascio.dotenv.Dotenv;

import java.nio.charset.StandardCharsets;
import java.util.*;
import java.util.concurrent.TimeoutException;

public final class TransactionSetup {

    /// see `.env.sample` in the repository root for how to specify these values
    // or set environment variables with the same names
    private static final AccountId OPERATOR_ID = AccountId.fromString(Objects.requireNonNull(Dotenv.load().get("MY_ACCOUNT_ID")));
    private static final PrivateKey OPERATOR_KEY = PrivateKey.fromString(Objects.requireNonNull(Dotenv.load().get("MY_PRIVATE_KEY")));
    // HEDERA_NETWORK defaults to testnet if not specified in dotenv
    private static final String HEDERA_NETWORK = Dotenv.load().get( envName: "HEDERA_NETWORK",  defValue: "testnet");


    private TransactionSetup() {
    }

    public static void main(String[] args) throws TimeoutException, PrecheckStatusException, ReceiptStatusException {
        Client client = Client.forName(HEDERA_NETWORK);

        // Defaults the operator account ID and key such that all generated transactions will be paid for
        // by this account and be signed by this key
        client.setOperator(OPERATOR_ID, OPERATOR_KEY);

        // Create three accounts, Alice and Bob. Alice will be the treasury for our example token.
        // Fees only apply in transactions not involving the treasury, so we need two other accounts.

        PrivateKey aliceKey = PrivateKey.generate();
        AccountId aliceId = new AccountCreateTransaction()
                .setInitialBalance(new Hbar( amount: 10)) AccountCreateTransaction
                .setKey(aliceKey)
                .freezeWith(client)
                .sign(aliceKey)
                .execute(client) TransactionResponse
                .getReceipt(client) TransactionReceipt
                .accountId;
```

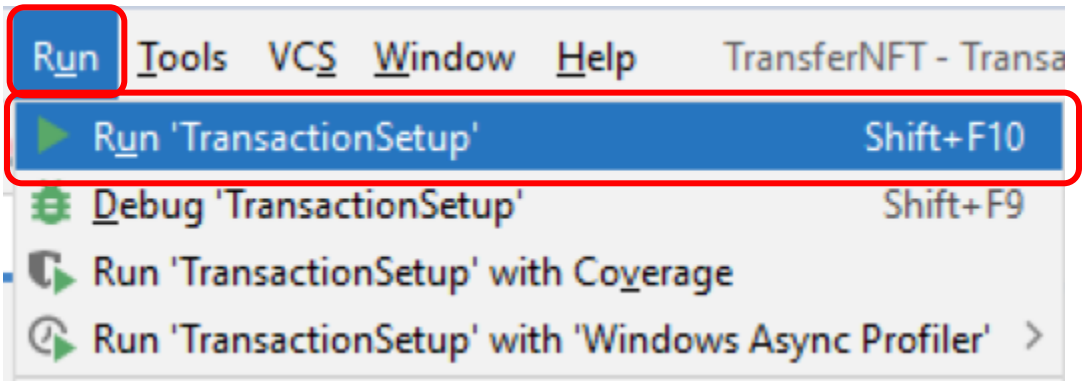
TransactionSetup.java

```java
        PrivateKey bobKey = PrivateKey.generate();
        AccountId bobId = new AccountCreateTransaction()
                .setInitialBalance(new Hbar( amount: 10)) AccountCreateTransaction
                .setKey(bobKey)
                .freezeWith(client)
                .sign(bobKey)
                .execute(client) TransactionResponse
                .getReceipt(client) TransactionReceipt
                .accountId;

        System.out.println("Alice account ID: " + aliceId);
        System.out.println("Bob account ID: " + bobId);

        TokenId tokenId = new TokenCreateTransaction()
                .setTokenName("TBA NFT") TokenCreateTransaction
                .setTokenSymbol("TBA-NFT")
                .setTokenType(TokenType.NON_FUNGIBLE_UNIQUE)
                .setAdminKey(aliceKey) // so that we can delete later on
                .setSupplyKey(aliceKey) // so that we can mint later on
                .setTreasuryAccountId(aliceId)
                .freezeWith(client)
                .sign(aliceKey)
                .execute(client) TransactionResponse
                .getReceipt(client) TransactionReceipt
                .tokenId;

        System.out.println("NFT Token ID: " + tokenId);

        // We must associate the token with Bob and Charlie before they can trade in it.

        new TokenAssociateTransaction()
                .setAccountId(bobId)
                .setTokenIds(Collections.singletonList(tokenId))
                .freezeWith(client)
                .sign(bobKey)
                .execute(client)
                .getReceipt(client);
```

TransactionSetup.java

```
        AccountBalance accountBalance = new AccountBalanceQuery() //One method to get account balance
                .setAccountId(aliceId)
                .execute(client);

        System.out.println("The Alice account balance before is: " +accountBalance);

        AccountBalance accountBobBalance = new AccountBalanceQuery() //One method to get account balance
                .setAccountId(bobId)
                .execute(client);

        System.out.println("The Bob account balance before is: " +accountBobBalance);

        //Transaction record used to output transaction details.
        //Send the last part of the NFT to the receiver. Charge HBar fees and get a record.
        TransactionRecord record1 = new TransferTransaction()
                //.addNftTransfer(new NftId(tokenId, serials.size()), aliceId, bobId)
                .addHbarTransfer(bobId, Hbar.from(-4))
                .addHbarTransfer(aliceId, Hbar.from(4))
                .freezeWith(client)
                .sign(bobKey)
                .sign(aliceKey)
                .execute(client)
                .getRecord(client);

        System.out.println("Transaction record: tokenID: {[TokenNftTransfer{sender, receiver, serial}]} " + record1.tokenNftTransfers);

        Hbar aliceHbar2 = new AccountBalanceQuery()
                .setAccountId(aliceId)
                .execute(client)
                .hbars;


        Hbar bobHbar2 = new AccountBalanceQuery()
                .setAccountId(bobId)
                .execute(client)
                .hbars;

        System.out.println("Alices's HBar balance after token transfer to Bob: " + aliceHbar2);
        System.out.println("Bob's HBar balance after token transfer from Alice: " + bobHbar2);

        client.close();
```

Click **Run** -> **Run 'TransactionSetup'**



The results will appear close to the output below:

```
9:36:56 AM: Executing task ':TransactionSetup.main()'...

> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE

> Task :TransactionSetup.main()
[main] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.7 in 250 ms after failure during attempt #1: OK
[main] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.7 in 500 ms after failure during attempt #2: OK
[main] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.5 in 250 ms after failure during attempt #1: OK
[main] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.5 in 500 ms after failure during attempt #2: OK
Alice account ID: 0.0.30873300
Bob account ID: 0.0.30873301
[main] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.3 in 250 ms after failure during attempt #1: OK
[main] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.3 in 500 ms after failure during attempt #2: OK
NFT Token ID: 0.0.30873302
[main] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.6 in 250 ms after failure during attempt #1: OK
[main] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.6 in 500 ms after failure during attempt #2: OK
The Alice account balance before is: AccountBalance{hbars=10 ?, tokens={0.0.30873302=0}}
The Bob account balance before is: AccountBalance{hbars=10 ?, tokens={0.0.30873302=0}}
[main] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.4 in 250 ms after failure during attempt #1: OK
[main] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.4 in 500 ms after failure during attempt #2: OK
Transaction record: tokenID: {[TokenNftTransfer{sender, receiver, serial}]} {}
Alices's HBar balance after token transfer to Bob: 14 ?
Bob's HBar balance after token transfer from Alice: 6 ?

Deprecated Gradle features were used in this build, making it incompatible with Gradle 8.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

See https://docs.gradle.org/7.1/userguide/command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 8s
2 actionable tasks: 1 executed, 1 up-to-date
9:37:05 AM: Task execution finished ':TransactionSetup.main()'.
```

Interactive Exercise 7

Consensus establishes what for the network?

All nodes agree and …

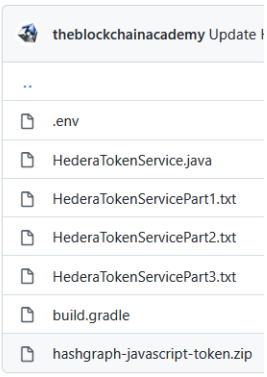
Double spending is prevented.

What else?

Interactive Exercise 8

Hedera Token Service

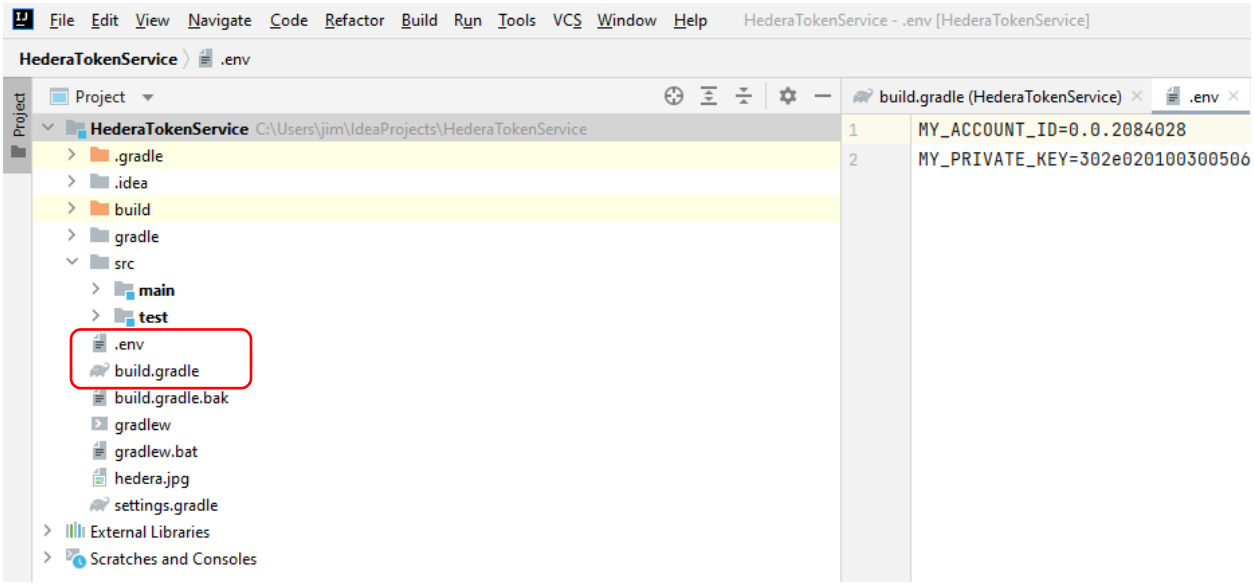
Use code in:
https://github.com/theblockchainacademy/Hedera/tree/main/Hedera_2_TokenServiceExercise2 and complete the following steps. Iteratively complete the exercise using files part1 to part3.



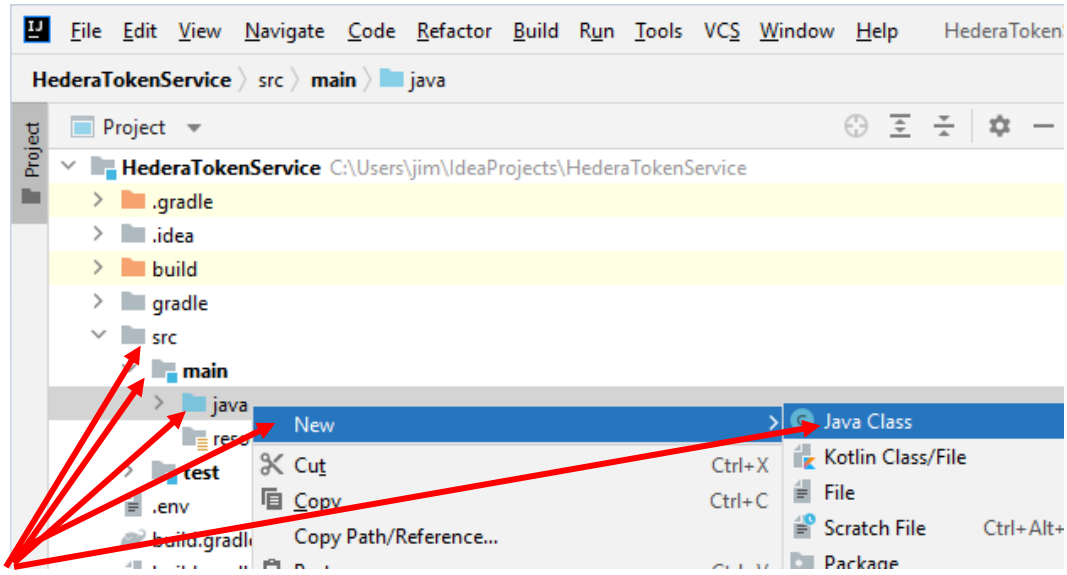
Start ItelliJ and click New Project. Name the new project HederaTokenService.

Follow the Environmental Configurations, IntelliJ Steps 1 – 8, from the beginning of this document.

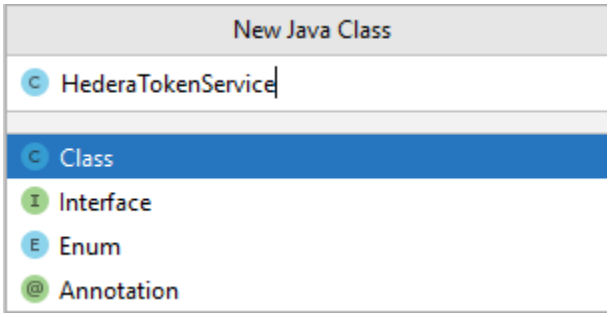
Make sure you update the build.gradle file is updated from this GitHub repository. Use your own .env file with your own account ID and private key. An example is included in this GitHub repository.



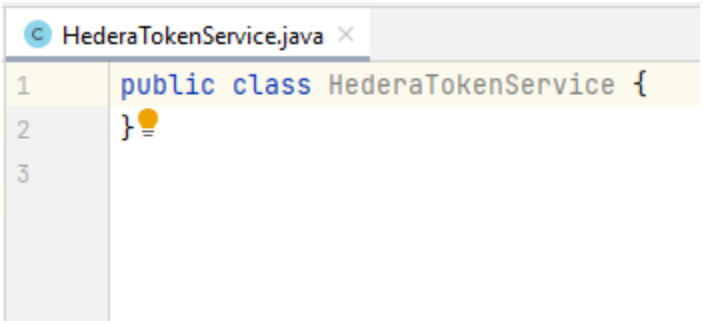
Right click on folder **src/main/java**. Click **New** -> **Java Class**.



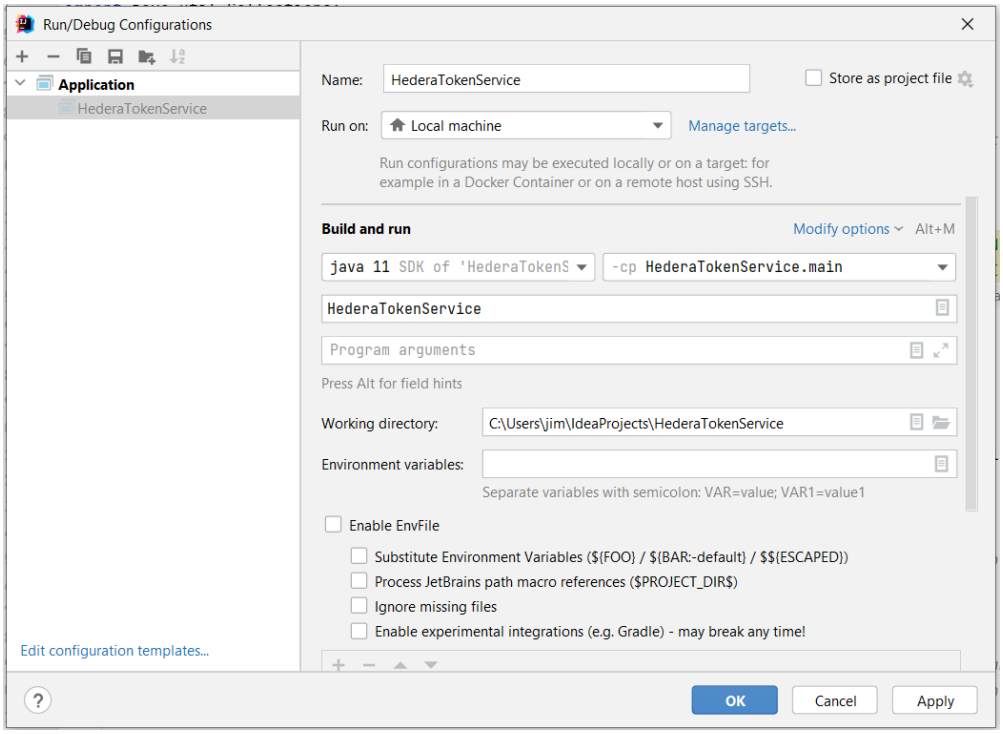
Name the new class **HederaTokenService** and press the **enter** key.



The code stubs will be created.



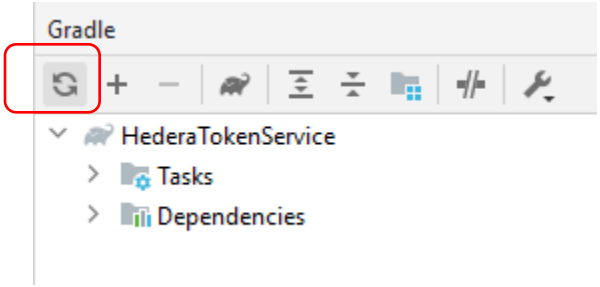
Validate project configurations, see IntelliJ step 9. Project configurations will resemble the image below.



Enter Part 1 of the code below, use GitHub file: HederaTokenServicePart1.txt, https://github.com/theblockchainacademy/Hedera/blob/main/Hedera_2_TokenServiceExercise2/HederaTokenServicePart1.txt

```
import com.hedera.hashgraph.sdk.*;

import io.github.cdimascio.dotenv.Dotenv;

import java.util.Collections;

import java.util.concurrent.TimeoutException;


public final class HederaTokenService {


    //private static final AccountId OPERATOR_ID =
AccountId.fromString(Objects.requireNonNull(Dotenv.load().get("OPERATO
R_ID")));

    //private static final AccountId OPERATOR_ID =
AccountId.fromString("0.0.2084028");


    // HEDERA_NETWORK defaults to testnet if not specified in dotenv

    private static final AccountId OPERATOR_ID =
AccountId.fromString(Dotenv.load().get("MY_ACCOUNT_ID"));

    private static final PrivateKey OPERATOR_KEY =
PrivateKey.fromString(Dotenv.load().get("MY_PRIVATE_KEY"));

    private static final String HEDERA_NETWORK =
Dotenv.load().get("HEDERA_NETWORK", "testnet");

    private HederaTokenService() {

    }


    public static void main(String[] args) throws TimeoutException,
PrecheckStatusException, ReceiptStatusException {


    }


}
```

Run only **IntelliJ Step 6, Refresh Gradle**, from **Exercise 1**.



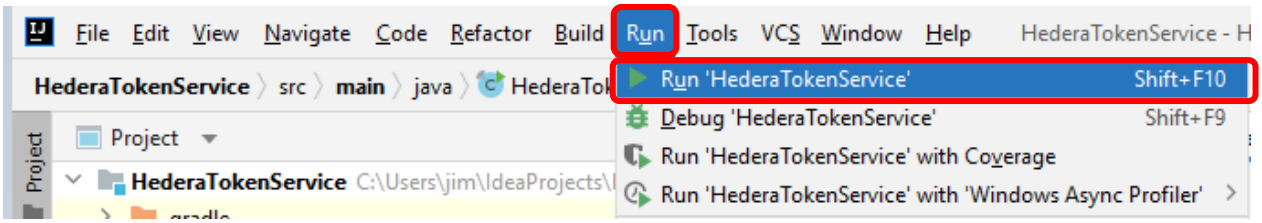
The application should appear as below.

HederaTokenService.java

```
1  import com.hedera.hashgraph.sdk.*;
2  import io.github.cdimascio.dotenv.Dotenv;
3
4  import java.util.Collections;
5  import java.util.concurrent.TimeoutException;
6
7  public final class HederaTokenService {
8
9      // see `.env.sample` in the repository root for how to specify these values
10     // or set environment variables with the same names
11     //private static final AccountId OPERATOR_ID = AccountId.fromString(Objects.requireNonNull(Dotenv.load().get("OPERATOR_ID")));
12     //private static final AccountId OPERATOR_ID = AccountId.fromString("0.0.2084028");
13
14     // HEDERA_NETWORK defaults to testnet if not specified in dotenv
15     private static final AccountId OPERATOR_ID = AccountId.fromString(Dotenv.load().get("MY_ACCOUNT_ID"));
16     private static final PrivateKey OPERATOR_KEY = PrivateKey.fromString(Dotenv.load().get("MY_PRIVATE_KEY"));
17     private static final String HEDERA_NETWORK = Dotenv.load().get( envName: "HEDERA_NETWORK",  defValue: "testnet");
18
19
20     private HederaTokenService() {
21     }
22
23     public static void main(String[] args) throws TimeoutException, PrecheckStatusException, ReceiptStatusException {
24
25     }
26
27 }
```

Run the Application

Click **Run** -> **Run 'HederaTokenService'**



The results will appears as shown below.

```
3:51:17 PM: Executing task ':HederaTokenService.main()'...

> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :HederaTokenService.main()

Deprecated Gradle features were used in this build, making it incompatible with Gradle 8.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

See https://docs.gradle.org/7.1/userguide/command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 596ms
2 actionable tasks: 1 executed, 1 up-to-date
3:51:18 PM: Task execution finished ':HederaTokenService.main()'.
```

Build a client to interact with Hedera Nodes: https://docs.hedera.com/guides/docs/sdks/client

- Client.forTestnet()
- Client.forMainnet()
- Client.forName(*HEDERA_NETWORK*)

Use the Hedera Token Service to create a fungible token, and accounts for trading.

- Generate a key for Alice
- Create an account for Alice
- Generate a key for Bob
- Create an account for Bob.
- Output the results to the console.

Refer to the solution in GitHub: HederaTokenServicePart2.txt, https://github.com/theblockchainacademy/Hedera/blob/main/Hedera_2_TokenServiceExercise2/HederaTokenServicePart2.txt

The results will resemble the results shown below

```
2:23:01 PM: Executing task ':HederaTokenService.main()'...

> Task :compileJava
> Task :processResources NO-SOURCE
> Task :classes

> Task :HederaTokenService.main()
[hedera-sdk-7] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.6 in 250 ms after failure during attempt #1: OK
[hedera-sdk-3] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.6 in 500 ms after failure during attempt #2: OK
[hedera-sdk-6] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.6 in 1000 ms after failure during attempt #3: OK
[hedera-sdk-3] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.3 in 250 ms after failure during attempt #1: OK
[hedera-sdk-6] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.3 in 500 ms after failure during attempt #2: OK
Alice's account ID: 0.0.2773618
Bob's account ID: 0.0.2773619
Alice's Hbar balance before token transfers: 10 ?

Deprecated Gradle features were used in this build, making it incompatible with Gradle 8.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

See https://docs.gradle.org/7.1/userguide/command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 7s
2 actionable tasks: 2 executed
2:23:08 PM: Task execution finished ':HederaTokenService.main()'.
```

Create the Token

Reference: https://docs.hedera.com/guides/docs/sdks/tokens,
https://docs.hedera.com/guides/docs/sdks/tokens/define-a-token

Use the Hedera Token Service to create a new fungible token (FT) .

Set the token name as **Example Token**, and the symbol as **EX**.

Set Alice as the treasury account, Set the initial supply to **20**.

Return the token ID, and the token type. Verify that that the token is of type fungible.

Associate Bob with the token and mint 100 additional tokens of type **Example Token**.

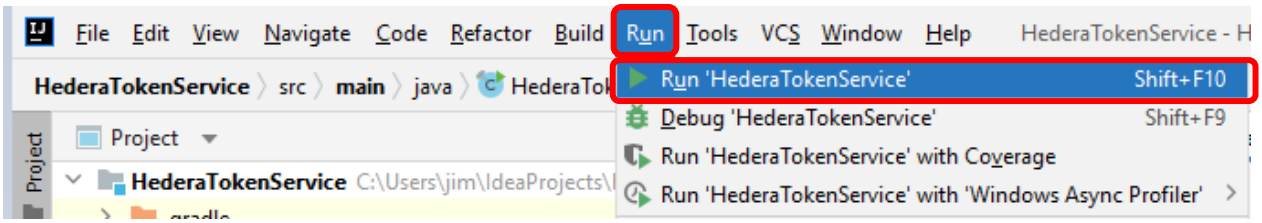
Get the balance of tokens in Alice's and Bob's wallet after minting and output the results.

Transfer 40 tokens of type **Example Token** from Alice to Bob, and charge Bob 1 HBar.

Get the token and HBar Balance for Alice and Bob after the transfer transaction and output to the console.

Run the Application

Click **Run** -> **Run 'HederaTokenService'**



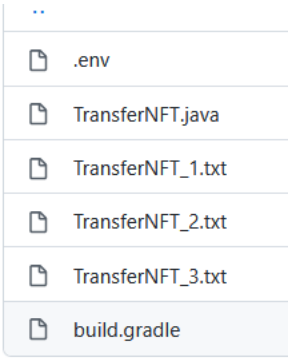
The results will appears as shown below.

Refer to the solution in GitHub: [HederaTokenServicePart3.txt](), https://github.com/theblockchainacademy/Hedera/blob/main/Hedera_2_TokenServiceExercise2/HederaTokenServicePart3.txt

```
> Task :HederaTokenService.main()
[hedera-sdk-7] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.5 in 250 ms after failure during att
[hedera-sdk-3] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.5 in 500 ms after failure during att
[hedera-sdk-6] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.4 in 250 ms after failure during att
[hedera-sdk-0] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.4 in 500 ms after failure during att
Alice's account ID: 0.0.2817465
Bob's account ID: 0.0.2817467
Alice's Hbar balance before token transfers: 10 ?
[hedera-sdk-0] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.6 in 250 ms after failure during att
[hedera-sdk-4] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.6 in 500 ms after failure during att
Token: 0.0.2817468
FUNGIBLE_COMMON
[hedera-sdk-4] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.7 in 250 ms after failure during att
[hedera-sdk-7] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.7 in 500 ms after failure during att
[hedera-sdk-2] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.7 in 1000 ms after failure during at
Minting
[hedera-sdk-2] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.3 in 250 ms after failure during att
[hedera-sdk-6] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.3 in 500 ms after failure during att
[hedera-sdk-3] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.3 in 1000 ms after failure during at
The transaction consensus status is SUCCESS
The Alice account balance before is: AccountBalance{hbars=10 ?, tokens={0.0.2817468=120}}
The Bob account balance before is: AccountBalance{hbars=10 ?, tokens={0.0.2817468=0}}
Transfer
[hedera-sdk-7] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.6 in 250 ms after failure during att
[hedera-sdk-0] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.6 in 500 ms after failure during att
The Alice account balance after is: AccountBalance{hbars=11 ?, tokens={0.0.2817468=80}}
The Bob account balance after is: AccountBalance{hbars=9 ?, tokens={0.0.2817468=40}}
Alice's Hbar balance after token: 11 ?
```

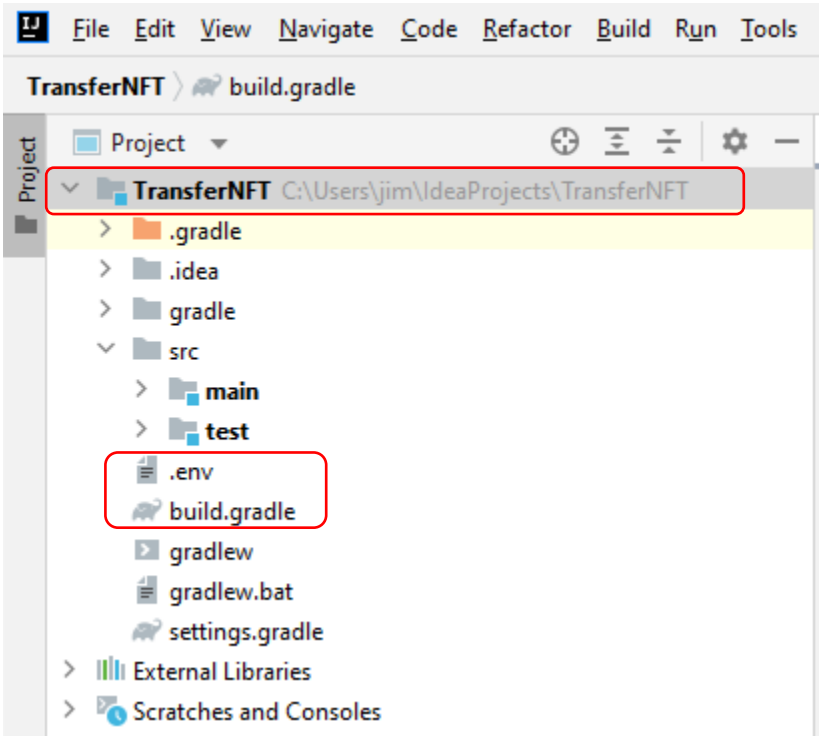
Create a Non-Fungible Token (NFT)

Use code in:
https://github.com/theblockchainacademy/Hedera/tree/main/Hedera_3_NFTTokensExercise3 and complete the following steps. Iteratively complete the exercise using files TransferNFT_1 to TransferNFT_3.

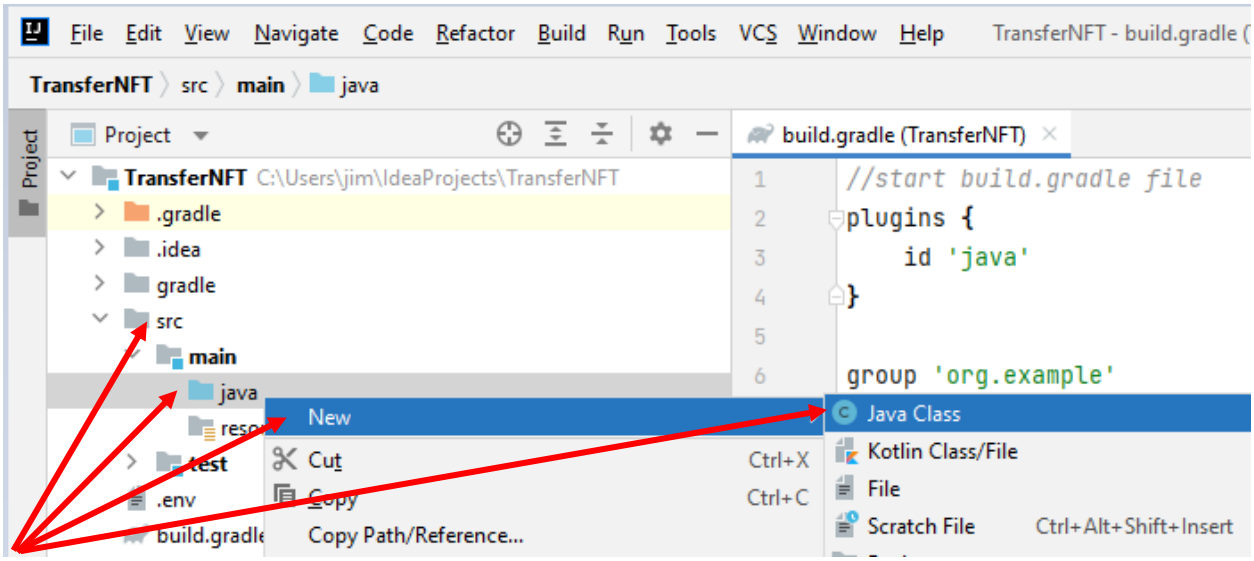


Start ItelliJ and click New Project called **TransferNFT**

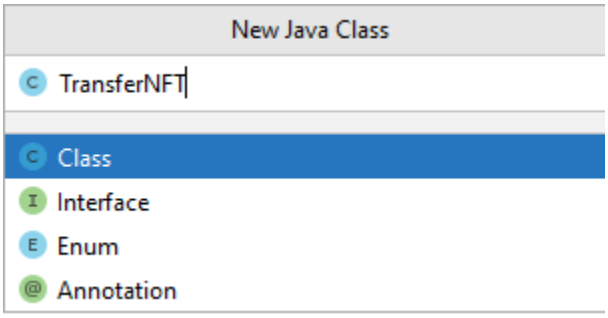
Follow the IntelliJ Steps 1 – 7, from assignment 1. Make sure the .env and build.gradle files are updated.



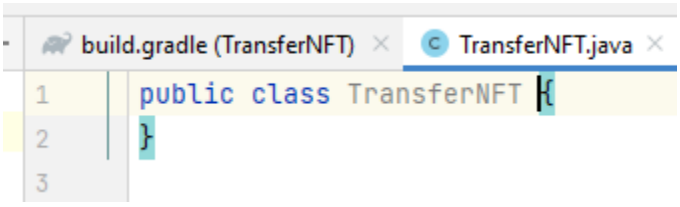
Right click on folder **src/main/java**. Click **New** -> **Java Class**.



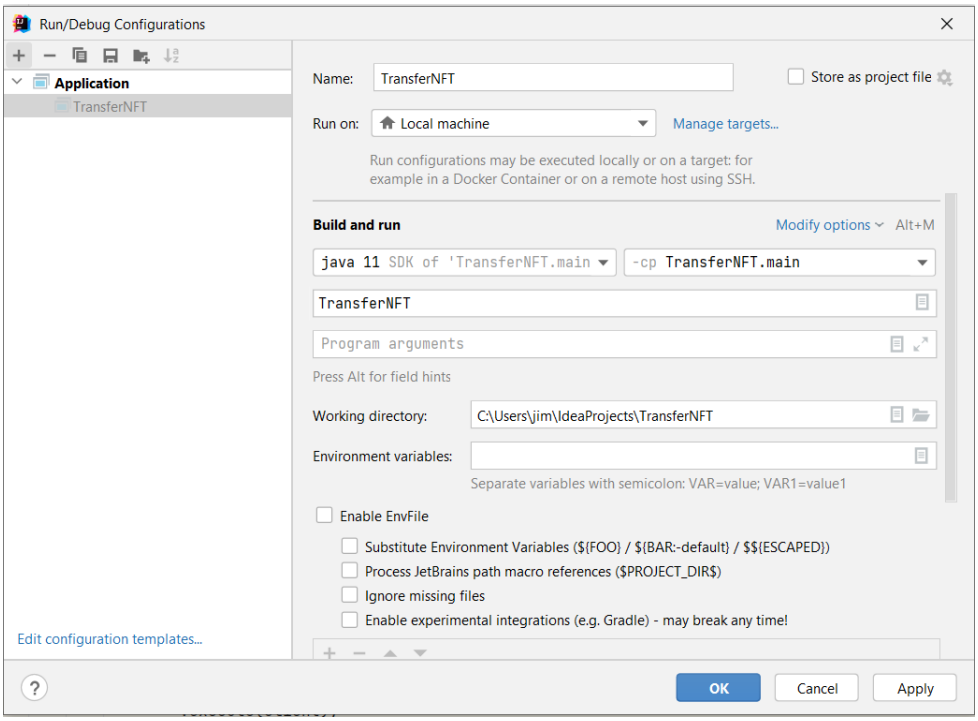
Name the new class **TransferNFT**, and press the enter key.



The code stubs will be created

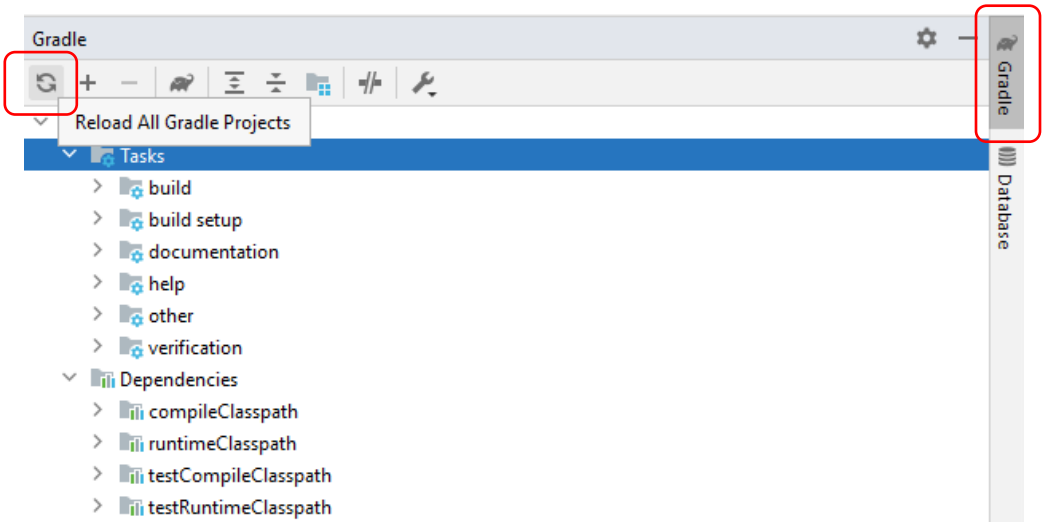


Validate project configurations, see **IntelliJ step 9**. Project configurations will resemble the image below.

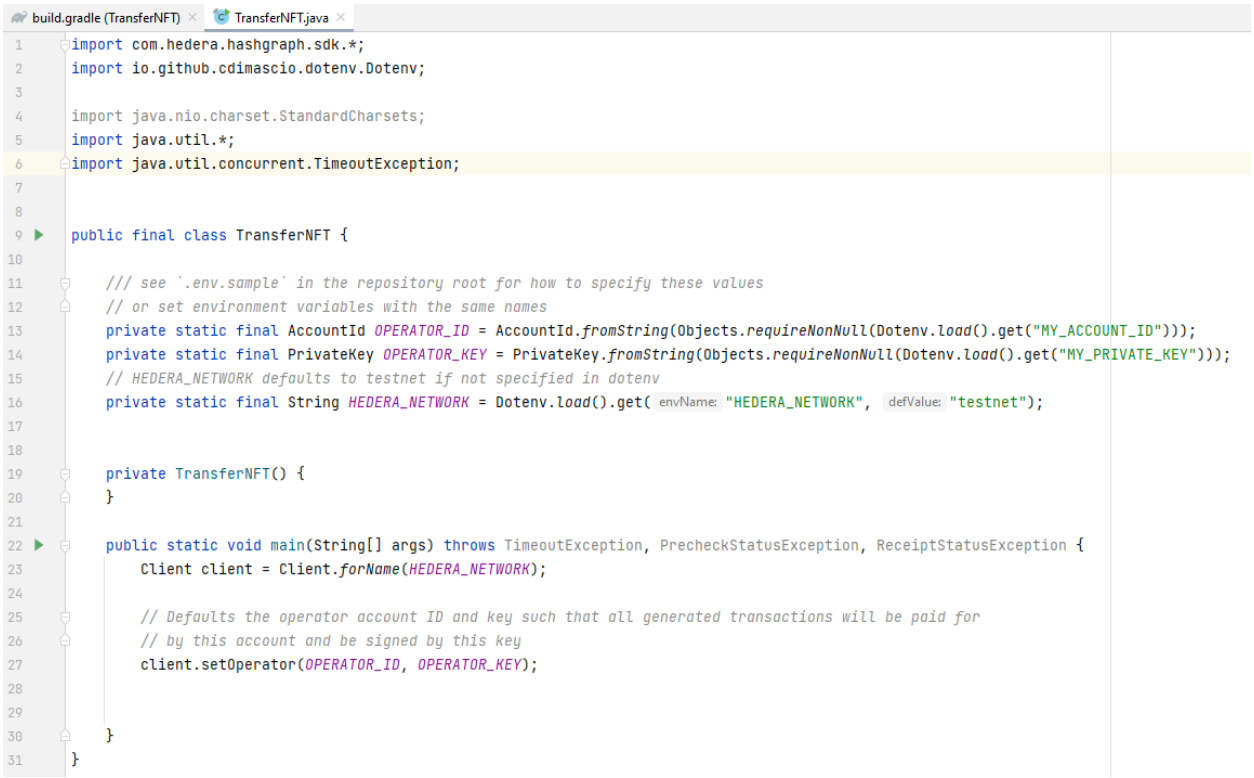


Update the dependencies

Click the **Gradle button** in the upper right corner, and then click the **Reload All Gradle Projects** button.



Once the update is complete, click the **Gradle button** in the upper right corner again to close.



Enter the source code from GitHub: TransferNFT_1.txt, https://github.com/theblockchainacademy/Hedera/blob/main/Hedera_3_NFTTokensExercise3/TransferNFT_1.txt

Create an account for Alice and Bob and output the results. Set the HBar initial supply to 10.

Create the token for the NFT. Name it **TBA NFT** and set the symbol to **TBA-NFT**.

Set the token type is non fungible token unique, set the admin, supply and treasury, to Alice's private key. Get a receipt for reporting.

Output the token ID to the console.

Use Get token info to create an object to query the token type and output the token type to the console.

Refer to the solution on GitHub: TransferNFT_2.txt, https://github.com/theblockchainacademy/Hedera/blob/main/Hedera_3_NFTTokensExercise3/TransferNFT_2.txt

As always, if needed, click the Gradle button in the upper right corner, and then click the Reload All Gradle Projects button.

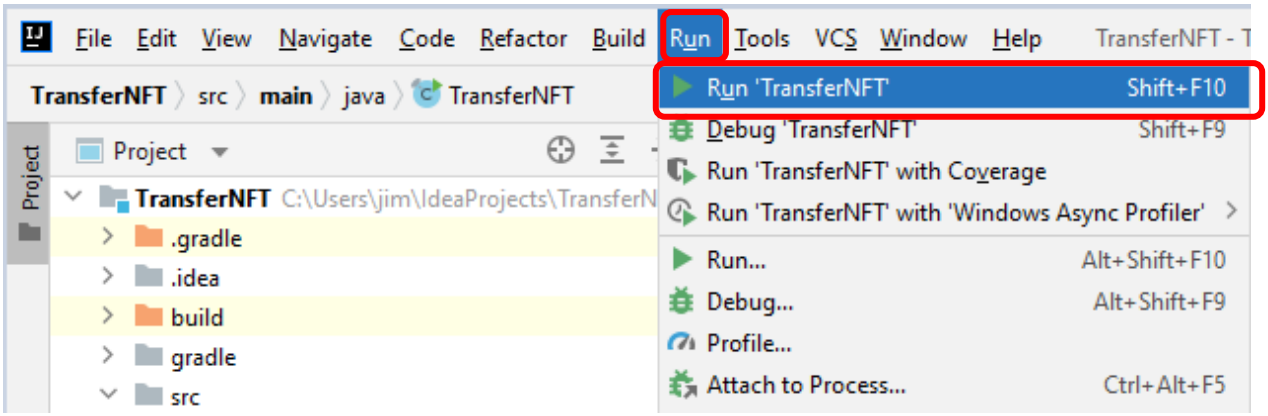
You project should appear as shown below. The screenshot below shows only the new code added.

```
30
31      PrivateKey aliceKey = PrivateKey.generate();
32      AccountId aliceId = new AccountCreateTransaction()
33              .setInitialBalance(new Hbar( amount: 10)) AccountCreateTransaction
34              .setKey(aliceKey)
35              .freezeWith(client)
36              .sign(aliceKey)
37              .execute(client) TransactionResponse
38              .getReceipt(client) TransactionReceipt
39              .accountId;
40
41      PrivateKey bobKey = PrivateKey.generate();
42      AccountId bobId = new AccountCreateTransaction()
43              .setInitialBalance(new Hbar( amount: 10)) AccountCreateTransaction
44              .setKey(bobKey)
45              .freezeWith(client)
46              .sign(bobKey)
47              .execute(client) TransactionResponse
48              .getReceipt(client) TransactionReceipt
49              .accountId;
50
51      System.out.println("Alice account ID: " + aliceId);
52      System.out.println("Bob account ID: " + bobId);
```

```
54      TokenId tokenId = new TokenCreateTransaction()
55              .setTokenName("TBA NFT") TokenCreateTransaction
56              .setTokenSymbol("TBA-NFT")
57              .setTokenType(TokenType.NON_FUNGIBLE_UNIQUE)
58              .setAdminKey(aliceKey) // so that we can delete later on
59              .setSupplyKey(aliceKey) // so that we can mint later on
60              .setTreasuryAccountId(aliceId)
61              .freezeWith(client)
62              .sign(aliceKey)
63              .execute(client) TransactionResponse
64              .getReceipt(client) TransactionReceipt
65              .tokenId;
66
67      System.out.println("NFT Token ID: " + tokenId);
68
69      TokenInfo tokenInfo = new TokenInfoQuery()
70              .setTokenId(tokenId)
71              .execute(client);
72
73      System.out.println(tokenInfo.tokenType);
```

Run the Application

Click **Run** -> **Run 'TransferNFT"**



The results will appears as shown below. Ignore the warnings: [hedera-sdk-7] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery.

```
10:49:22 AM: Executing task ':TransferNFT.main()'...

> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE

> Task :TransferNFT.main()
[hedera-sdk-7] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.3 in 250 ms after failure during attempt #1: OK
[hedera-sdk-3] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.3 in 500 ms after failure during attempt #2: OK
[hedera-sdk-6] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.6 in 250 ms after failure during attempt #1: OK
[hedera-sdk-2] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.6 in 500 ms after failure during attempt #2: OK
[hedera-sdk-6] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.6 in 1000 ms after failure during attempt #3: OK
Alice account ID: 0.0.2841611
Bob account ID: 0.0.2841612
[hedera-sdk-1] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.4 in 250 ms after failure during attempt #1: OK
[hedera-sdk-4] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.4 in 500 ms after failure during attempt #2: OK
[hedera-sdk-1] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.4 in 1000 ms after failure during attempt #3: OK
NFT Token ID: 0.0.2841613
NON_FUNGIBLE_UNIQUE

Deprecated Gradle features were used in this build, making it incompatible with Gradle 8.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

See https://docs.gradle.org/7.1/userguide/command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 7s
2 actionable tasks: 1 executed, 1 up-to-date
10:49:30 AM: Task execution finished ':TransferNFT.main()'.
```

Associate the token with Bob so he can transact it.

Create byte arrays with the NFT's metadata. Then create a byte array list array for the metadata.

Mint the NFT using TokenMintTransaction, https://docs.hedera.com/guides/docs/sdks/tokens/mint-a-token.

The TokenMintTransaction will return a long which is the NFT's serial number.

Based on the serials, this results in NFT IDs: token ID.0, token ID.1, token ID.2 ….

Get the account balances before the token transfer. Use AccountBalanceQuery to return Alice and Bob's account balance of NFTs and HBar before the NFT transfer. Query Alice's and Bob's account balance and return it to the console.

Use TransferTransaction to send all the NFTs from Alice to Bob. Also transfer 4 HBar from Bob to Alice.

Use AccountBalanceQuery to return Alice and Bob's account balance of NFTs and HBar after the transfer.

Refer to the solution on GitHub: TransferNFT_3.txt, https://github.com/theblockchainacademy/Hedera/blob/main/Hedera_3_NFTTokensExercise3/TransferNFT_3.txt

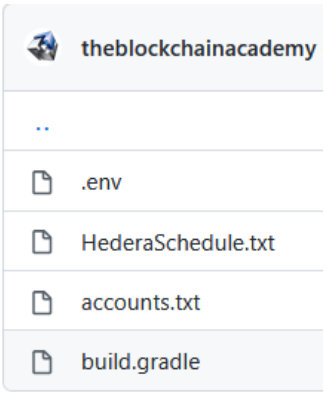
The results will appears as shown below. Ignore the warnings: [hedera-sdk-7] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery.

```
[hedera-sdk-5] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.6 in 500 ms after failure during attempt #2: OK
[hedera-sdk-1] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.6 in 1000 ms after failure during attempt #3: OK
Alice account ID: 0.0.2844117
Bob account ID: 0.0.2844118
[hedera-sdk-4] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.4 in 250 ms after failure during attempt #1: OK
[hedera-sdk-1] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.4 in 500 ms after failure during attempt #2: OK
[hedera-sdk-3] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.4 in 1000 ms after failure during attempt #3: OK
NFT Token ID: 0.0.2844119
NON_FUNGIBLE_UNIQUE
[hedera-sdk-7] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.7 in 250 ms after failure during attempt #1: OK
[hedera-sdk-2] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.7 in 500 ms after failure during attempt #2: OK
[hedera-sdk-5] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.7 in 1000 ms after failure during attempt #3: OK
Minting
[hedera-sdk-5] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.5 in 250 ms after failure during attempt #1: OK
[hedera-sdk-2] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.5 in 500 ms after failure during attempt #2: OK
[hedera-sdk-6] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.5 in 1000 ms after failure during attempt #3: OK
The NFT has 3 individual parts.
The Alice account balance before is: AccountBalance{hbars=10 ?, tokens={0.0.2844119=3}}
The Bob account balance before is: AccountBalance{hbars=10 ?, tokens={0.0.2844119=0}}
Transfer
[hedera-sdk-7] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.3 in 250 ms after failure during attempt #1: OK
[hedera-sdk-2] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.3 in 500 ms after failure during attempt #2: OK
[hedera-sdk-6] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.3 in 1000 ms after failure during attempt #3: OK
Transaction record: tokenID: {[TokenNftTransfer{sender, receiver, serial}]} {0.0.2844119=[TokenNftTransfer{sender=0.0.2844117, receiver=0.0.2844118, serial=3}]}
Alices's NFT balance after transfer to Bob: {0.0.2844119=0}
Alices's HBar balance after token transfer to Bob: 14 ?
Bob's NFT balance after transfer from Alice: {0.0.2844119=3}
Bob's HBar balance after token transfer from Alice: 6 ?
```

Interactive Exercise 9

Hedera Schedule Service

Use code in:
https://github.com/theblockchainacademy/Hedera/tree/main/Hedera_7_ScheduleExercise7 and complete the exercise using the code in the repository.

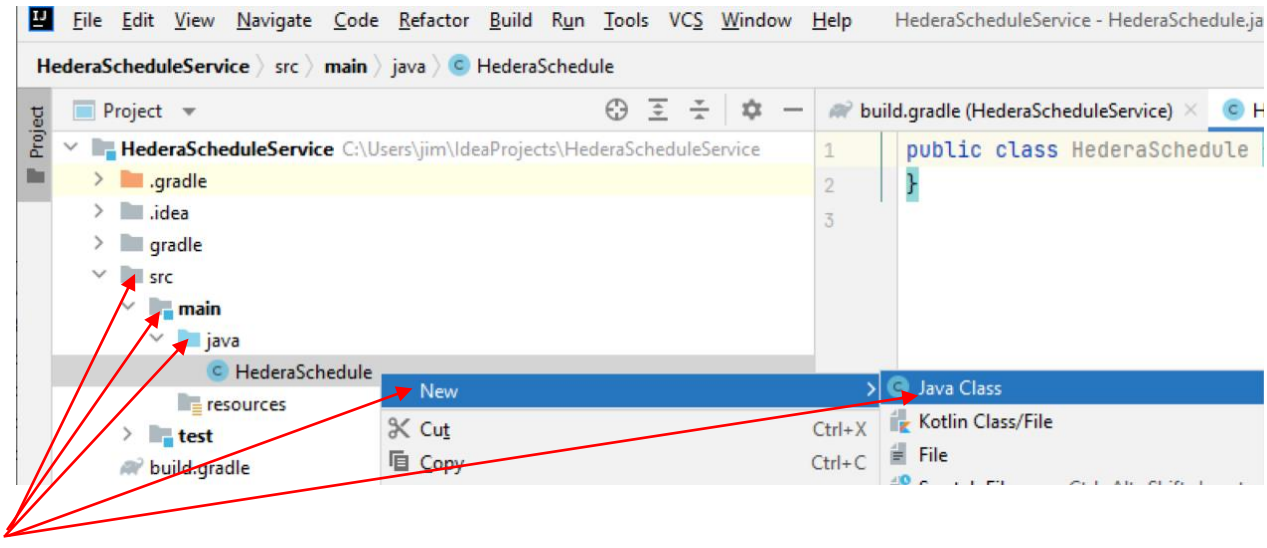


Start **ItelliJ** and click New Project. Name the new project HederaScheduleService.

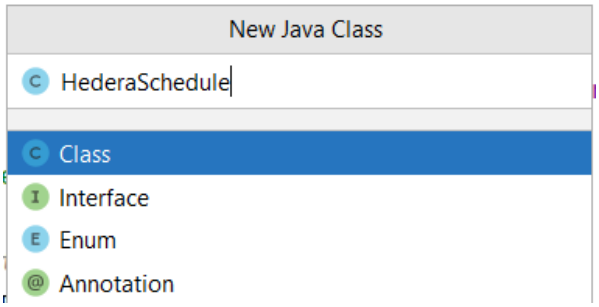
Follow the Environmental Configurations, IntelliJ Steps 1 – 8, from the beginning of this document.

Make sure you update the build.gradle file is updated from this GitHub repository. Use your own .env file with your own account ID and private key. An example is included in this GitHub repository.

Right click on folder **src/main/java**. Click **New** -> **Java Class**.



Enter: **HederaSchedule**



Paste in the code from:
https://github.com/theblockchainacademy/Hedera/tree/main/Hedera_7_ScheduleExercise7

build.gradle (HederaScheduleService) | .env | HederaSchedule.java | accounts.txt

```java
import com.hedera.hashgraph.sdk.*;
import io.github.cdimascio.dotenv.Dotenv;

import java.util.concurrent.TimeoutException;

public class HederaSchedule {

    // HEDERA_NETWORK defaults to testnet if not specified in dotenv
    private static final AccountId myAccountId = AccountId.fromString(Dotenv.load().get("MY_ACCOUNT_ID"));
    private static final PrivateKey myPrivateKey = PrivateKey.fromString(Dotenv.load().get("MY_PRIVATE_KEY"));
    private static final String HEDERA_NETWORK = Dotenv.load().get( envName: "HEDERA_NETWORK",  defValue: "testnet");

    public static void main(String[] args) throws TimeoutException, PrecheckStatusException, ReceiptStatusException {

        //Create your Hedera testnet client
        Client client = Client.forTestnet();
        client.setOperator(myAccountId, myPrivateKey);

        // Generate a new key pair
        PrivateKey newAccountPrivateKey = PrivateKey.generate();
        PublicKey newAccountPublicKey = newAccountPrivateKey.getPublicKey();

        //Create new account and assign the public key
        TransactionResponse newAccount = new AccountCreateTransaction()
                .setKey(newAccountPublicKey)
                .setInitialBalance( Hbar.fromTinybars(1000))
                .execute(client);

        // Get the new account ID
        AccountId newAccountId = newAccount.getReceipt(client).accountId;

        System.out.println("The new account ID is: " +newAccountId);

        //Check the new account's balance
        AccountBalance accountBalance = new AccountBalanceQuery()
                .setAccountId(newAccountId)
                .execute(client);

        System.out.println("The new account balance is: " +accountBalance.hbars);
```

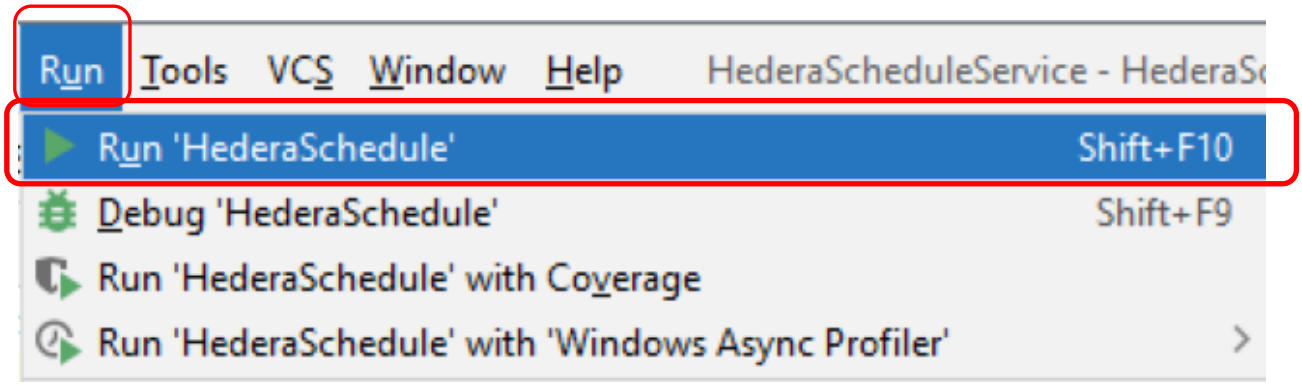
build.gradle (HederaScheduleService) | .env | HederaSchedule.java | accounts.txt

```
46          //Schedule a transaction
47          TransactionResponse scheduleTransaction = new ScheduleCreateTransaction()
48                  .setScheduledTransaction(sendHbar)
49                  .execute(client);
50
51          //Get the receipt of the transaction
52          TransactionReceipt receipt = scheduleTransaction.getReceipt(client);
53
54          //Get the schedule ID
55          ScheduleId scheduleId = receipt.scheduleId;
56          System.out.println("The schedule ID is " +scheduleId);
57
58
59          //Get the scheduled transaction ID
60          TransactionId scheduledTxId = receipt.scheduledTransactionId;
61          System.out.println("The scheduled transaction ID is " +scheduledTxId);
62
63          //Submit the first signatures
64          TransactionResponse signature1 = new ScheduleSignTransaction()
65                  .setScheduleId(scheduleId)
66                  .freezeWith(client)
67                  .sign(myPrivateKey)
68                  .execute(client);
69
70          //Verify the transaction was successful and submit a schedule info request
71          // TransactionReceipt receipt1 = signature1.getReceipt(client);
72          // System.out.println("The transaction status is " +receipt1.status);
73
74          ScheduleInfo query1 = new ScheduleInfoQuery()
75                  .setScheduleId(scheduleId)
76                  .execute(client);
77
78          //Confirm the signature was added to the schedule
79          System.out.println(query1);
80
81          //Submit the second signature
82          TransactionResponse signature2 = new ScheduleSignTransaction()
83                  .setScheduleId(scheduleId)
84                  .freezeWith(client)
85                  .sign(myPrivateKey)
86                  .execute(client);
```

```
96
97          System.out.println(query2);
98
99          //Get the scheduled transaction record
100         TransactionRecord scheduledTxRecord = TransactionId.fromString(scheduledTxId.toString()).getRecord(client);
101         System.out.println("The scheduled transaction record is: " +scheduledTxRecord);
102
103     }
104 }
```

Run **HederaSchedule**

Click: **Run** (menu) -> **Run 'HederaSchedule'**



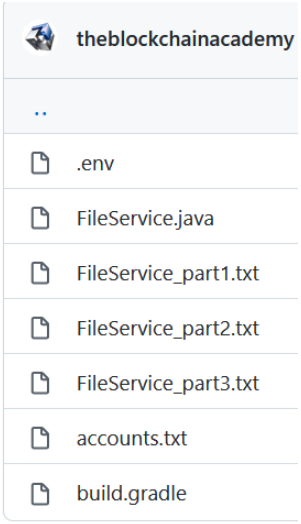
The results should resemble what is shown below:

```
11:11:12 AM: Executing task ':HederaSchedule.main()'...

> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE

> Task :HederaSchedule.main()
[main] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.6 in 250 ms after failure during attempt #1: OK
The new account ID is: 0.0.30873619
The new account balance is: 1000 t?
[main] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.5 in 250 ms after failure during attempt #1: OK
[main] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.5 in 500 ms after failure during attempt #2: OK
The schedule ID is 0.0.30873620
The scheduled transaction ID is 0.0.2084028@1646669466.184897216?scheduled
ScheduleInfo{scheduleId=0.0.30873620, creatorAccountId=0.0.2084028, payerAccountId=0.0.2084028, signatories=KeyList{threshold=null, keys=[302a300506032b6570032100665edb671816
ScheduleInfo{scheduleId=0.0.30873620, creatorAccountId=0.0.2084028, payerAccountId=0.0.2084028, signatories=KeyList{threshold=null, keys=[302a300506032b6570032100665edb671816
The scheduled transaction record is: TransactionRecord{receipt=TransactionReceipt{status=SUCCESS, exchangeRate=ExchangeRate{hbars=30000, cents=627366, expirationTime=2022-03-

Deprecated Gradle features were used in this build, making it incompatible with Gradle 8.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

See https://docs.gradle.org/7.1/userguide/command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 6s
2 actionable tasks: 1 executed, 1 up-to-date
11:11:19 AM: Task execution finished ':HederaSchedule.main()'.
```

Interactive Exercise 10

The Hedera File Service

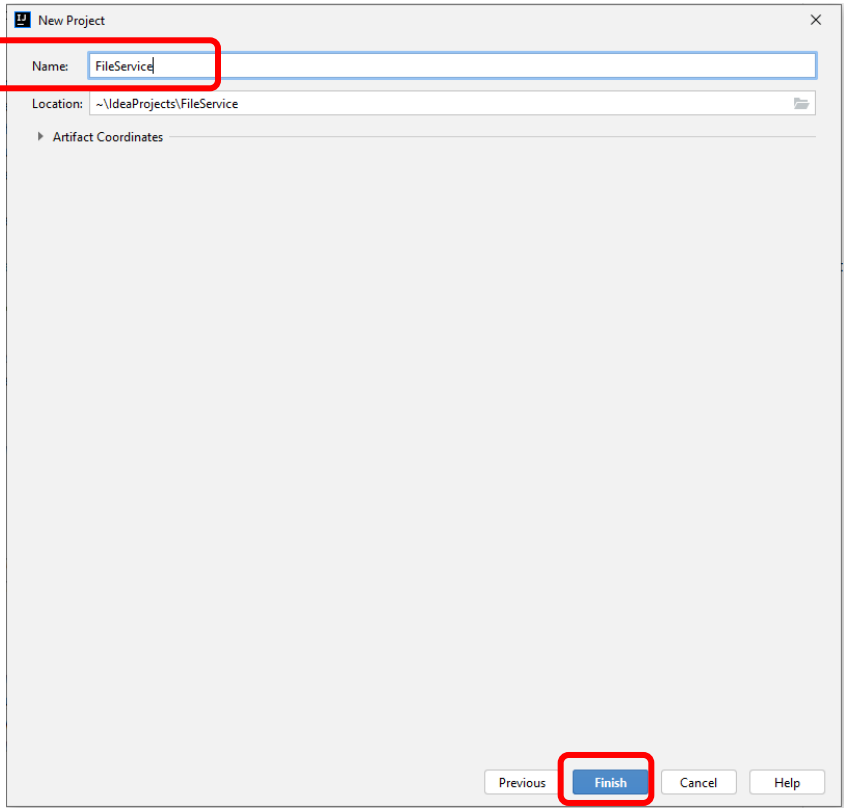
Use code in:
https://github.com/theblockchainacademy/Hedera/tree/main/Hedera_4_FileServiceExercise4 and complete the exercise using the code in the repository. Iteratively complete the exercise using files FileService_part1.txt to FileService_part3.txt.



Start ItelliJ and click New Project called FileService

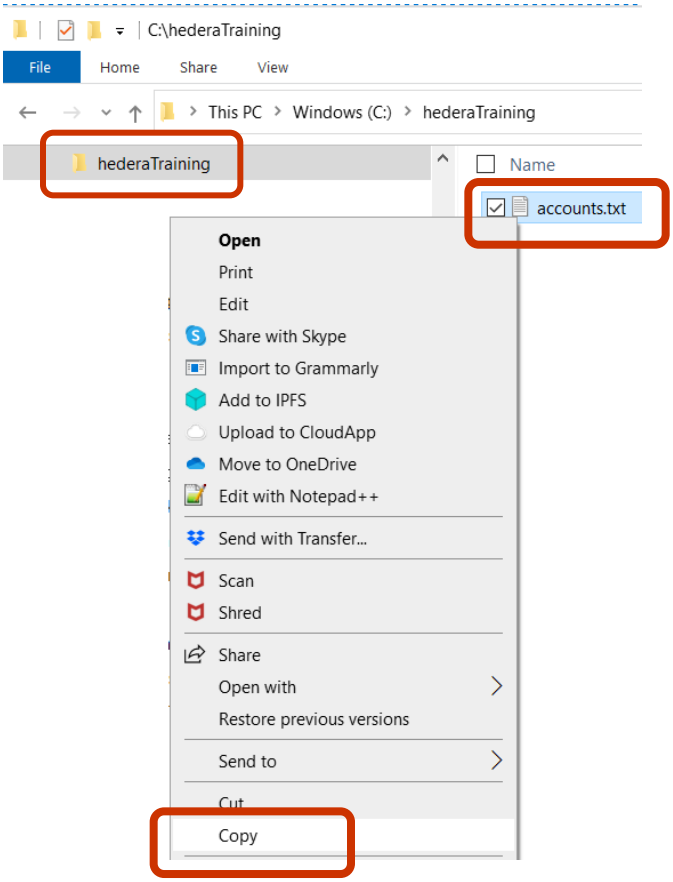
Follow the IntelliJ Steps 1 – 7, from assignment 1. Make sure the .env and build.gradle files are updated. You may copy paste in the .env file and build.gradle files from the previous exercises or from the GitHub repository. The same .env and build.gradle files are required.

Make sure you update the build.gradle file is updated from this GitHub repository. Use your own .env file with your own account ID and private key. An example is included in this GitHub repository.

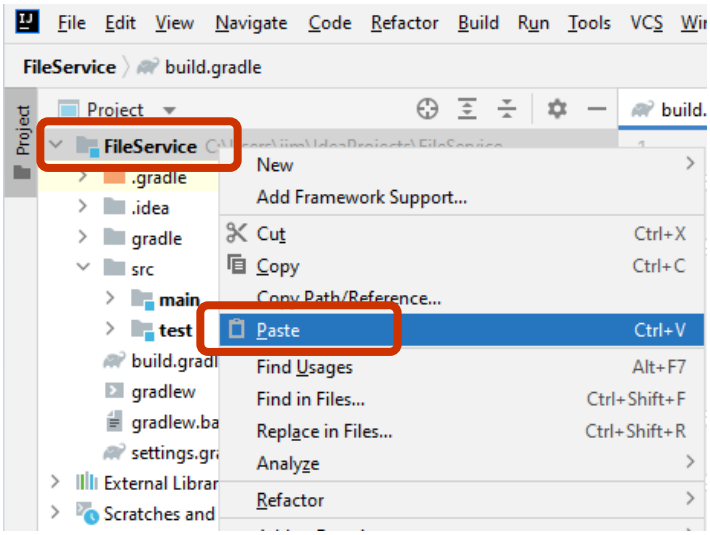


In the prerequisites section, a file called account.txt was created. The accounts.txt file was stored in a directory called hederaTraining.

Open the directory and copy the accounts.txt file.

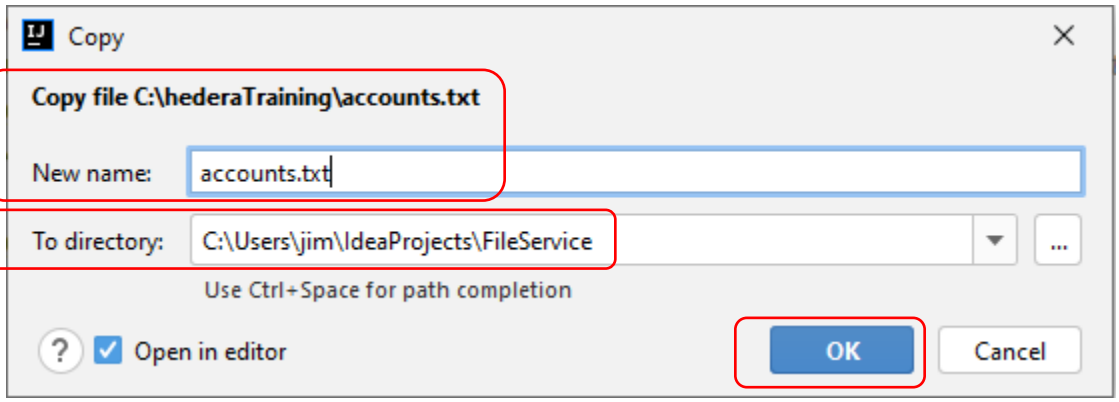


Paste the accounts.txt file into the top level of the FileService project.

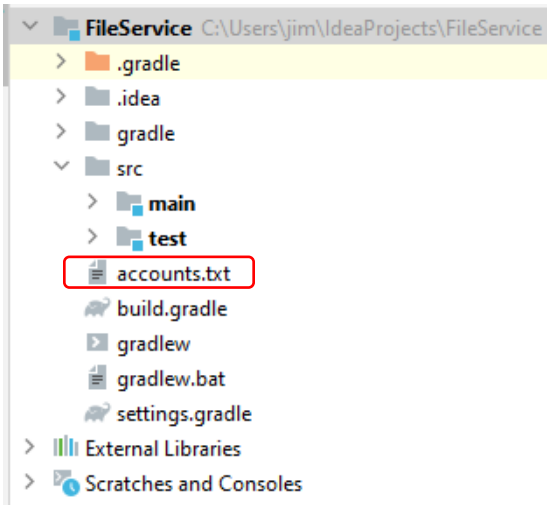


Validate the file, the source directory, and the destination (to directory) directory.

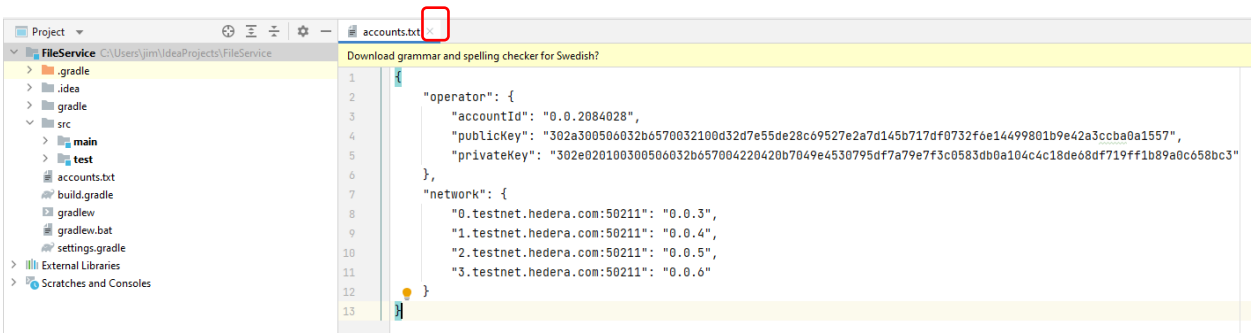
Click **OK**.



The **accounts.txt** file will appear in the project.



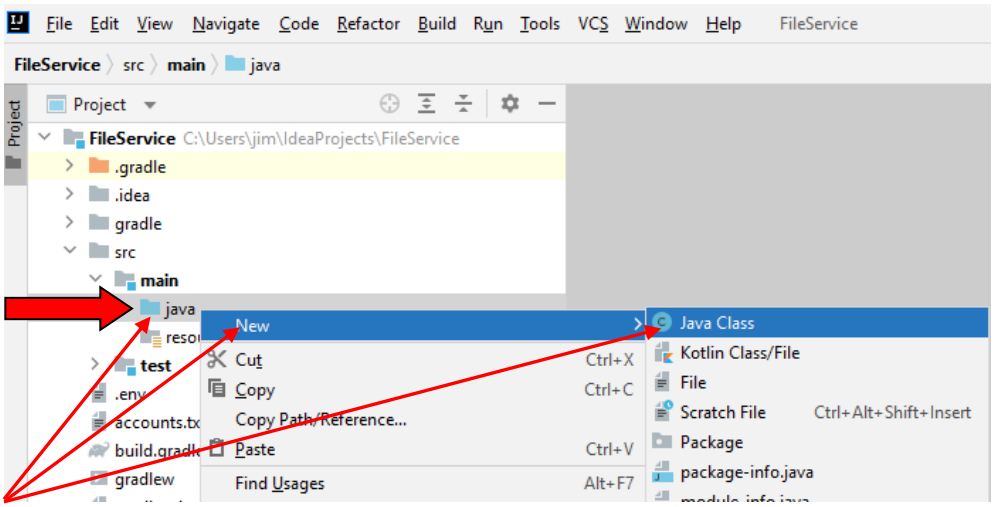
The accounts.txt file will open when pasted into the FileService project. Close the file when you are done validating it.



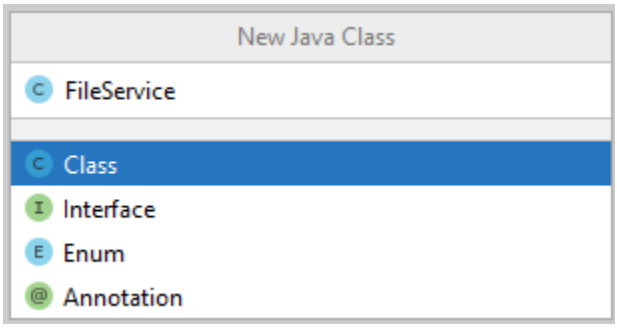
Create a new class called FileService

In the project expand \src\main

Right click on \src\main\java, and click New -> Java Class

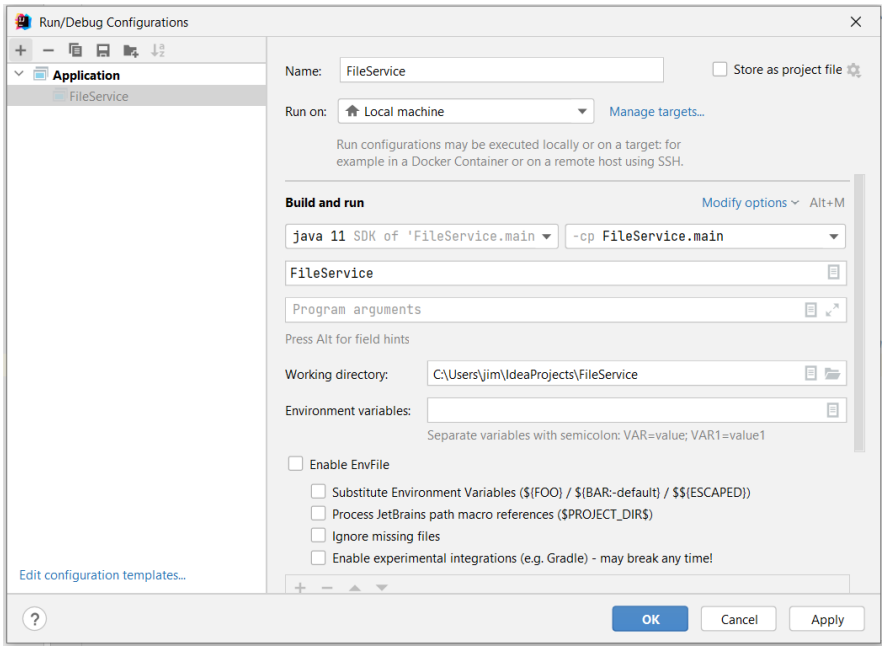


Enter the class name: **FileSerivce** and tap enter key.



The **FileService.java** file will open.

Validate project configurations, see **IntelliJ step 9**. Project configurations will resemble the image below.

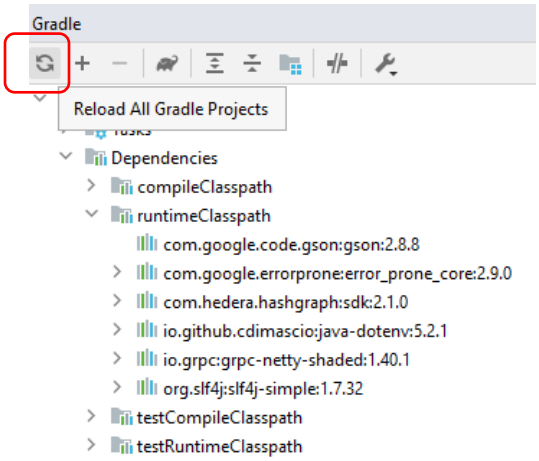


At the top of the FileService.java file, enter all the import libraries shown below.

```
import com.hedera.hashgraph.sdk.*;
import io.github.cdimascio.dotenv.Dotenv;
import com.google.protobuf.ByteString;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.security.NoSuchAlgorithmException;
import java.util.concurrent.TimeoutException;
```

Update the Gradle dependencies.

Click the Gradle button in the upper right corner. Click the **Refresh Gradle Project** button.



Close the Gradle tools, click the  button in the upper right corner.

Create a main function as shown below.

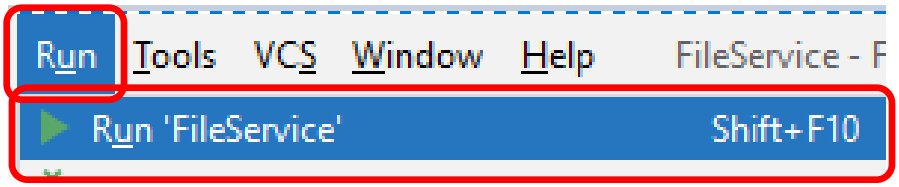
Declare a string variable for the filename accounts.txt

FileService.java

```java
import com.hedera.hashgraph.sdk.*;
import io.github.cdimascio.dotenv.Dotenv;
import com.google.protobuf.ByteString;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.security.NoSuchAlgorithmException;
import java.util.concurrent.TimeoutException;

public class FileService {

    public static void main(String[] args) throws TimeoutException, PrecheckStatusException, ReceiptStatusException, IOException, NoSuchAlgorithmException {

        String myFile = "accounts.txt";

    }
}
```

Run the FileService project.

Click Run -> Run 'FileService'



The output will appear as similar to what is shown below

```
> Task :compileJava
> Task :processResources NO-SOURCE
> Task :classes
> Task :FileService.main()

Deprecated Gradle features were used in this build, making it incompatible with Gradle 8.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

See https://docs.gradle.org/7.1/userguide/command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 829ms
2 actionable tasks: 2 executed
3:51:38 PM: Task execution finished ':FileService.main()'.
```

Refer to the solution in GitHub: FileService_part1.txt, https://github.com/theblockchainacademy/Hedera/blob/main/Hedera_4_FileServiceExercise4/FileService_part1.txt

Part 2

Create a file object from the accounts.txt file. Since the accounts.txt file is at the project level, the path to the accounts.txt file is ./ there we do not have to prefix a path to the accounts.txt filename.

Use the source code below

```
String myFile = "accounts.txt";

//pass the filename as a File object
File f = new File(myFile);

String content = Files.readString(Paths.get(myFile));
```

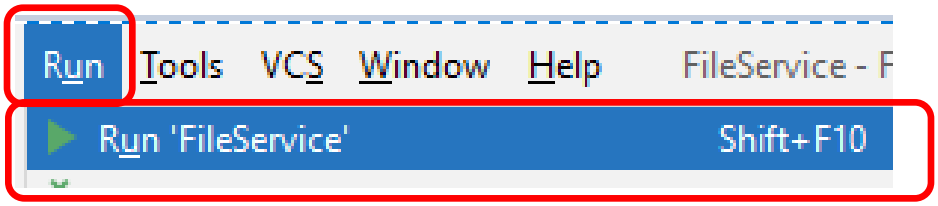
This is a standard IO operation for files.

FileService.java

```
import com.hedera.hashgraph.sdk.*;
import io.github.cdimascio.dotenv.Dotenv;
import com.google.protobuf.ByteString;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.security.NoSuchAlgorithmException;
import java.util.concurrent.TimeoutException;

public class FileService {

    public static void main(String[] args) throws TimeoutException, PrecheckStatusException, ReceiptStatusException, IOException, NoSuchAlgorithmException {

        String myFile = "accounts.txt";

        //pass the filename as a File object
        File f = new File(myFile);

        String content = Files.readString(Paths.get(myFile));

    }
}
```

Run the FileService project.

Click Run -> Run 'FileService'



The output will appear as similar to what is shown below

```
> Task :compileJava
> Task :processResources NO-SOURCE
> Task :classes
> Task :FileService.main()

Deprecated Gradle features were used in this build, making it incompatible with Gradle 8.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

See https://docs.gradle.org/7.1/userguide/command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 829ms
2 actionable tasks: 2 executed
3:51:38 PM: Task execution finished ':FileService.main()'.
```

Refer to the solution in GitHub: FileService_part2.txt, https://github.com/theblockchainacademy/Hedera/blob/main/Hedera_4_FileServiceExercise4/FileService_part2.txt

Part 3

1. Obtain your account ID and private key form the .env file
2. Create a client from the Hedera testnet,
3. Set your account ID and private for the testnet client.
4. Generate a new public and private key pair.
5. Use FileCreateTransaction to create a file transaction that sets: the file private key, a file key, a file memo, set the file contents using the file object.
6. Call setMaxTransactionFee on the transaction object to change the default max transaction fee to 2 hbars.
7. Prepare transaction for signing, sign with the key on the file, sign with the client operator key and submit to a Hedera network.
8. Confirm the transaction fee by requesting a receipt by calling getReceipt on the transaction response.
9. Get the file ID by calling fileId on the receipt object. Return the file ID to the console.
10. Get the file contents by calling getContents on the transaction object created in step 5.
11. Return the contents to a string and out put it to the console, the output may be abridged.
12. Use FileInfoQuery to query the file on the network for information.
13. Out the query information to the console. It will be abridged.

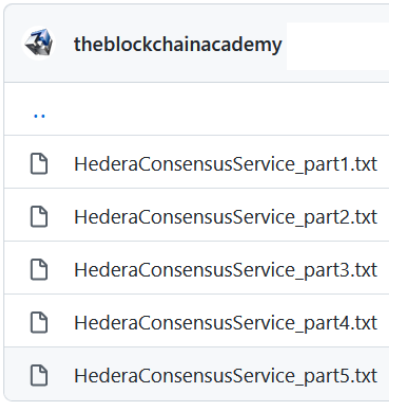
Refer to the solution on GitHub: FileService_part3.txt, https://github.com/theblockchainacademy/Hedera/blob/main/Hedera_4_FileServiceExercise4/FileService_part3.txt

The output will appear similar to what is shown below.

Interactive Exercise 11

The Hedera Consensus Service

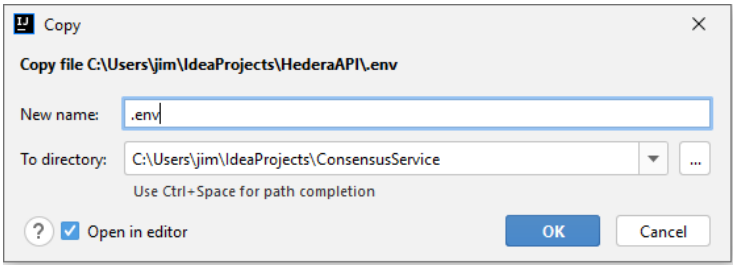
Use code in:
https://github.com/theblockchainacademy/Hedera/tree/main/Hedera_5_ConsensusServiceExercise5
and complete the exercise using the code in the repository. Iteratively complete the exercise using files HederaConsensusService_part1.txt to HederaConsensusService_part5.txt.



Start ItelliJ and click New Project called ConsensusService

Follow the IntelliJ Steps 1 – 7, from assignment 1, or the previous exercise. Make sure the .env and build.gradle files are updated. You may copy paste in the .env file and build.gradle files from the previous exercises. The same .env and build.gradle files are required.

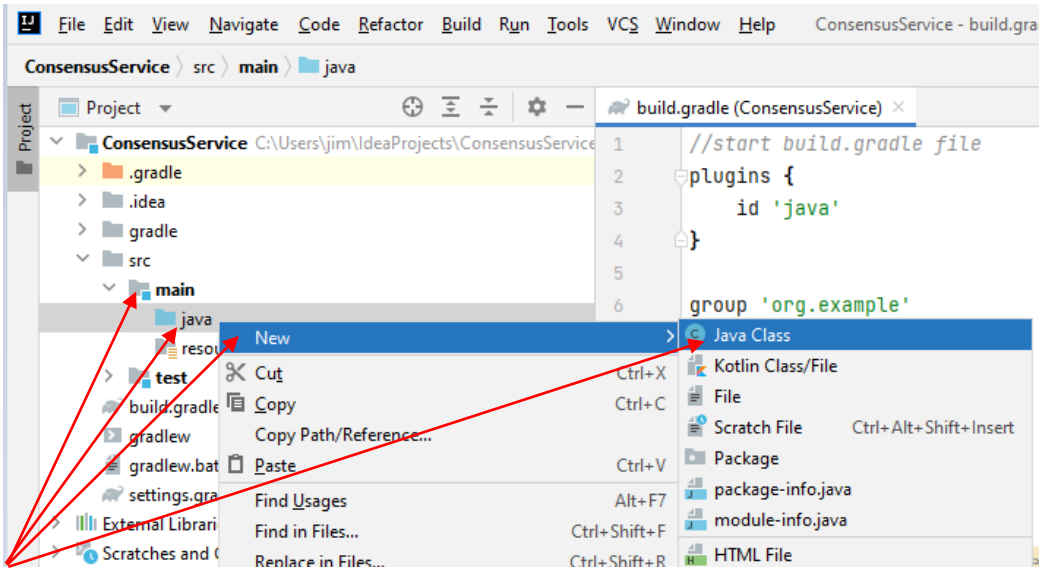
Make sure you update the build.gradle file is updated from this GitHub repository. Use your own .env file with your own account ID and private key. An example is included in this GitHub repository.



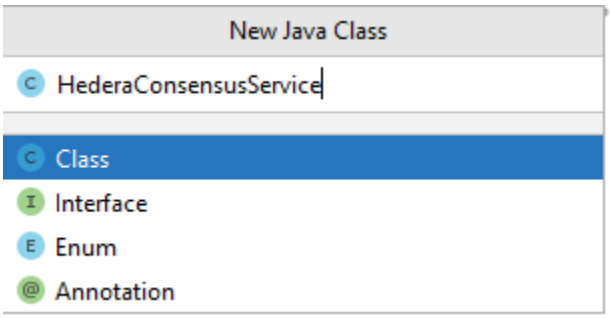
Create a new class called HederaConsensusService

In the project expand \src\main

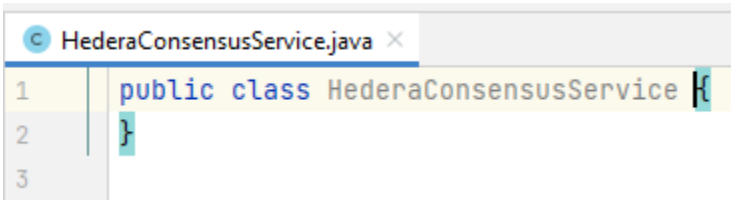
Right click on \src\main\java, and click New -> Java Class



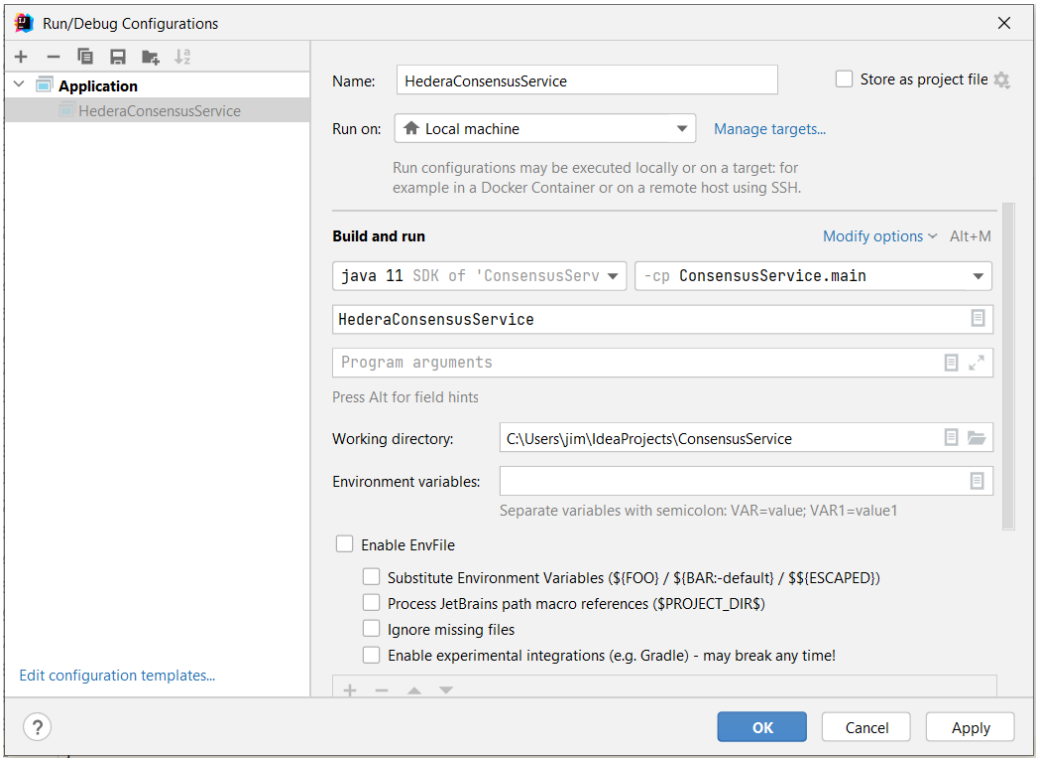
Enter HederaConsensusService and tap the enter key.



The HederaConsensusService class file will open.



Validate project configurations, see **IntelliJ step 9**. Project configurations will resemble the image below.



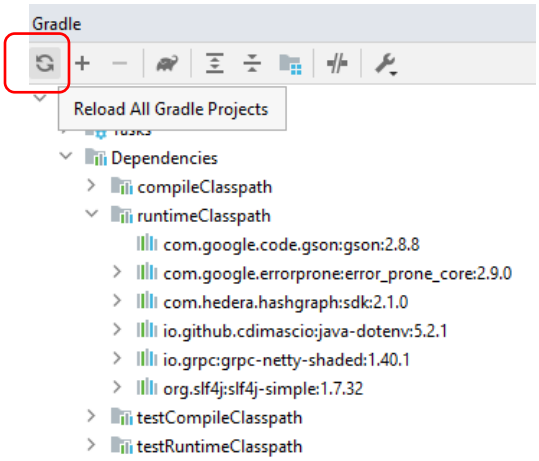

```
import com.hedera.hashgraph.sdk.*;
import com.google.protobuf.ByteString;
import io.github.cdimascio.dotenv.Dotenv;
import java.util.concurrent.TimeoutException;
```

Create the main method as shown below.

```
public static void main(String[] args) throws TimeoutException,
PrecheckStatusException, ReceiptStatusException {


}
```

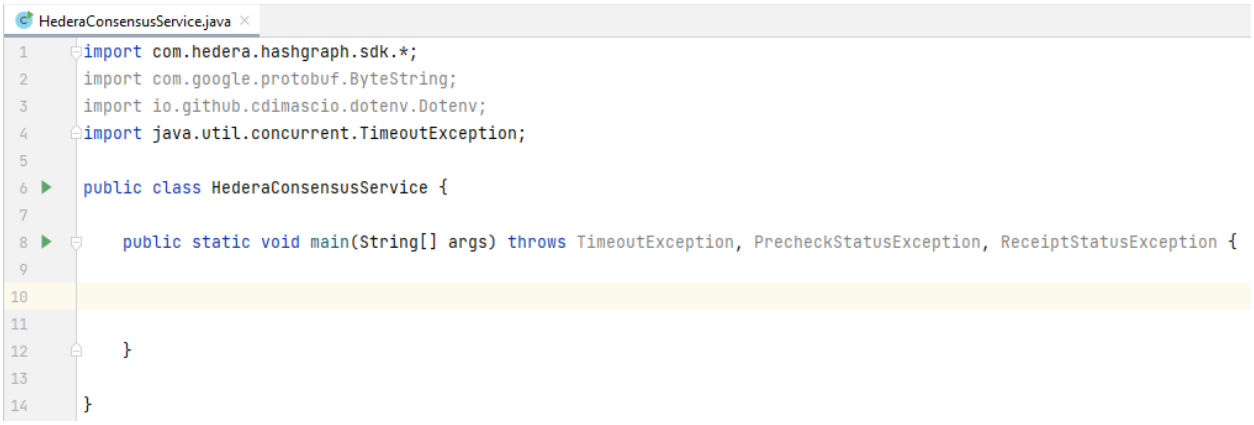
Update the Gradle dependencies.

Click the Gradle button in the upper right corner. Click the **Refresh Gradle Project** button.



Close the Gradle tools, click the  button in the upper right corner.

The class file should look similar to the image below.



1. Obtain your account ID and private key form the .env file
2. Create a client from the Hedera testnet,
3. Set your account ID and private for the testnet client.
4. Use TopicCreateTransaction to create a transaction object and call setSubmitKey, setTopicMemo, and setAdminKey
5. On the transaction object, call execute client. This creates a transaction response object.
6. Get a transaction receipt. Call the getReceipt method on the transaction response object.
7. To return the topic ID, call the topicID method on the receipt object and output it to the console.

Refer to the solution on GitHub: HederaConsensusService_part1.txt, https://github.com/theblockchainacademy/Hedera/blob/main/Hedera_5_ConsensusServiceExercise5/HederaConsensusService_part1.txt

The output will appear similar to below

```
3:33:09 PM: Executing task ':HederaConsensusService.main()'...

> Task :compileJava
> Task :processResources NO-SOURCE
> Task :classes

> Task :HederaConsensusService.main()
[hedera-sdk-6] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.3 in 250 ms after failure during attempt #1: OK
[hedera-sdk-0] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.3 in 500 ms after failure during attempt #2: OK
[hedera-sdk-4] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.3 in 1000 ms after failure during attempt #3: OK

The new topic ID is 0.0.2908164

Deprecated Gradle features were used in this build, making it incompatible with Gradle 8.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

See https://docs.gradle.org/7.1/userguide/command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 4s
2 actionable tasks: 2 executed
3:33:14 PM: Task execution finished ':HederaConsensusService.main()'.
```

8. Set a message for the topic ID. Use TopicMessageSubmitTransaction set the topic ID to the newly generated topic ID using the setTopicID method. Set the message using the setMessage method.
9. This will create a TopicMessageSubmitTransaction object.
10. Use the TopicMessageSubmitTransaction object and call the getMessage method.
11. Output the message to the console and confirm it was the message passed in.
12. Sign in with the client and start to submit transaction to a Hedera network, transaction consensus status.
13. Create and submit an account info query to the network. Output the results to the console.
14. Use TopicInfoQuery to query all topic metadata. Pass in t to the console.

Refer to the solution on GitHub: HederaConsensusService_part2.txt, https://github.com/theblockchainacademy/Hedera/blob/main/Hedera_5_ConsensusServiceExercise5/HederaConsensusService_part2.txt

The output will appear similar to below

```
3:37:59 PM: Executing task ':HederaConsensusService.main()'...

> Task :compileJava
> Task :processResources NO-SOURCE
> Task :classes

> Task :HederaConsensusService.main()
[hedera-sdk-6] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.4 in 250 ms after failure during attempt #1: OK
[hedera-sdk-0] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.4 in 500 ms after failure during attempt #2: OK
[hedera-sdk-4] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.4 in 1000 ms after failure during attempt #3: OK

The new topic ID is 0.0.2908212

The first msg:  <ByteString@1755e85b size=48 contents="Good Day and welcome to Hedera Consensus Service">
[hedera-sdk-7] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.6 in 250 ms after failure during attempt #1: OK
[hedera-sdk-3] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.6 in 500 ms after failure during attempt #2: OK
[hedera-sdk-7] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.6 in 1000 ms after failure during attempt #3: OK

The transaction consensus status is SUCCESS
TopicInfo{topicId=0.0.2908212, topicMemo=First Message, runningHash=[-128, -56, 91, -14, 66, -10, -68, 127, -124, -37, -60, 113, -124, -69, -65, -27, -25, 123, 59,

Deprecated Gradle features were used in this build, making it incompatible with Gradle 8.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

See https://docs.gradle.org/7.1/userguide/command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 7s
2 actionable tasks: 2 executed
3:38:07 PM: Task execution finished ':HederaConsensusService.main()'.
```

15. Use TopicUpdateTransaction set the topic ID, the submit and admin keys using the same keys from the previous part.
16. Get the receipt, status and return the status to the console.
17. Use TopicInfoQuery to output the updated metadata to the console.

Refer to the solution on GitHub: [HederaConsensusService_part3.txt](), [https://github.com/theblockchainacademy/Hedera/blob/main/Hedera_5_ConsensusServiceExercise5/HederaConsensusService_part3.txt]()

The output will appear similar to below

```
> Task :classes

> Task :HederaConsensusService.main()
[hedera-sdk-7] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.7 in 250 ms after failure during attempt #1: OK
[hedera-sdk-2] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.7 in 500 ms after failure during attempt #2: OK
[hedera-sdk-6] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.7 in 1000 ms after failure during attempt #3: OK

The new topic ID is 0.0.2908257

The first msg:  <ByteString@1755e85b size=48 contents="Good Day and welcome to Hedera Consensus Service">
[hedera-sdk-1] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.4 in 250 ms after failure during attempt #1: OK
[hedera-sdk-2] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.4 in 500 ms after failure during attempt #2: OK
[hedera-sdk-7] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.4 in 1000 ms after failure during attempt #3: OK

The transaction consensus status is SUCCESS
TopicInfo{topicId=0.0.2908257, topicMemo=First Message, runningHash=[-72, 91, 100, 51, -118, -52, -53, 54, -104, 91, -50, 51, 16, -22, -115, 119, -10, 22, 100, -34,

Update Topic ----------------------------------------------------

[hedera-sdk-0] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.5 in 250 ms after failure during attempt #1: OK
[hedera-sdk-2] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.5 in 500 ms after failure during attempt #2: OK
[hedera-sdk-7] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.5 in 1000 ms after failure during attempt #3: OK

The transaction consensus status is SUCCESS

The message after the update:  <ByteString@1755e85b size=48 contents="Good Day and welcome to Hedera Consensus Service">
TopicInfo{topicId=0.0.2908257, topicMemo=New message memo update, runningHash=[-72, 91, 100, 51, -118, -52, -53, 54, -104, 91, -50, 51, 16, -22, -115, 119, -10, 22,

Deprecated Gradle features were used in this build, making it incompatible with Gradle 8.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

See https://docs.gradle.org/7.1/userguide/command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 9s
2 actionable tasks: 2 executed
3:41:34 PM: Task execution finished ':HederaConsensusService.main()'.
```

18. Use the TopicMessageSubmitTransaction to submit a new message a to the same topic ID .
19. Get the new message for the topic ID and return the new message and metadata to the console.

Refer to the solution on GitHub: [HederaConsensusService_part4.txt](), https://github.com/theblockchainacademy/Hedera/blob/main/Hedera_5_ConsensusServiceExercise5/HederaConsensusService_part4.txt

The output will appear similar to below

```
[hedera-sdk-0] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.3 in 500 ms after failure during attempt #2: OK
[hedera-sdk-5] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.3 in 1000 ms after failure during attempt #3: OK

The new topic ID is 0.0.2908270

The first msg:  <ByteString@3f6db3fb size=48 contents="Good Day and welcome to Hedera Consensus Service">
[hedera-sdk-1] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.6 in 250 ms after failure during attempt #1: OK
[hedera-sdk-3] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.6 in 500 ms after failure during attempt #2: OK

The transaction consensus status is SUCCESS
TopicInfo{topicId=0.0.2908270, topicMemo=First Message, runningHash=[76, 90, -18, 97, -64, -116, 22, -46, -30, 107, 60, 68, -72, -97, 111, -48, -98, -87, -84, -28,

Update Topic ----------------------------------------------------

[hedera-sdk-5] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.4 in 250 ms after failure during attempt #1: OK
[hedera-sdk-0] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.4 in 500 ms after failure during attempt #2: OK
[hedera-sdk-4] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.4 in 1000 ms after failure during attempt #3: OK

The transaction consensus status is SUCCESS

The message after the update:  <ByteString@3f6db3fb size=48 contents="Good Day and welcome to Hedera Consensus Service">
TopicInfo{topicId=0.0.2908270, topicMemo=New message memo update, runningHash=[76, 90, -18, 97, -64, -116, 22, -46, -30, 107, 60, 68, -72, -97, 111, -48, -98, -87,

New message for the same topic ----------------------------------------------


The new msg:  <ByteString@2a2c13a8 size=46 contents="Time to work with the Hedera Consensus Service">
TopicInfo{topicId=0.0.2908270, topicMemo=New message memo update, runningHash=[76, 90, -18, 97, -64, -116, 22, -46, -30, 107, 60, 68, -72, -97, 111, -48, -98, -87,

Deprecated Gradle features were used in this build, making it incompatible with Gradle 8.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

See https://docs.gradle.org/7.1/userguide/command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 8s
2 actionable tasks: 2 executed
3:43:48 PM: Task execution finished ':HederaConsensusService.main()'.
```

20. Delete the topic from the Hedera Network. Use TopicDeleteTransaction and call the setTopicId method with the current topic ID passed into it.
21. Get the TransactionResponse and TransactionReceipt.
22. Return the transaction receipt and the transaction status.

Refer to the solution on GitHub: [HederaConsensusService_part5.txt](), https://github.com/theblockchainacademy/Hedera/blob/main/Hedera_5_ConsensusServiceExercise5/HederaConsensusService_part5.txt

The output will appear similar to below.

```
> Task :HederaConsensusService.main()
[hedera-sdk-7] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.4 in 250 ms after failure during attempt #1: OK
[hedera-sdk-3] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.4 in 500 ms after failure during attempt #2: OK
[hedera-sdk-7] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.4 in 1000 ms after failure during attempt #3: OK

The new topic ID is 0.0.2908271

The first msg:  <ByteString@3f6db3fb size=48 contents="Good Day and welcome to Hedera Consensus Service">
[hedera-sdk-3] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.7 in 250 ms after failure during attempt #1: OK
[hedera-sdk-6] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.7 in 500 ms after failure during attempt #2: OK
[hedera-sdk-2] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.7 in 1000 ms after failure during attempt #3: OK

The transaction consensus status is SUCCESS
TopicInfo{topicId=0.0.2908271, topicMemo=First Message, runningHash=[94, -25, 118, -118, -9, 10, -103, 119, 45, -90, -18, -61, 120, -107, 26, -80, -96, 61, -67, 120,

Update Topic ----------------------------------------------------

[hedera-sdk-5] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.3 in 250 ms after failure during attempt #1: OK
[hedera-sdk-0] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.3 in 500 ms after failure during attempt #2: OK
[hedera-sdk-3] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.3 in 1000 ms after failure during attempt #3: OK

The transaction consensus status is SUCCESS

The message after the update:  <ByteString@3f6db3fb size=48 contents="Good Day and welcome to Hedera Consensus Service">
TopicInfo{topicId=0.0.2908271, topicMemo=New message memo update, runningHash=[94, -25, 118, -118, -9, 10, -103, 119, 45, -90, -18, -61, 120, -107, 26, -80, -96, 61,

New message for the same topic ----------------------------------------------


The new msg:  <ByteString@b6b1987 size=46 contents="Time to work with the Hedera Consensus Service">
TopicInfo{topicId=0.0.2908271, topicMemo=New message memo update, runningHash=[94, -25, 118, -118, -9, 10, -103, 119, 45, -90, -18, -61, 120, -107, 26, -80, -96, 61,

Delete a topic from the network ----------------------------------------------

[hedera-sdk-1] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.5 in 250 ms after failure during attempt #1: OK
[hedera-sdk-3] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.5 in 500 ms after failure during attempt #2: OK
[hedera-sdk-6] WARN com.hedera.hashgraph.sdk.TransactionReceiptQuery - Retrying node 0.0.5 in 1000 ms after failure during attempt #3: OK
The delete consensus: SUCCESS

Deprecated Gradle features were used in this build, making it incompatible with Gradle 8.0.
```

Additional Hands-on Exercises

Create a Token using the JavaScript API

Prerequisites

In order to the JavaScript API Node.js must be installed.

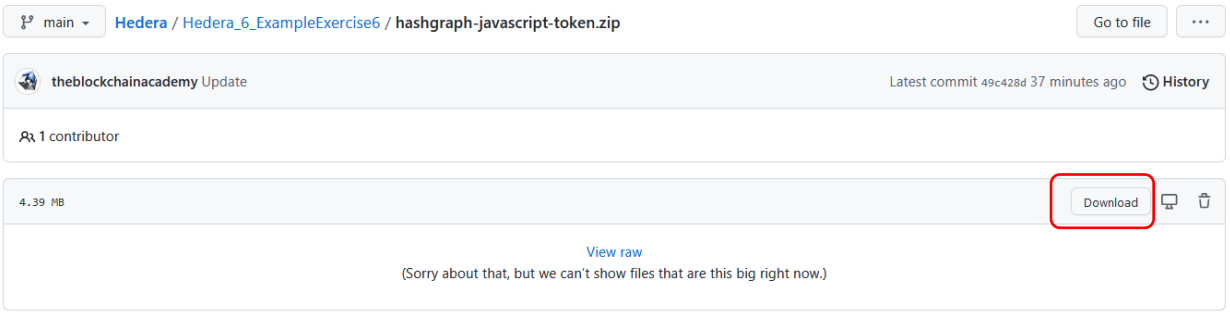
To install Node.js follow the steps on this site: https://nodejs.org/en/download/

Refer to Hedera documents for JavaScript: https://docs.hedera.com/guides/docs/sdks/tokens/define-a-token

Create a local directory named: hedera-token-js

Download the Create a token JavaScript application: hashgraph-javascript-token.zip, https://github.com/theblockchainacademy/Hedera/blob/main/Hedera_6_ExampleExercise6/hashgraph-javascript-token.zip

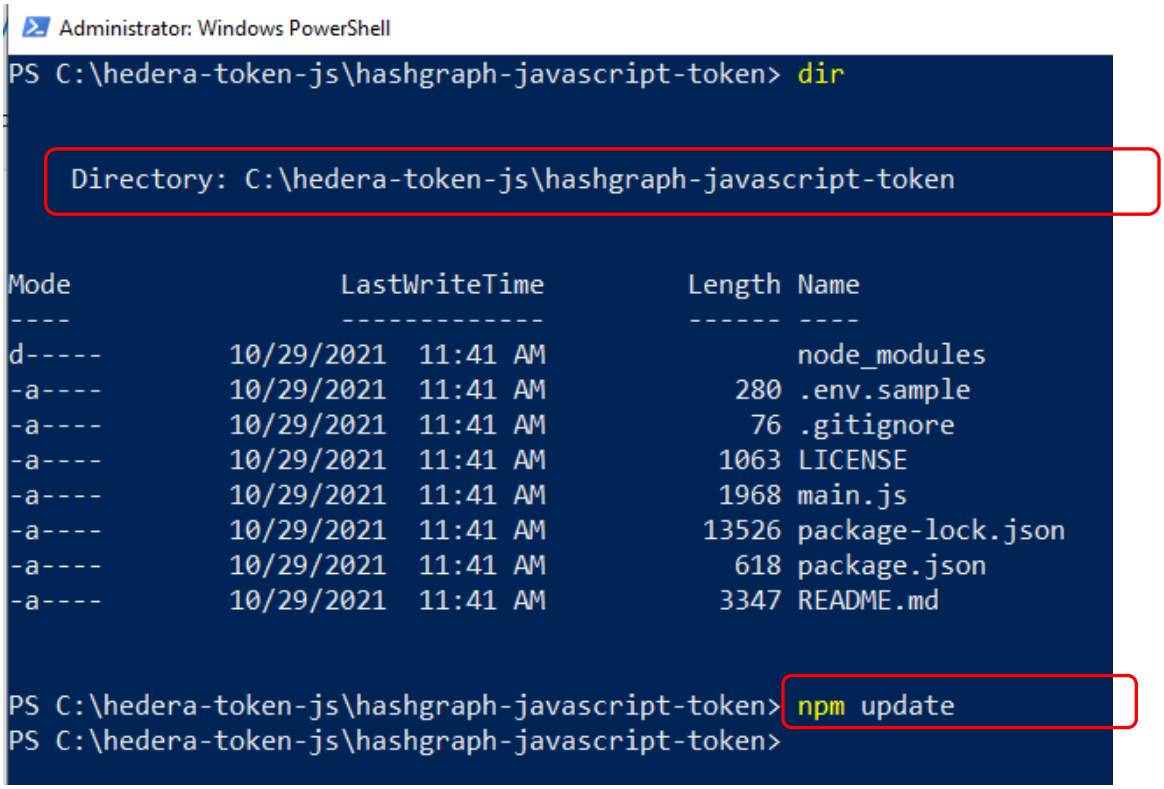
Click **Download**



The application is stored in a zip file, it must be extracted to local directory: hedera-token-js.

The application will be contained in directory: \hedera-token-js\hashgraph-javascript-token

Run: `npm update`



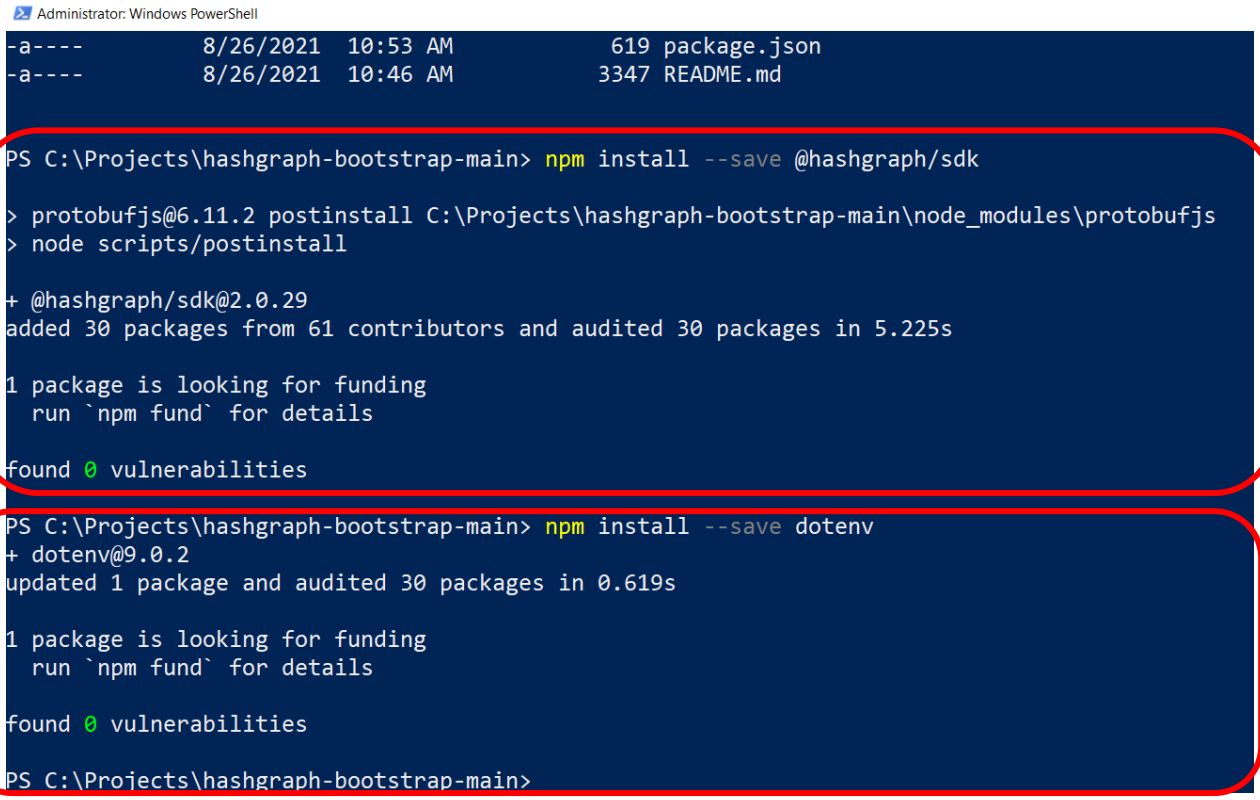
Install the Hedera JavaScipt SDK package, run:

`npm install --save @hashgraph/sdk`, or
`npm install --save @hashgraph/sdk@latest`

Install the dotenv package: `npm install --save dotenv`, or
`npm install --save dotenv@latest`

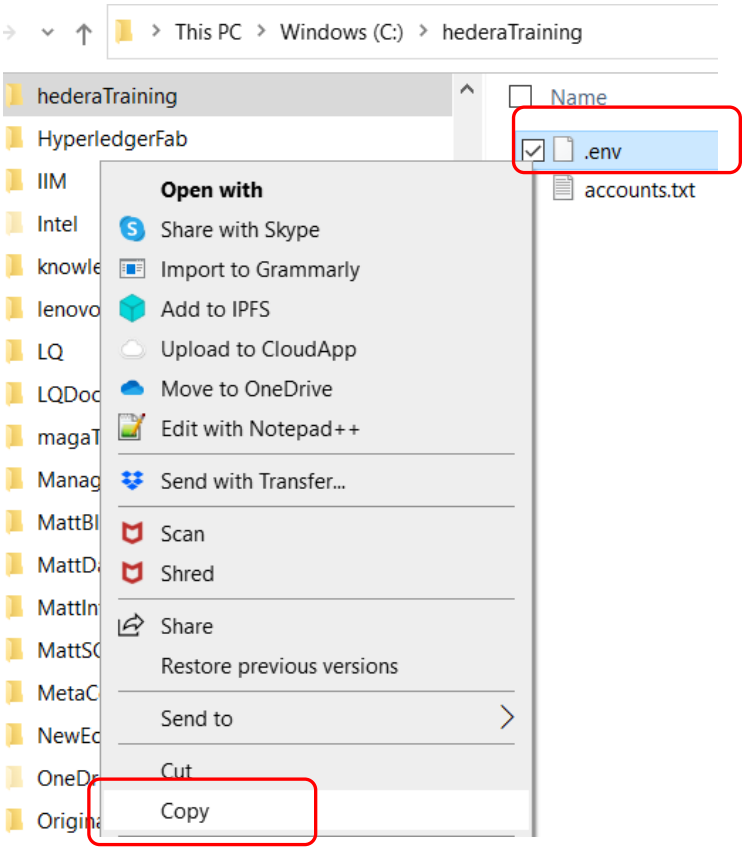
For more information about `npm install` see: https://docs.npmjs.com/cli/v7/commands/npm-install

The input and output will resemble the image below.

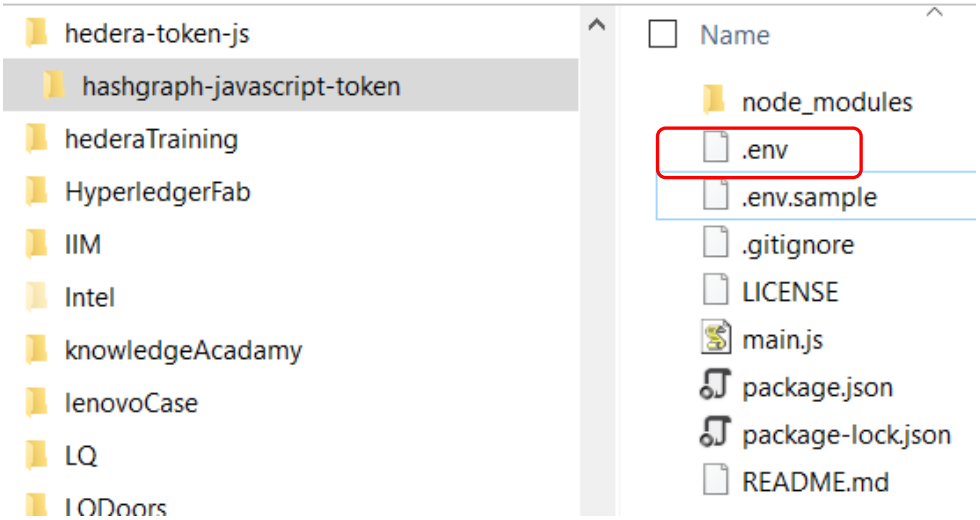


Copy the .env file from the hederaTraining directory to the hedera-token-js\hashgraph-javascript-token directory.

Use your own .env file with your own account ID and private key. An example is included in this GitHub repository.



Paste the .env file to the hedera-token-js\hashgraph-javascript-token directory



Execute the application, run: node .\main.js

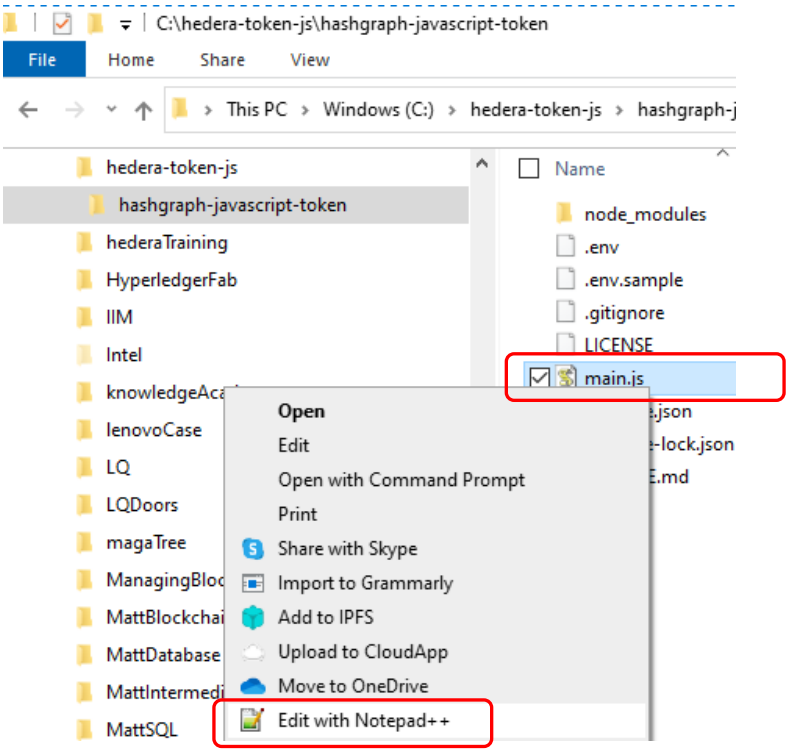
```
Administrator: Windows PowerShell
PS C:\hedera-token-js\hashgraph-javascript-token> node .\main.js
```

The output will resemble the results below. The more the application is run, the more tokens will be created.

```
Administrator: Windows PowerShell
PS C:\hedera-token-js\hashgraph-javascript-token> node .\main.js
Greetings earthling....
This is a JavaScript Hedera API application
Attempting to get token info from the Hedera API...
The token balance(s) for this account: {"0.0.2299317":"500","0.0.2299333":"500"
,"0.0.3039821":"500"}

Creating the new TBA Token

new TBA token id:  0.0.3039889   ;)
PS C:\hedera-token-js\hashgraph-javascript-token>
```

Edit the JavaScript application.

Navigate to the hedera-token-js\hashgraph-javascript-token directory. Open the main.js file for editing.



Scroll to the TokenCreateTransaction() near or on line 55.

Change the name of the token.

Change the initial supply to: 75

Enter “hello” into the console log output.

```
55      var createTokenTx = await new TokenCreateTransaction()
56          .setTokenName("TheNewTBAToken")
57          .setTokenSymbol("TBAT")
58          .setDecimals(5)
59          .setInitialSupply(75)
60          //.setTreasuryAccountId("0.0.2084028")
61          .setTreasuryAccountId(moperatorId)
62          .execute(client);
```

```
63
64      var createReceipt = await createTokenTx.getReceipt(client);
65      var newTokenId = createReceipt.tokenId;
66
67      console.log('new hello TBA token id: ', newTokenId.toString(), '  ;)');
68
```

Save and close the main.js file.

Execute the application, run: node .\main.js

```
Administrator: Windows PowerShell
PS C:\hedera-token-js\hashgraph-javascript-token> node .\main.js
```

Run the application **at least twice** to output the details of all tokens.
The resulting output will appear similar to the image below.

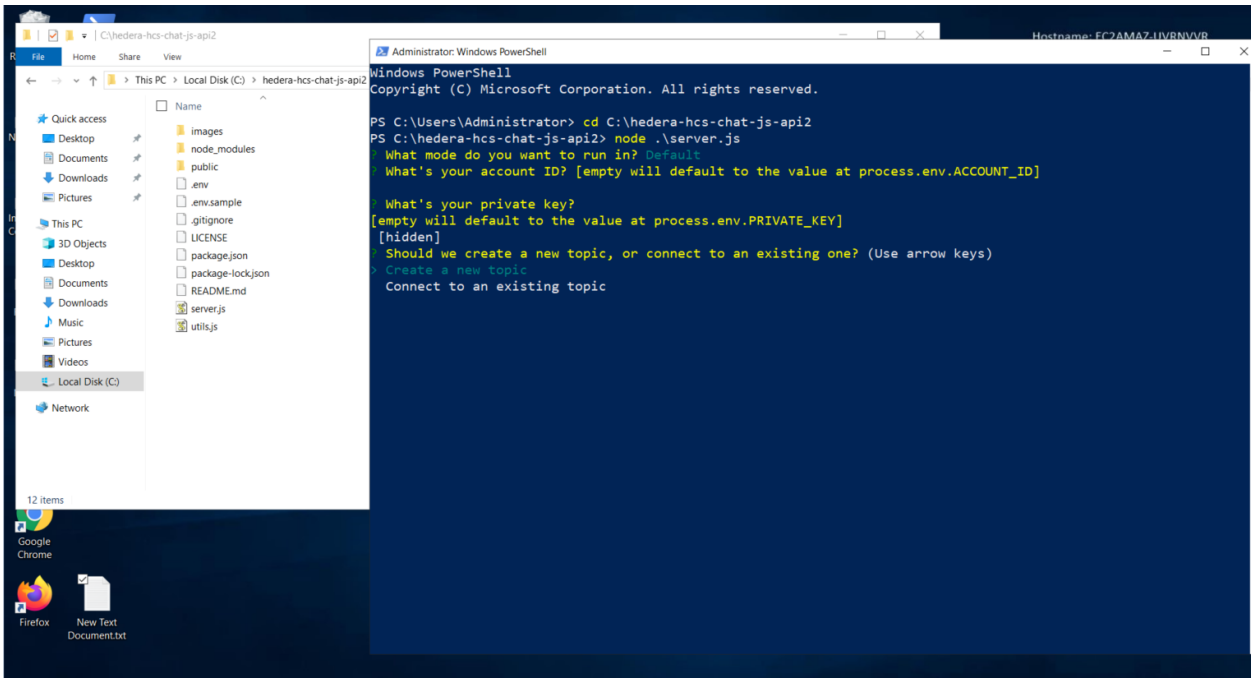
```
 "0.0.3039903":"75"}

Creating the new TBA Token

new hello TBA token id:  0.0.3039904   ;)
PS C:\hedera-token-js\hashgraph-javascript-token>
```

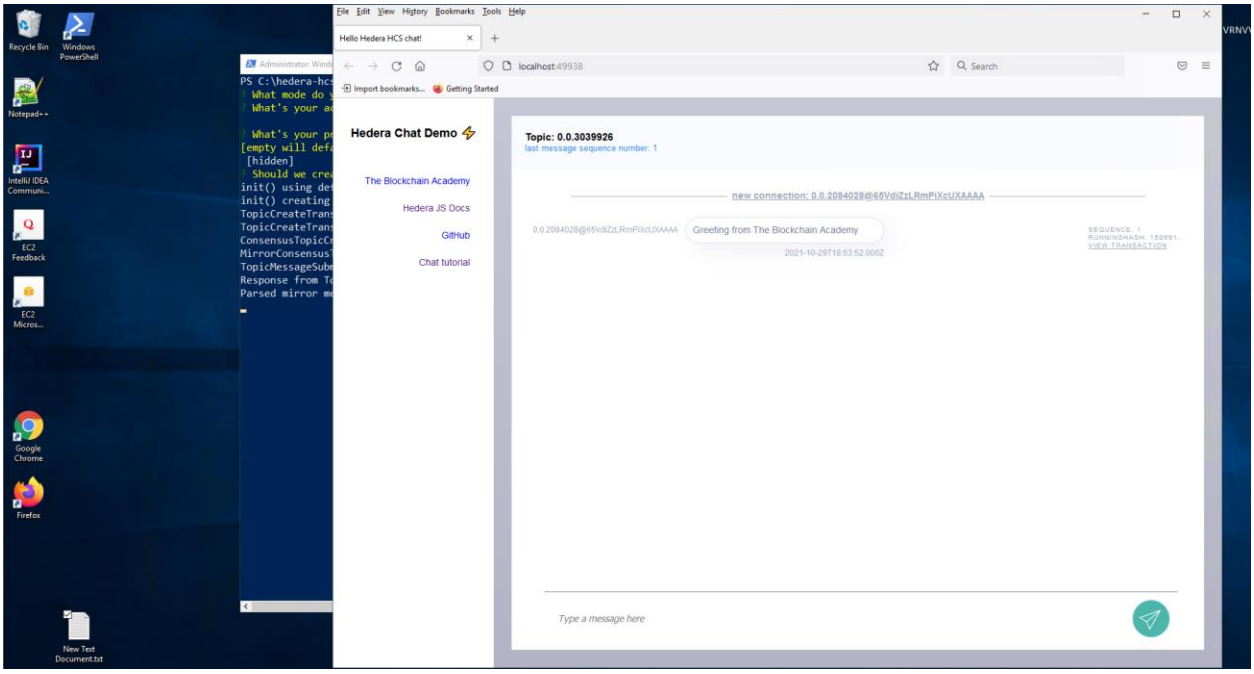
The Hedera Consensus Service Chat application

Use the code and steps on GitHub to install and configure the Hedera Consensus Service Chat application.

The Hedera Consensus Service Chat application is a node.js web application.

https://github.com/hashgraph/hedera-hcs-chat-js





If you have any issues setting up, or running the Hedera HCS chat application, contact: learner-support@blockchainhub360.com

A link for the application can be sent to you to run it.

Final Exercise

Create an application that satisfies the requirements below.
Send the output, including your name to email: learner-support@blockchainhub360.com

Create a great Hedera application based on what you have learned. Greatly elaborate on the solution. All solutions are posted anonymously.

Requirement 1:

Create accounts for: Abbey, Bill, and Chuck

Create a fungible token with an initial supply of 100. Make Alice the treasury account. Name the fungible token.

Send tokens from: Abbey, Bill, and Chuck.

Return the user accounts, the token IDs and names, and the balance of each user's account.

Requirement 2:

Create an NFT, name it, mint it, and add image metadata.

Send the NFTs to all three account users.

Return the NFT ID, and the balance of each NFT for each user.

Requirement 3:

Upload a file to Hedera. Return the file ID

Set the maximum transaction fee for the file.

Output file contents once it is on the Hedera.

Delete the file.

Requirement 4:

Use the Hedera Consensus Service to create a topic, modify the topic and output the topic message.

Again, send your name and a screenshot of all output to: learner-support@blockchainhub360.com