**Background**

After waiting for years and passing several tests, the Martian Aerospace Agency selected you to become part of the first human colony on Mars. As a prominent fintech professional, they chose you to lead a project developing a monetary system for the new Mars colony. You decided to base this new system on blockchain technology and to define a new cryptocurrency named **KaseiCoin**. (Kasei means Mars in Japanese.)

KaseiCoin will be a fungible token that’s ERC-20 compliant. You’ll launch a crowdsale that will allow people who are moving to Mars to convert their earthling money to KaseiCoin.

**Files**

Download the following files to help you get started:

[KaseiCoin.sol](/utoronto-bootcamp/UTOR-VIRT-FIN-PT-02-2022-U-LOL/-/blob/main/Homeworks/21-Advanced-Solidity/Instructions/Starter_Code/KaseiCoin.sol)

[KaseiCoinCrowdsale.sol](/utoronto-bootcamp/UTOR-VIRT-FIN-PT-02-2022-U-LOL/-/blob/main/Starter_Code/KaseiCoinCrowdsale.com)

**Instructions**

The steps for this assignment are divided into the following subsections:

1. Create the KaseiCoin Token Contract
2. Create the KaseiCoin Crowdsale Contract
3. Create the KaseiCoin Deployer Contract
4. Deploy and Test the Crowdsale on a Local Blockchain
5. Optional: Extend the Crowdsale Contract by Using OpenZeppelin

**Note:** You can choose whether to complete the optional section. It’s designed to further your professional growth and development but won’t be graded as part of this assignment. If you choose to complete this section, you’ll use OpenZeppelin to extend the functionality of your crowdsale contract by adding time restrictions, refund capabilities, and a cap for the number of tokens that can be created. If you have any questions about how to complete the optional section, please reach out to your instructional team.

Note that the provided starter files for this homework assignment contain a pragma statement for Solidity version 0.5.0. You’ll use the starter files to complete the steps in the subsections.

In the subsections, you’ll create a fungible token that’s ERC-20 compliant. This token will be minted by using a Crowdsale contract from the OpenZeppelin Solidity library.

The crowdsale contract that you create will manage the entire crowdsale process. This process will allow users to send ether to the contract and receive KaseiCoin tokens, or **KAI**, in return. Your contract will automatically mint the tokens and distribute them to a buyer in one transaction.

Note that you’ll record a short video or animated GIF or take several screenshots that show the deployed contract in action.

In the README.md file of your GitHub repository for this homework assignment, you’ll create a section named Evaluation Evidence. In this section, you’ll share screenshots of your work from each subsection of the assignment.

**Step 1: Create the KaseiCoin Token Contract**

In this subsection, you’ll create a smart contract that defines KaseiCoin as an ERC-20 token. To do so, complete the following steps:

1. Import the provided KaseiCoin.sol starter file into the Remix IDE.
2. Import the following contracts from the OpenZeppelin library:
   * ERC20
   * ERC20Detailed
   * ERC20Mintable
3. Define a contract for the KaseiCoin token, and name it KaseiCoin. Have the contract inherit the three contracts that you just imported from OpenZeppelin.
4. Inside your KaseiCoin contract, add a constructor with the following parameters: name, symbol, and initial\_supply.
5. As part of your constructor definition, add a call to the constructor of the ERC20Detailed contract, passing the parameters name, symbol, and 18. (Recall that 18 is the value for the decimals parameter.)
6. Compile the contract by using compiler version 0.5.0.
7. Check for any errors, and debug them as needed.
8. Take a screenshot of the successful compilation of the contract, and add it to the Evaluation Evidence section of the README.md file for your GitHub repository.

**Step 2: Create the KaseiCoin Crowdsale Contract**

In this subsection, you’ll define the KaseiCoin crowdsale contract. To do so, complete the following steps:

1. Import the provided KaseiCoinCrowdsale.sol starter code into the Remix IDE.
2. Have this contract inherit the following OpenZeppelin contracts:
   * Crowdsale
   * MintedCrowdsale
3. In the KaisenCoinCrowdsale constructor, provide parameters for all the features of your crowdsale, such as rate, wallet (where to deposit the funds that the token raises), and token. Configure these parameters as you want for your KaseiCoin token.
4. Compile the contract by using compiler version 0.5.0.
5. Check for any errors, and debug them as needed.
6. Take a screenshot of the successful compilation of the contract, and add it to the Evaluation Evidence section of the README.md file for your GitHub repository.

**Step 3: Create the KaseiCoin Deployer Contract**

In this subsection, you’ll create the KaseiCoin deployer contract. Start by uncommenting the KaseiCoinCrowdsaleDeployer contract in the provided KaseiCoinCrowdsale.sol starter code.

Next, in the KaseiCoinCrowdsaleDeployer contract, you’ll add variables to store the addresses of the KaseiCoin and KaseiCoinCrowdsale contracts, which this contract will deploy. Finally, you’ll complete the KaseiCoinCrowdsaleDeployer contract. To do so, complete the following steps:

1. Create an address public variable named kasei\_token\_address, which will store the KaseiCoin address once that contract has been deployed.
2. Create an address public variable named kasei\_crowdsale\_address, which will store the KaseiCoinCrowdsale address once that contract has been deployed.
3. Add the following parameters to the constructor for the KaseiCoinCrowdsaleDeployer contract: name, symbol, and wallet.
4. Inside of the constructor body (that is, between the braces), complete the following steps:
   * Create a new instance of the KaseiCoinToken contract.
   * Assign the address of the KaseiCoin token contract to the kasei\_token\_address variable. (This will allow you to easily fetch the token's address later.)
   * Create a new instance of the KaseiCoinCrowdsale contract by using the following parameters:
     + The rate parameter: Set rate equal to 1 to maintain parity with ether.
     + The wallet parameter: Pass in wallet from the main constructor. This is the wallet that will get paid all the ether that the crowdsale contract raises.
     + The token parameter: Make this the token variable where KaseiCoin is stored.
   * Assign the address of the KaseiCoin crowdsale contract to the kasei\_crowdsale\_address variable. (This will allow you to easily fetch the crowdsale’s address later.)
   * Set the KaseiCoinCrowdsale contract as a minter.
   * Have the KaseiCoinCrowdsaleDeployer renounce its minter role.
5. Compile the contract by using compiler version 0.5.0.
6. Check for any errors, and debug them as needed.
7. Take a screenshot of the successful compilation of the contract, and add it to the Evaluation Evidence section of the README.md file for your Git repository.

**Step 4: Deploy and Test the Crowdsale on a Local Blockchain**

In this subsection, you’ll deploy the crowdsale to a local blockchain. You’ll then perform a real-world, preproduction test of your crowdsale. To do so, complete the following steps:

**Important:** Record a short video or take screenshots that illustrate the following steps as evidence of your deployed crowdsale contract.

1. Deploy the crowdsale to a local blockchain by using Remix, MetaMask, and Ganache.
2. Test the functionality of the crowdsale by using test accounts to buy new tokens and then checking the balances of those accounts.
3. Review the total supply of minted tokens and the amount of wei that the crowdsale contract has raised.

**Optional: Extend the Crowdsale Contract by Using OpenZeppelin**

In this optional subsection, you can extend the crowdsale contract to enhance its functionality. To do so, you’ll use the following OpenZeppelin contracts:

* The CappedCrowdsale contract: Allows you to cap the total amount of ether that your crowdsale can raise.
* The TimedCrowdsale contract: Allows you to set a time limit for your crowdsale by adding an opening time and a closing time.
* The RefundablePostDeliveryCrowdsale contract: Allows you to refund your investors. Every time that you launch a crowdsale, you set a goal amount of ether to raise. If you don’t reach the goal, it’s a common practice to refund your investors.

**Hint:** We encourage you to read more about these contracts on the [Crowdsales page](https://docs.openzeppelin.com/contracts/2.x/crowdsales) of the OpenZeppelin documentation.

To enhance your KaseiCoin crowdsale with this added functionality, complete the following steps:

1. Import the three OpenZeppelin contracts just described into the KaseiCoinCrowdsale.sol contract by using the following code:
2. import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v2.5.0/contracts/crowdsale/validation/CappedCrowdsale.sol";
3. import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v2.5.0/contracts/crowdsale/validation/TimedCrowdsale.sol";

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v2.5.0/contracts/crowdsale/distribution/RefundablePostDeliveryCrowdsale.sol";

1. In addition to the Crowdsale and MintedCrowdsale contracts, which your contract previously inherited from OpenZeppelin, have your KaseiCoinCrowdsale contract inherit the following three contracts, which you just imported:
   * CappedCrowdsale
   * TimedCrowdsale
   * RefundablePostDeliveryCrowdsale
2. In the KaseiCoinCrowdsale constructor, add the following parameters:
   * The uint goal parameter: The amount of ether that you hope to raise during the crowdsale—that is, the goal of the crowdsale.
   * The uint open parameter: The opening time for the crowdsale.
   * The uint close parameter: The closing time for the crowdsale.
3. Complete the KaseiCoinCrowdsale constructor code by adding calls to the new contracts, as the following code shows:
4. constructor(
5. uint256 rate, // rate in TKNbits
6. address payable wallet, // sale beneficiary
7. KaseiCoin token, // the KaseiCoin itself that the KaseiCoinCrowdsale will work with
8. uint goal, // the crowdsale goal
9. uint open, // the crowdsale opening time
10. uint close // the crowdsale closing time
11. ) public
12. Crowdsale(rate, wallet, token)
13. CappedCrowdsale(goal)
14. TimedCrowdsale(open, close)
15. RefundableCrowdsale(goal)
16. {
17. // constructor can stay empty

}

**Important:** The RefundablePostDeliveryCrowdsale contract itself inherits the RefundableCrowdsale contract, which requires a goal parameter. So in addition to the others, you must call the RefundableCrowdsale constructor from your KaseiCoinCrowdsale constructor. The RefundablePostDeliveryCrowdsale contract doesn’t have its own constructor, which is why we use the RefundableCrowdsale constructor that it inherits.

If you forget to call the RefundableCrowdsale constructor, RefundablePostDeliveryCrowdsale will fail. This is because it doesn't have its own constructor, so it relies on the RefundableCrowdsale constructor.

1. Update the KaseiCoinCrowdsaleDeployer contract to allow the deployment of the updated crowdsale contract. In the constructor of the deployer contract, add a new uint parameter named goal that will allow you to set the crowdsale goal.
2. You previously added an instance of the KaseiCoinCrowdsale contract to the KaseiCoin deployer contract. Because we modified the KaseiCoinCrowdsale contract to support new functionality, you now need to update your previous code with the following code:

KaseiCoinCrowdsale kasei\_crowdsale = new KaseiCoinCrowdsale (1, wallet, token, goal, now, now + 24 weeks);

Note that in the preceding code, you added values for the three new parameters. The goal parameter represents the amount of ether to raise during the crowdsale. The now parameter represents the crowdsale opening time. And, now + 24 weeks represents the closing time.

The now function returns the current Ethereum block timestamp in the form of seconds since the Unix epoch. The **Unix epoch** (also known as **Unix time**, **POSIX time**, or **Unix timestamp**) is an integer representing the number of seconds that have elapsed since January 1, 1970 (at midnight coordinated universal time, UTC), not counting leap seconds.

1. Compile and test the updated contract by completing following steps:
   * Send ether to the crowdsale from a different account (that is, not the same account that’s raising funds). Then, once you confirm that the crowdsale works as expected, try to add the token to your wallet and to test a transaction.
   * Set the close time to now + 5 minutes (for a shorter crowdsale) or to any timeline that you'd like to test.
   * When sending ether to the contract, make sure that you meet the goal of the contract. Then finalize the sale by using the finalize function of the Crowdsale contract. Note that to finalize the sale, isOpen must return false (isOpen comes from TimedCrowdsale and checks whether the close time has passed). If you set the goal to 300 ether, for example, you might need to have multiple accounts buy tokens to meet the goal. If you run out of prefunded accounts in Ganache, you can create a new workspace.
   * Review your tokens in MetaMask. To do so in MetaMask, click Add Token, click Custom Token, and then enter the address of the token contract. Make sure to buy larger amounts of tokens to get the denomination to appear in your wallet as more than a few wei worth.
2. Create a GitHub repository and a README.md file that explains the process for buying KaseiCoin.

Make sure that your README.md file includes screenshots that illustrate the functionality of your contracts as the earlier instructions detailed.

You can also record your interactions with the executed contract as a short video or an animated GIF. To record a video, you can use the following tools:

* If you’re working on macOS, you can create a screen recording by using the built-in QuickTime player. To learn more about this tool, refer to [Use QuickTime Player](https://support.apple.com/en-us/HT208721#quicktime) in the Apple Support documentation.
* If you’re working on Windows 10, you can create a screen recording by using the built-in Xbox Game Bar. To learn more about this tool, refer to [Use Xbox Game Bar to capture game clips and screenshots on Windows 10](https://beta.support.xbox.com/help/friends-social-activity/share-socialize/record-game-clips-game-bar-windows-10) in the Microsoft Support documentation.
* To create an animated GIF, you can use [Recordit](https://recordit.co/) on either macOS or Windows.

**Submission**

* Upload the files for this assignment to your GitHub repository.
* Submit the link to your GitHub repo on Bootcamp Spot.

© 2021 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.