**Documentation**

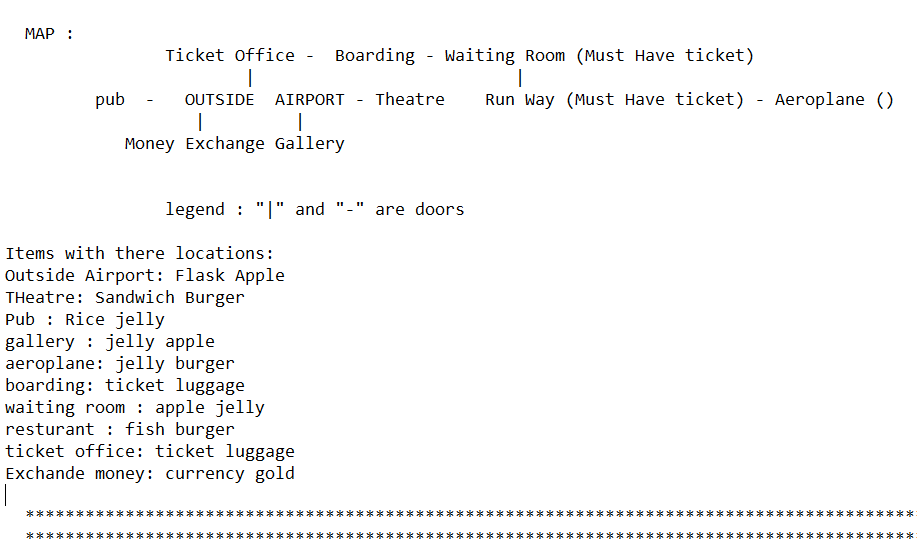
Write a brief description of your game in your word document. You must:

• Describe your game including the back story and the setting

• List the items in the game and how to use it

* 1. • Draw a map for your game scenario. You must: o Label the rooms
  2. o Label the exits (connections between rooms)
  3. o Specify the locations of the items in the map

The document must include the map (e.g., screenshot, picture).



**• Explain how the player wins**

**This project is based on the "World of Zuul" project with an Airport scenario.**

**This game is based on the moving of a player through different steps or walkthroughs. The objective is obviously to find the Runway and get to the Aeroplane in order to win the game.**

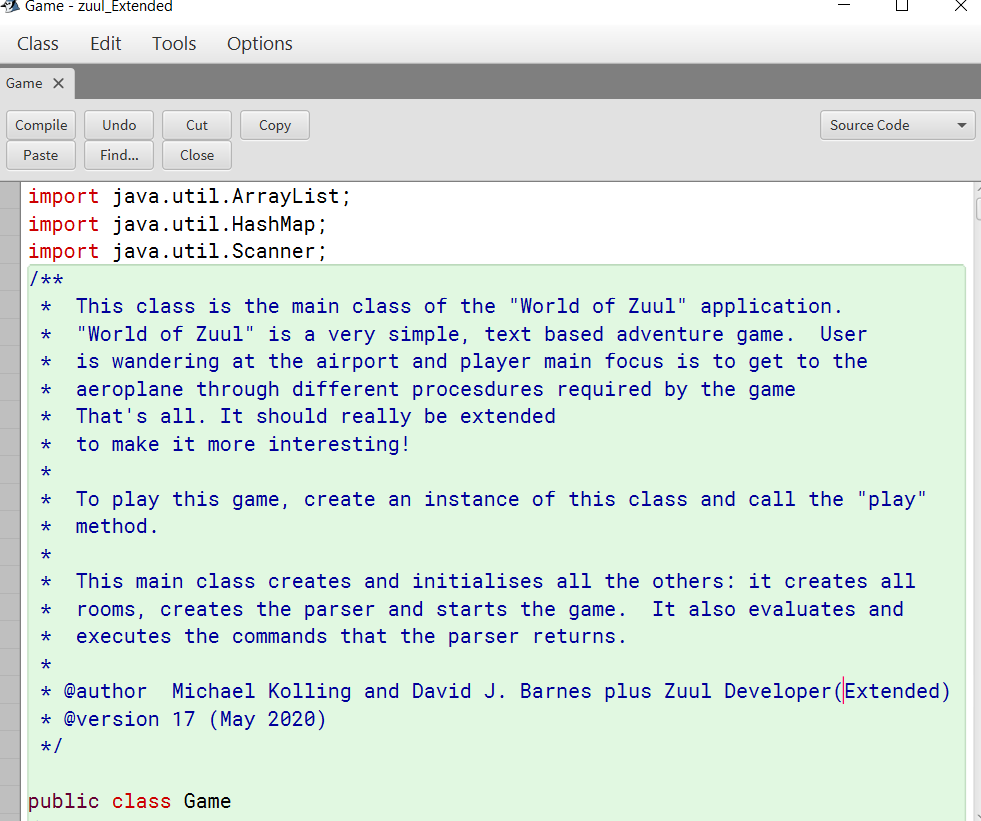
**The Locked Doors: The player has to have the Ticket to pass those two doors which are the most critical part of this game to win in the scenario!**

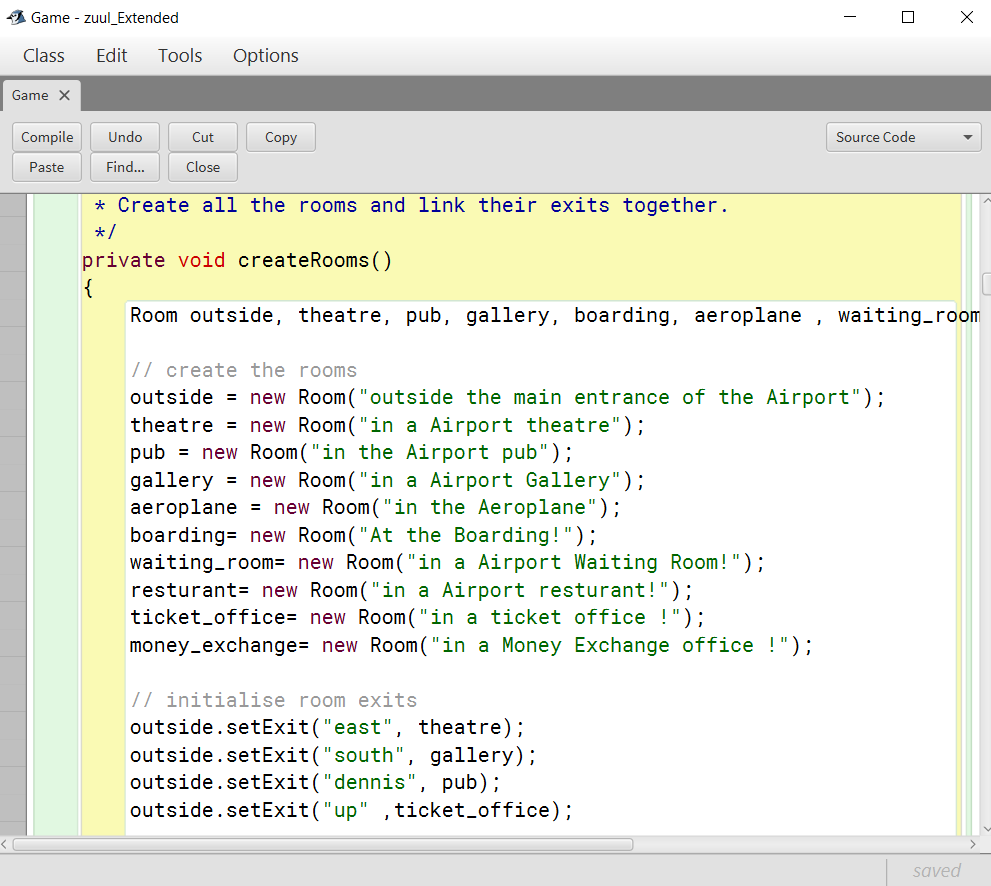
**Part 2:**

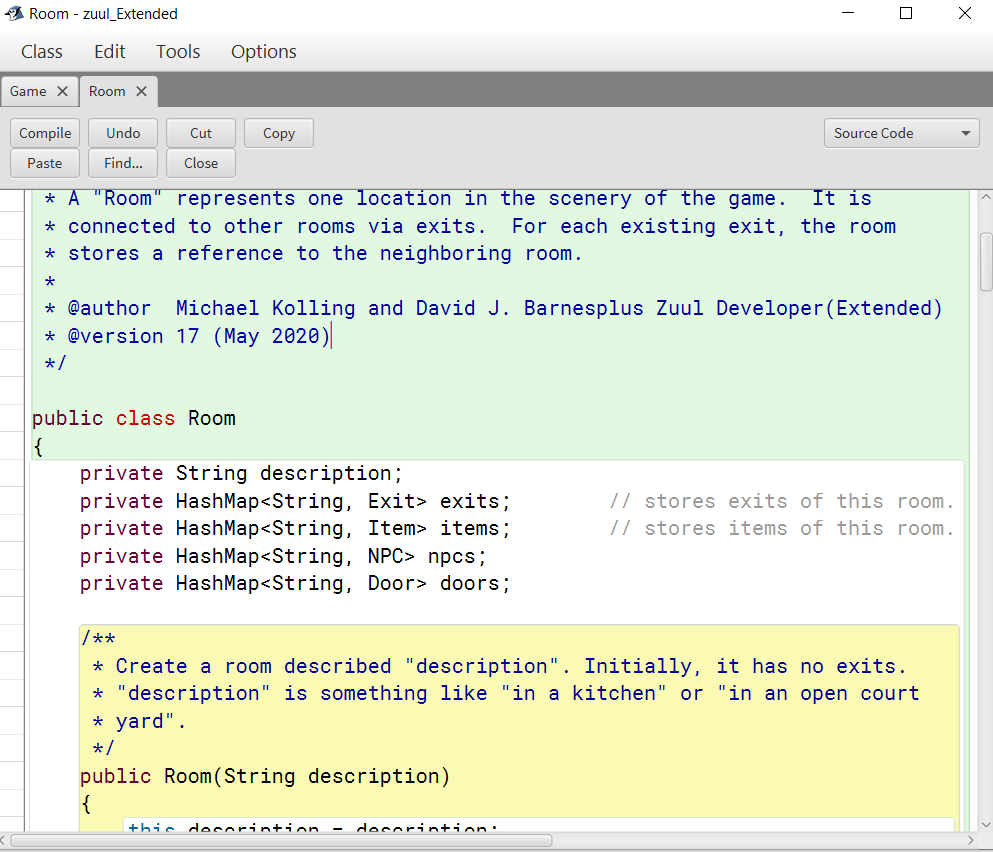
**Programming exercise 1**

• Update the **comments** at the beginning of the **Game** class, the **Room** class and the message displayed by the **printWelcome** and **printHelp** method so that they describe your game.

• Update the **Game** and **Room** class so that it creates the **rooms** and **exits** that you invented for your game. You do not need to add any items to your game yet. You will add items later.

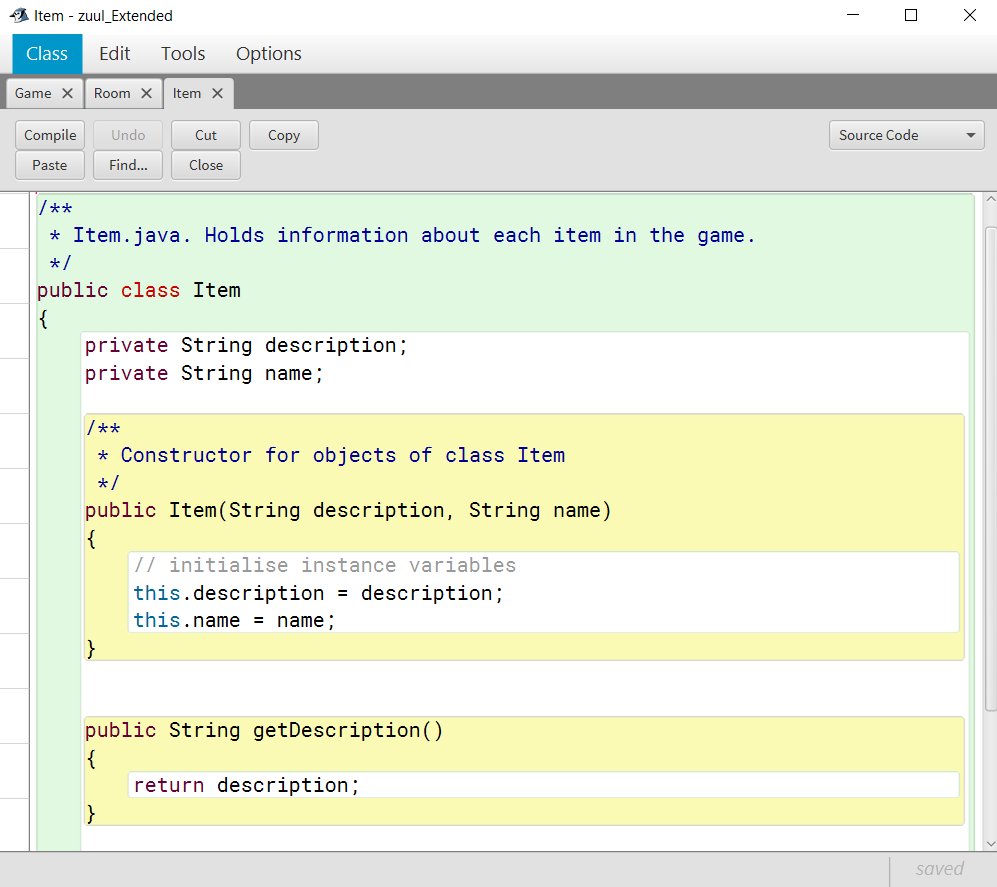






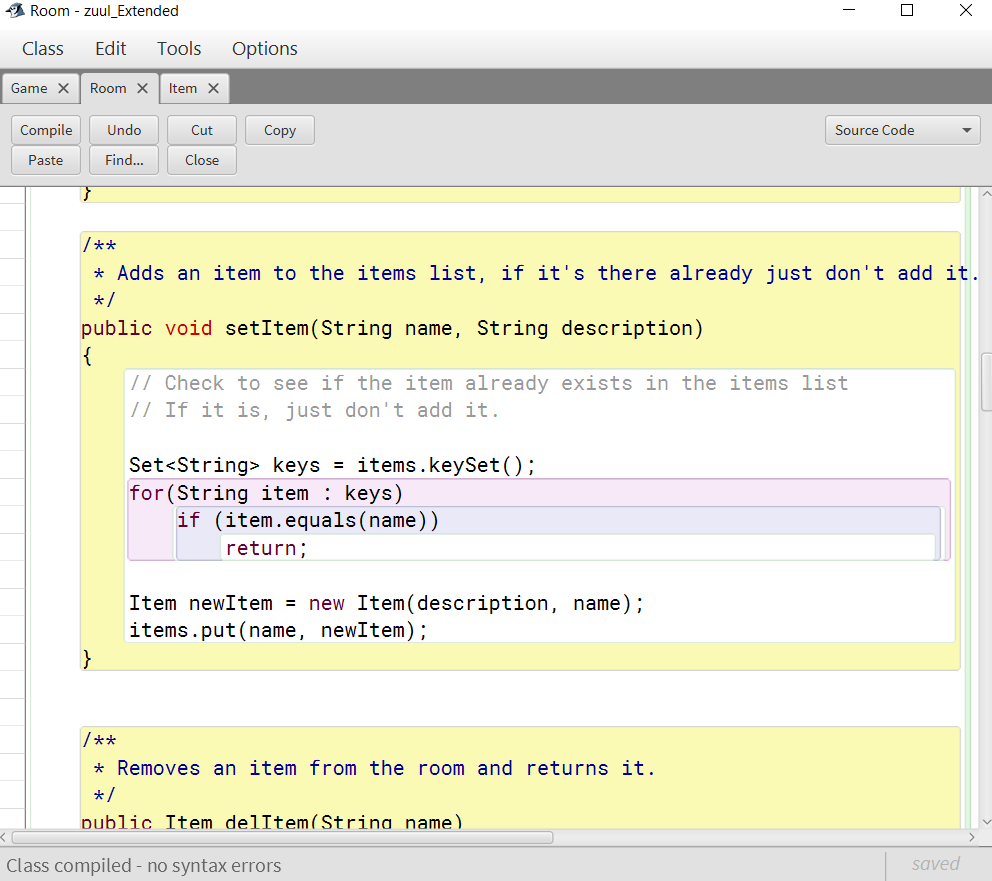
**Programming exercise 2**

Your game scenario requires that there be items positioned throughout the world that the player can pick up and possibly use. An item sounds like something that should be represented by an object! So create an **Item class** to represent the items in your game. You will need to decide what fields your Item class needs to have, what parameters the constructor will require and what methods the class will have. At a minimum, items will have a **name** and a **description**. However, items may have many other attributes that make sense for your game (e.g. weight, colour, value, destructive power ..)



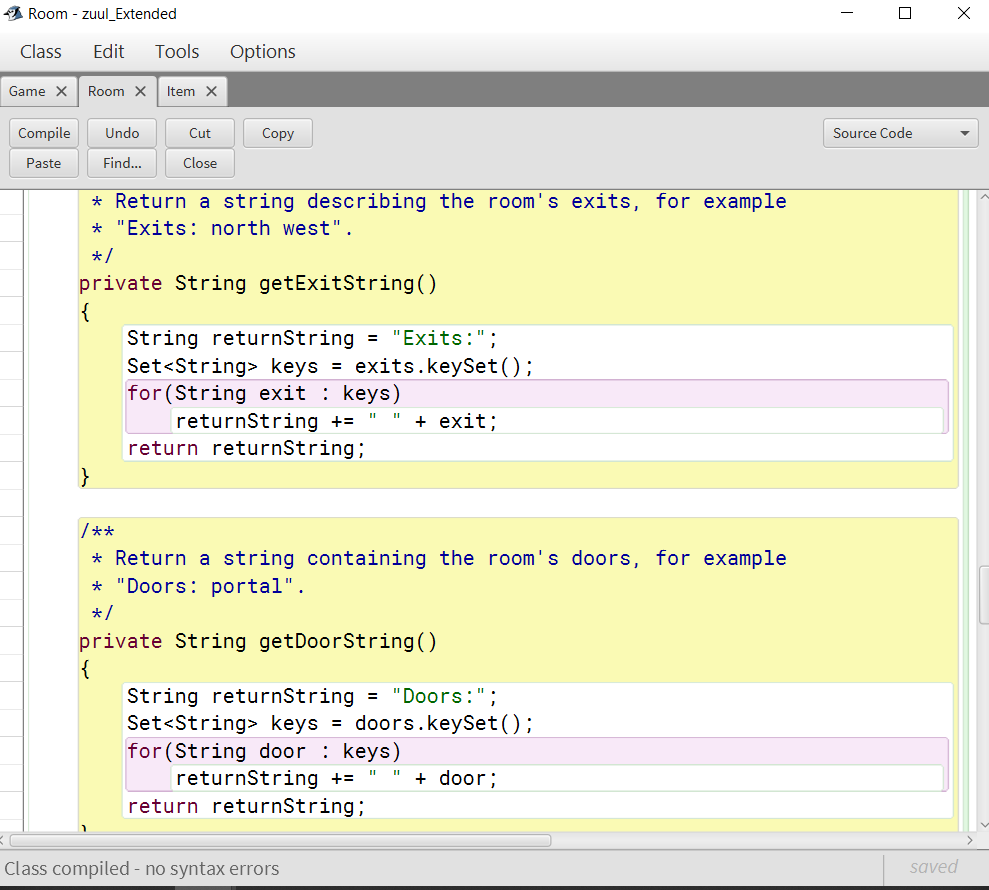
**Programming exercise 3**

Now that there is a class for representing Items we need a way to allow the rooms to contain an item. Modify the Room class so that **items can be added to or removed** from the room. You will need to think about what fields and methods to add to the Room class. Also, think about what the methods that you add should do when an attempt is made to add an item to a room that already contains an item, or an attempt is made to remove an item from a room that does not contain an item.



**Programming exercise 4**

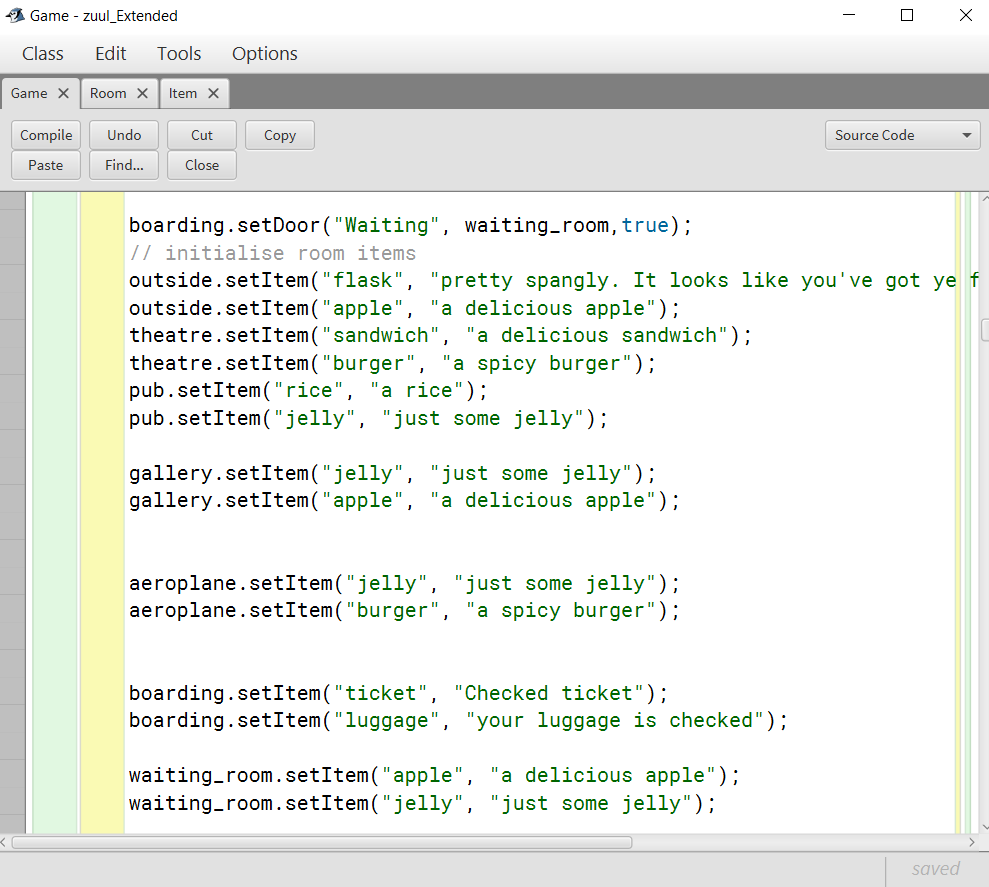
Now that a room can contain an item, when the player enters a room he/she should be **told about the item** in that room (unless you have taken the item). Modify the appropriate code so that if the player enters a room containing an item, the name and description of the item are displayed along with the description of the room and the list of exits.



**Programming exercise 5**

Edit the code in the Game class so that the items for your game are created and added to the appropriate rooms at the start of the game. Recall that your game must include at least two items per room. Be sure to test any methods that you add or modify.

Play the game to ensure that your items are appearing in the rooms.

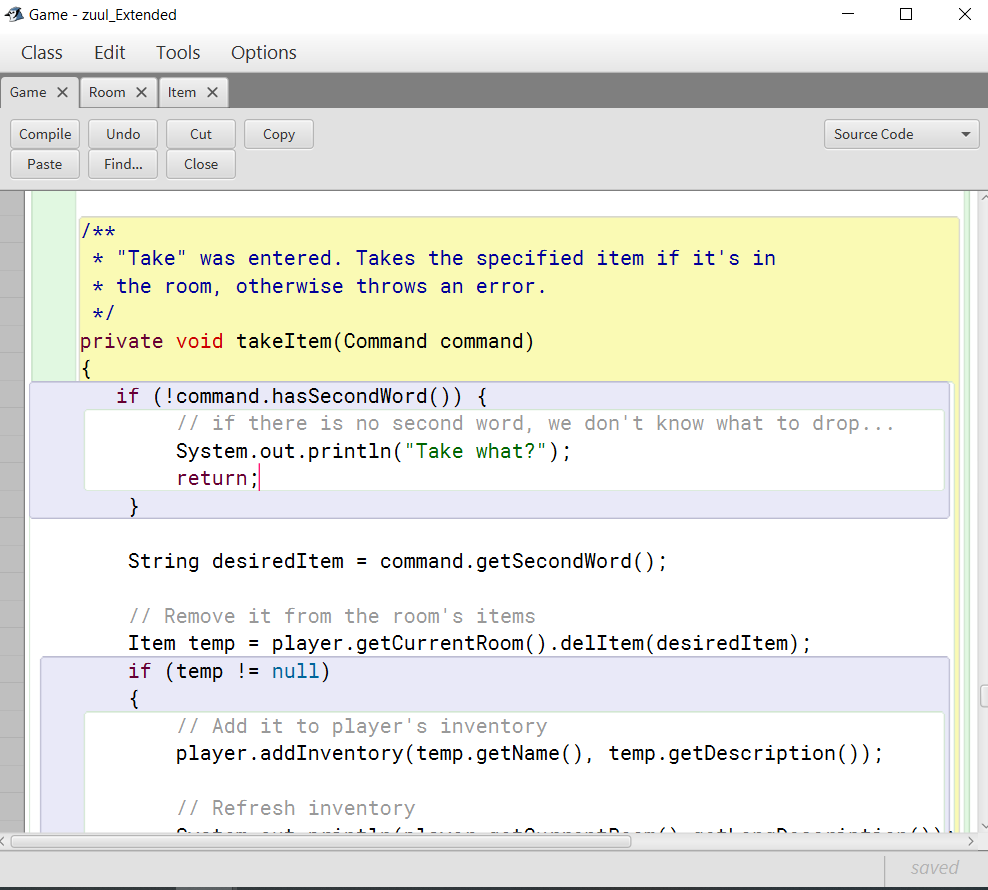


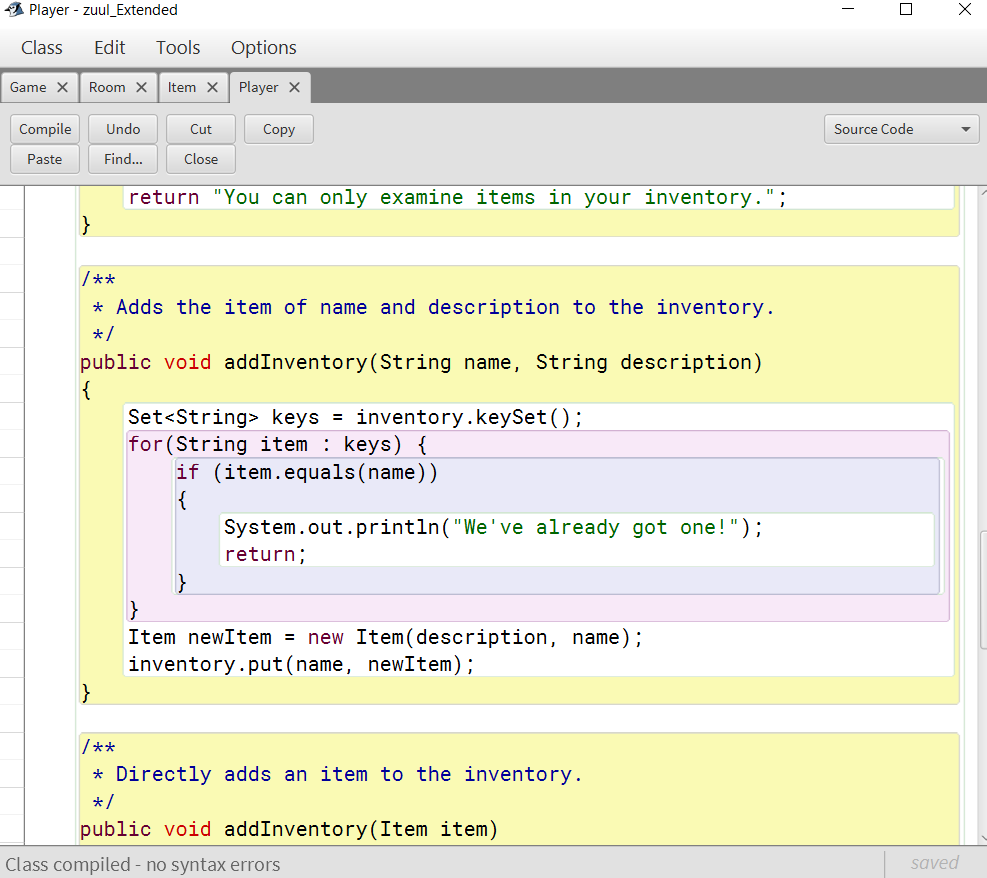
**Part 3:**

Now that rooms can contain items and a player will know when they enter a room with an item, it would be nice if the player could pick up and carry items. Add functionality to the Player class that will allow the player to pick up and drop items. The player should be able to carry any number (i.e. a collection) of items.

**Programming exercise 6**

Modify the Game class so that it will recognize the command **take**. When the user enters the "take" command, the item in the current room, if there is one, should be added to the items that the player is carrying, and a message should be printed indicating that the player has taken the item. If there is no item in the current room the take command should print an error message. Be sure to test any methods that you add or modify. (Hint: Remember that one task of the Game constructor is to "teach" the CommandReader what words are valid commands. Thus, you will need to make a change in Game's constructor if you want to introduce a new command.)

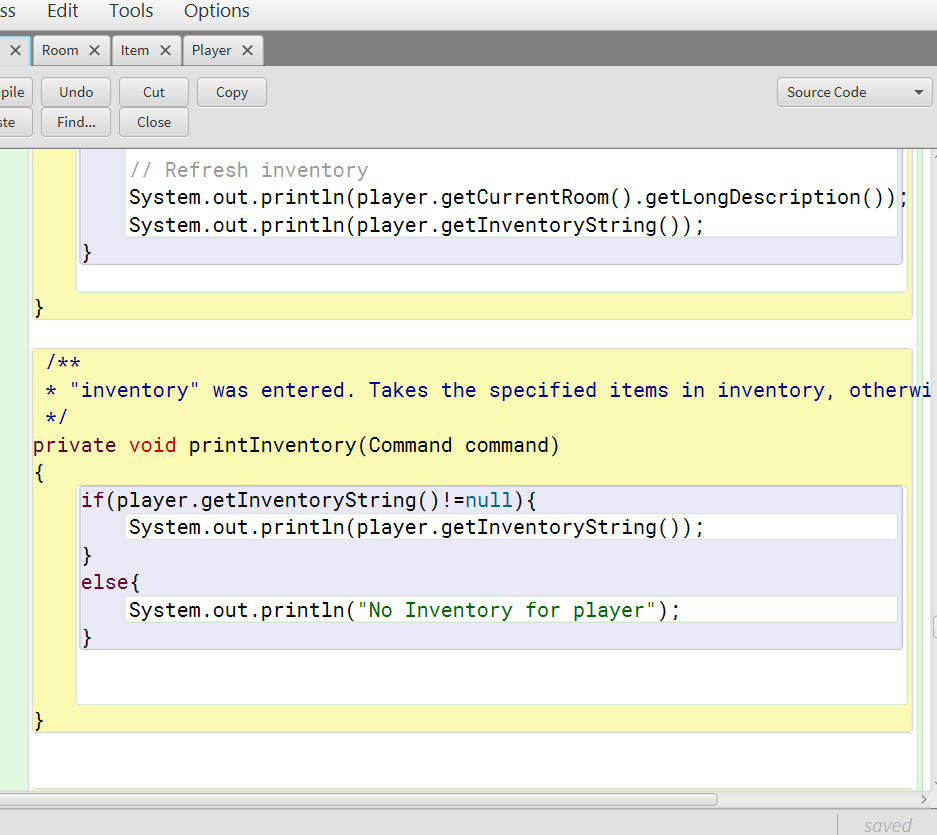




**Programming exercise 7**

Modify the Game class so that it will recognize the command **inventory**. When the user types "inventory" the game prints the names of the items that the player is currently carrying. You should think carefully about where the list of item names should be generated. (Consider the fact that the player is carrying the items and think about how the list of exits for a room is generated and displayed.)

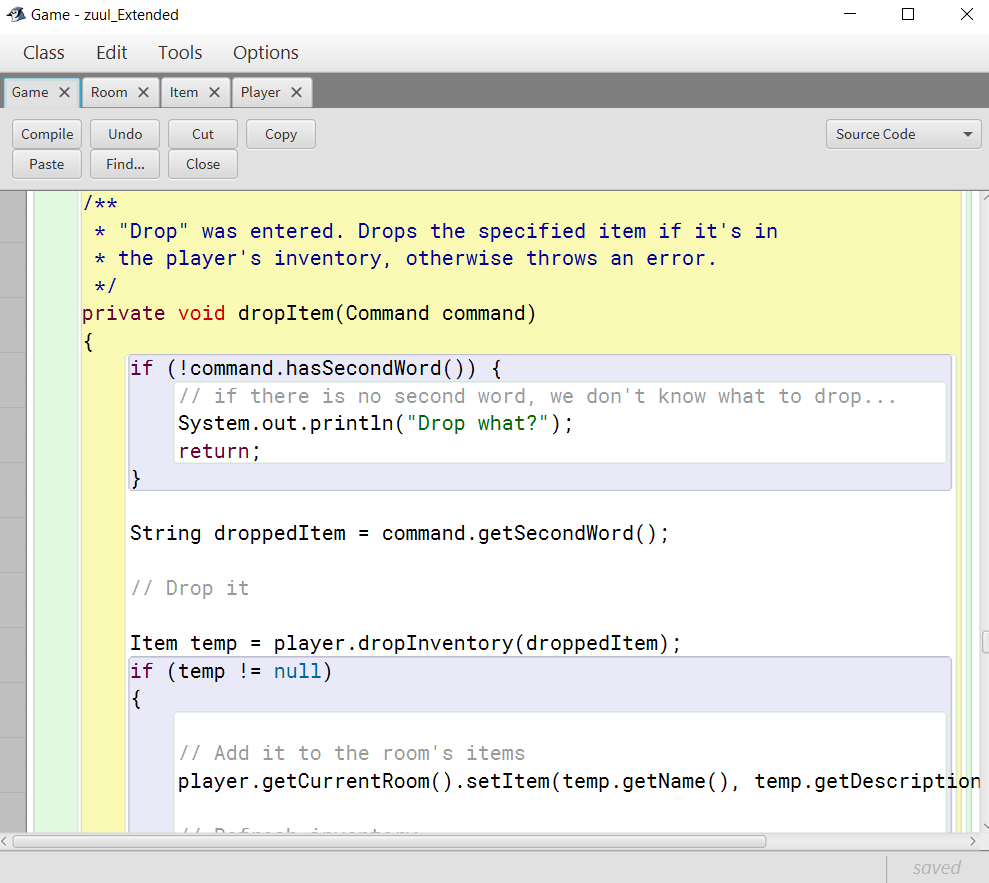
Play the game to be sure the inventory command works!



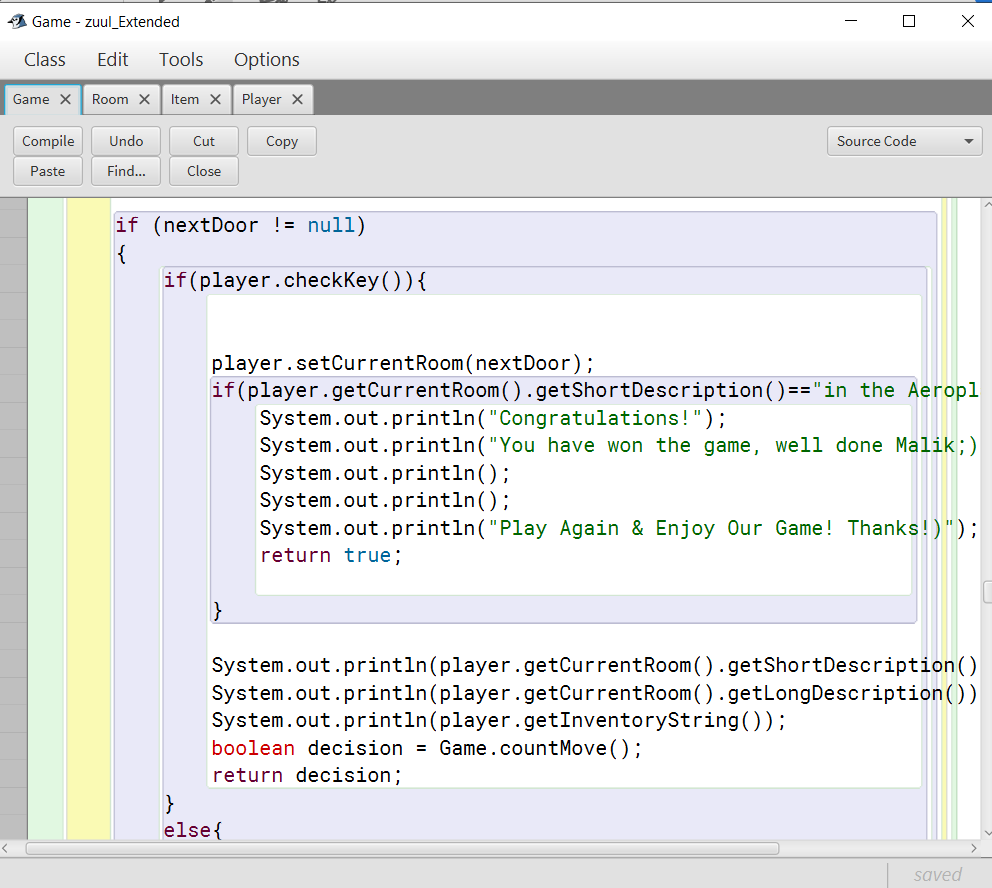
**Programming exercise 8**

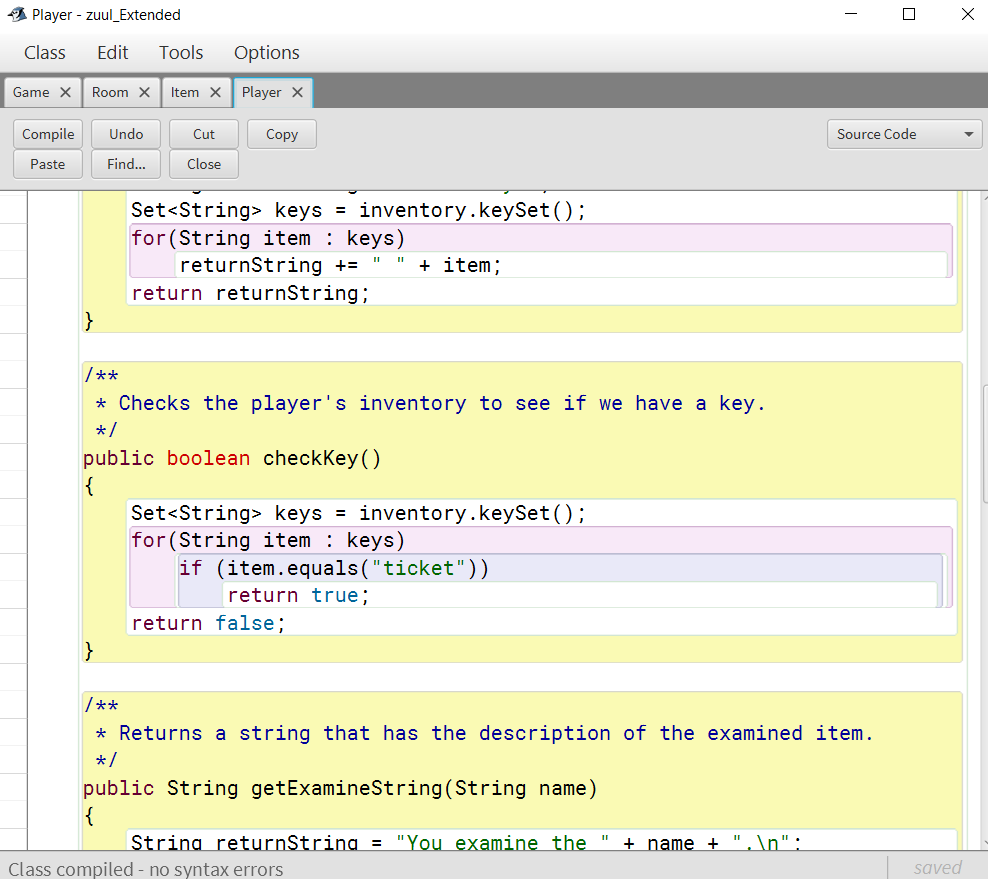
Add support to the game for a **drop** command so that the player can drop an item by name (e.g. "drop book"). The dropped item should appear in the current room. If the current room already contains more than 2 items, the drop command should print an error message indicating that the room is full and the player should continue to carry the item.

Play the game to be sure the drop command works!



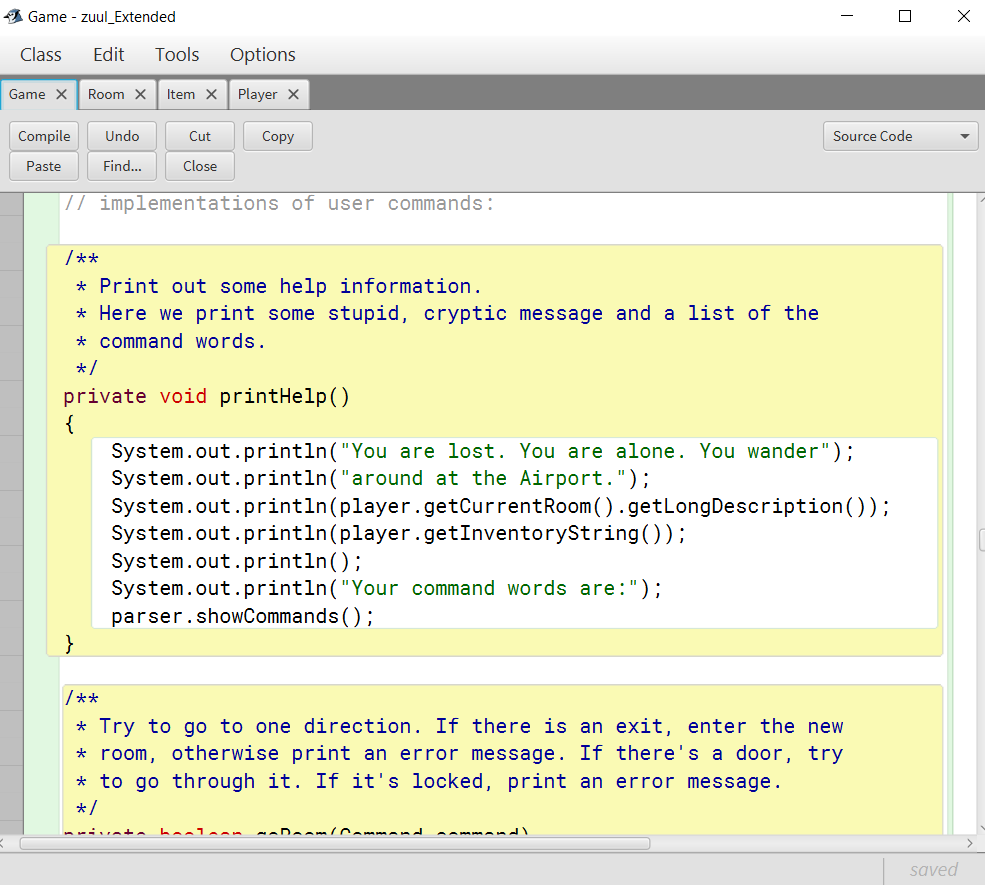
**Programming exercise 9**

Implement **at least two locked doors** to your game. The first door/entrance door should be locked and for the second one, you may choose any other doors. The player needs to find (or otherwise obtain) a key/item to open a door. You will need to create a new **“use”** command to open the door. Hint: Reuse the take command to implement the use command. Adjust your winning condition if necessary. 



**Programming exercise 10**

Modify the **printHelp** method of the Game class so that it also prints the status of the current room and make sure to include helpful messages to the players.



**Programming exercise 11**

Add some form of time limit to your game. If a certain task is not completed in a specified time, the player loses. A time limit can easily be implemented by counting the number of moves or the number of entered commands. It does not have to be real-time.

Play the game to be sure the modified help command works - celebrate!

