P05 Dragon Treasure Adventure 2.0

Overview

In P03, you each implemented instantiable classes to make a simple text-based adventure game. If you recall, you wrote a lot of similar code (e.g. Dragon and Player) in multiple places or had a large set of methods or fields that were used for only one type of Room (e.g. teleportLocation or getPortalWarning()). Using the OOP concepts of inheritance, interfaces, and encapsulation, we will iterate on the previous version of Dragon Treasure Adventure by restructuring the code and incorporating some rudimentary graphics using the same GUI and processing library as in P02! You can view a demo of the finished project here.

Grading Rubric

5 points	**Pre-Assignment Quiz:** The P5 pre-assignment quiz is accessible through Canvas before having access to this specification by **11:59PM on Sunday 10/23/2022**. You CANNOT take the pre-assignment quiz for credit passing its deadline. But you can still access it. The pre-assignment quiz contains hints which can help you in the development of this assignment.
20 points	**Immediate Automated Tests:** Upon submission of your assignment to Gradescope, you will receive feedback from automated grading tests about whether specific parts of your submission conform to this write-up specification. If these tests detect problems in your code, they will attempt to give you some feedback about the kind of defect that they noticed. Note that passing all of these tests does NOT mean your program is otherwise correct. To become more confident of this, you should run additional tests of your own.
15 points	**Additional Automated Tests:** When your manual grading feedback appears on Gradescope, you will also see the feedback from these additional automated grading tests. These tests are similar to the Immediate Automated Tests, but may test different parts of your submission in different ways.
10 points	**Manual Grading Feedback:** After the deadline for an assignment has passed, the course staff will begin manually grading your submission. We will focus on looking at your algorithms, use of programming constructs, and the style and readability of your code. This grading usually takes about a week from the hard deadline, after which you will find feedback on Gradescope.

Learning Objectives

The goals of this assignment include:

- Implementing an interface.
- Practice with coding and utilizing inheritance in a program.
- Practice overriding methods from a parent class or interface.
- More practice with instantiable classes and throwing exceptions.
- Seeing polymorphism being used in action!

Additional Assignment Requirements and Notes

(Please read carefully!)

- Pair programming is **ALLOWED** but not required for this assignment. If you decide to work with a partner on this assignment, REGISTER your partnership NO LATER than **11:59PM on Sunday 10/23/2022** and MAKE SURE that you have read and understood the CS300 Pair Programming Policy.
- **The ONLY external libraries** you may use in your program are these libraries: `java.io.File`, `java.io.IOException`, `java.util.ArrayList`, `java.util.Scanner`, `java.util.Random`, `processing.core.PApplet`, and `processing.core.PImage`.
- Only your submitted `DragonTreasureGame` class can contain a **main** method.
- You are NOT allowed to add any constant or variable not defined in this write-up **outside of any method**.
- You CAN define local variables (inside a method) or **private** helper methods that you may need to implement the methods defined in this program.
- You CAN NOT add any **public methods** to your classes other than those defined in this write-up.
- Feel free to **reuse** any of the provided source code or javadoc comments in this write-up verbatim in your own submission.
- Be sure that your code follows the CS300 Course Style Guide.
- You MUST adhere to the Academic Conduct Expectations and Advice.
- Make sure that all overridden methods contain a @Override above the method signature.

1 Getting Started

- Start by creating a new Java Project in eclipse called **P05 Dragon Treasure Adventure 2.0**, or something similar. As always make sure that you are using **Java 17**, don't add a module, and that you use the default package.
- Then, create a new Java class called `DragonTreasureGame` and add a main() method.
- Download this core.jar file so that you have the processing library needed for the GUI. Add it to your project folder, right-click it, and pick "Build Path > Add to Build Path". (You may need to refresh the Project Explorer in Eclipse if the jar doesn't appear immediately.)
- Finally, download this zip file that holds all the images and add them to a folder called "images" in your project.
- Here you can download a completed implementation of P03 that you can use as a reference to help with implementing this program.

2 Game Overview & Changes

In this game the player will be placed into a room in the cave based on a layout that has been loaded in. The player gives input about which room they'd like to move to and will move to that room (if allowed). The game will print out messages if certain dangers or special rooms are nearby. The dragon also gets to make a move and enter a new room. This behavior will repeat until either 1) the player reaches the treasure room unharmed or 2) the dragon catches up to the player and burns them to a crisp.

NEW to 2.0!

- Instead of using solely text to interact with the player, graphics will be used to show some visuals of the room and the keyboard directly to receive input.
- Portal rooms will now teleport the player to *any random* room that is adjacent to it.
- One more gameplay addition will be a "KEYHOLDER" character who is stationary.
- The player must find and visit them before being able to open the treasure chest and win.

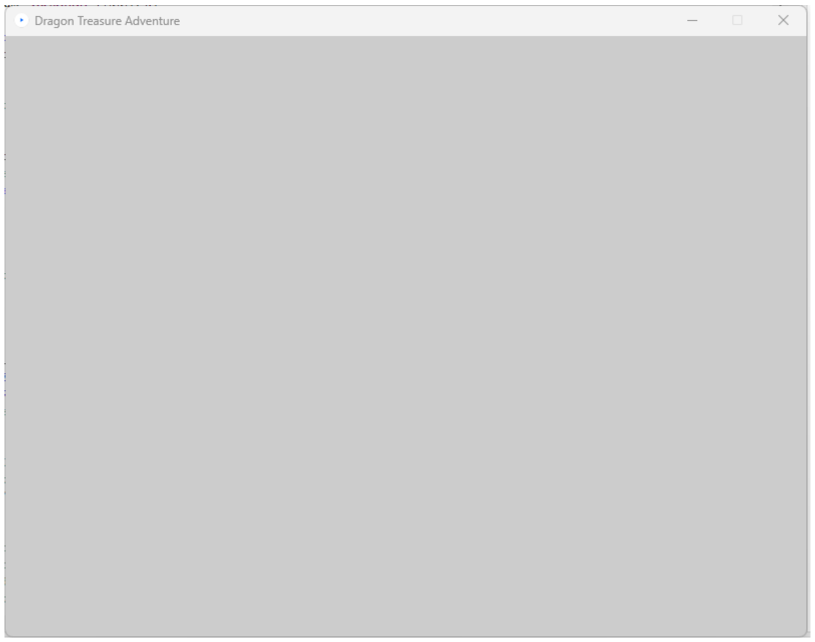
3 Setting Up the GUI

First let's start by getting our game window up and running! In P02, we relied on the Utility class to do a lot of the work for us. This time around we will use the PApplet class by using inheritance.

- Now, make the `DragonTreasureGame` class inherit from `PApplet`. Make sure to **import** `processing.core.PApplet` to make your code compiles.
- Inside the `main()` method, call `PApplet.main("DragonTreasureGame")`.
- If you launch your program successfully, right now a `very small` window should appear. Let's continue to make some adjustment to it.
- Add and **override** the **public void** `settings()` method.
- In the `settings()` method add a call to `size(800,600)`. This will set the window to have a width of 800 and height of 600.
- Next, add and **override** the **public void** `setup()` method. Add the following code to the beginning of the method:

```
this.getSurface().setTitle("Dragon Treasure Adventure"); // sets the title of the window
this.imageMode(PApplet.CORNER); //Images are drawn using the x,y-coordinate
  //as the top-left corner
this.rectMode(PApplet.CORNERS); //When drawing rectangles interprets args
  //as top-left corner and bottom-right corner respectively
this.focused = true; // window will be active upon running program
this.textAlign(CENTER); // sets the text alignment to center
this.textSize(20); //sets the font size for the text
```

- **[CHECKPOINT]** Once you've completed these steps, you should see a blank window as pictured below with no errors upon running the program!



4 Building Rooms

Instead of shoving all different types of room information into one class, we'll restructure it into one parent (base) class `Room` and the remainder into children (derived) classes.

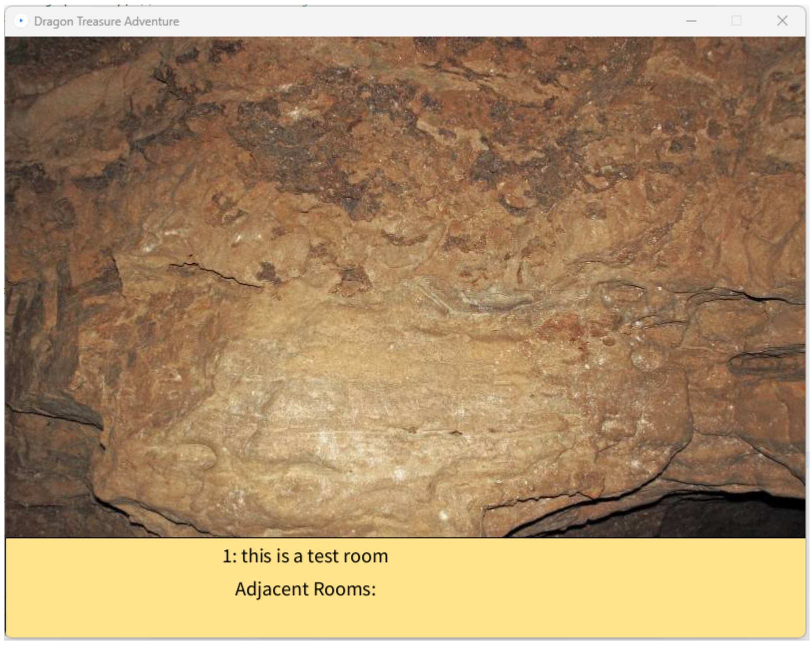
- Start by creating a new Java class in your project named `Room.java`. Then add the following data fields:

```java
private String description; //verbal description of the room
private ArrayList<Room> adjRooms; //list of all rooms directly connect
private final int ID; // a "unique" identifier for each room
protected static PApplet processing; //PApplet object which the rooms will use to
//draw stuff to the GUI
private PImage image; //stores the image that corresponds to the background of a room
```

Now go ahead and implement the methods for the Room class as described in the Javadocs. A lot of them should be familiar/similar to the ones found in P03. We'll also provide a bit of extra information on the two completely brand new methods.

- Take note of the `setProcessing()` method. If this is not implemented correctly, then the program will have issues drawing things to the window.
- The `draw()` method will need to do the following steps:
  1. Use the `PApplet`'s image() instance method to draw the image at $(0, 0)$.
  2. Use the `PApplet`'s fill() instance method to change the draw color to giving it a value of -7028. This will change it to *Flavescent* a light yellow-brown color.
  3. Use the `PApplet`'s rect() instance method to draw a rectangle. The first two arguments are the xy-coordinates of the top left corner respectively. The third and fourth arguments are the xy-coordinates of the bottom right corner respectively. Place the upper left corner at $(0, 500)$ and the other at $(800, 600)$.
  4. Use the `PApplet`'s fill() instance method again to change the draw color to giving it a value of 0. This will change it to black.
  5. Use the `PApplet`'s text() instance method to draw the `Room`'s `toString()` at $(300, 525)$.
- Once you're done implementing them, return back to your `DragonTreasureGame` class and add this data field: **private ArrayList<Room>** `roomList`. In the `setup()` method initialize it to an empty ArrayList.
- In `setup()` call `Room.setProcessing()`. Remember that `setProcessing()` takes a `PApplet` as its argument, and that `DragonTreasureGame` *is a* `PApplet`.

- [CHECKPOINT] In `setup()` use `PApplet`'s loadImage() instance method to load an image from the "images" folder. Use that `PImage` to make a new `Room` object and add it to the ArrayList.
- Add a new method **public void** `draw()` in `DragonTreasureGame`. In this method call the `Room`'s `draw()` method. If you run your program you should see the image, rectangle, and text of the room in the window. If you use 1.jpg as the image, that will look like:



1: this is a test room

Adjacent Rooms:

4.1 StartRoom Child Class

We'll start out with one of the simpler children classes first.

- In your project create a new class called `StartRoom`. This class should inherit from the `Room` class you wrote earlier.
- Add to this class a constructor with the following signature:

```
public StartRoom(int ID, PImage image){};
```

- Now go ahead and implement the constructor such that a `StartRoom` by calling the `super()` constructor will have the following description reading.

```
"You find yourself in the entrance to a cave holding treasure."
```

- **[CHECKPOINT]** Return to the `DragonTreasureGame`, create a `StartRoom` object and add it to the `rooms` ArrayList. Notice how you can add something that is not EXACTLY type Room, even though the data type the is listed as `Room`? This is polymorphism in action! `StartRoom` takes on the form of `Room` so it can be added to the ArrayList because of the parent-child relationship between the two!
- Now have the `DragonTreasureGame`'s `draw()` method draw only the `StartRoom` object and you should notice that it still works but the description is the one you used in the `StartRoom` constructor!

4.2 TreasureRoom Child Class

- In your project, create a new class called TreasureRoom. This class should inherit from the `Room` class you wrote earlier.
- ALL `TreasureRoom` objects (there could be more than one) will use the same picture for the background. Now add the following data fields:

```
private static final String TREASURE_WARNING = "You sense that there is treasure nearby.\n";
private static PImage treasureBackground; //the image ALWAYS used for treasure rooms
```

- Implement the methods described in the Javadocs for the class TreasureRoom **except** `playerCanGrabTreasure()`.
- Return to the `DragonTreasureGame` `setup()` method, load the "treasure.jpg" image and set the background for the `TreasureRoom` class.
- **[CHECKPOINT]** Just like the previous rooms, make an instance of a `TreasureRoom`, draw it, and check that it works as expected.

4.3 PortalRoom Child Class

Now onto the last child class for `Room`. In your project create a new class called PortalRoom. This class should inherit from the `Room` class you wrote earlier. These rooms will draw an additional image to the screen and randomly pick an adjacent room to be the teleport destination (as opposed to one exact room on the map). Add the following data fields to the class:

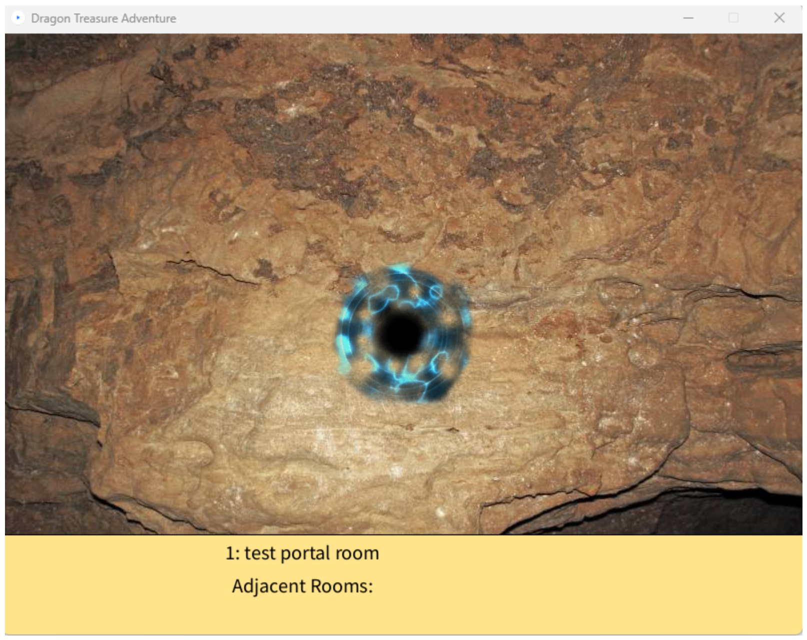
```
private Random randGen; //random number generator for location picking
private static final String PORTAL_WARNING = "You feel a distortion in space nearby.\n";
private static final String TELEPORT_MESSAGE = "The space distortion teleported
    you to another room!\n";
private static PImage portalImage; //image of a portal to be shown in all portal rooms
```

Implement the methods described in the Javadocs for the class PortalRoom. Here are some additional details for some of the methods:

- The `getTeleportLocation()` method should randomly pick one of the adjacent rooms and return that room. A `PortalRoom` having no adjacent rooms is a situation that will not occur, you can assume that all rooms will have non-empty adjRoom lists.
- The `draw()` method will `partially override` the one from the parent class. It should do the same thing steps as the parent but in addition have it draw the `portalImage` on top of everything else at $(325, 225)$.

Load and set the `portalImage` in the `setup()` method for the image "portal.png".

[CHECKPOINT] Just like the previous rooms, make an instance of a `PortalRoom`, draw it, and check that it works as expected. An example of a successfully drawn portal room might look like this:



1: test portal room

Adjacent Rooms:

4.4 Room Loading

- Clear out the other room objects you might have put in the list earlier.
- Then, go ahead and add these methods to your `DragonTreasureGame` and these text files (map.txt and roominfo.txt) with map and room information to your project.
- NOTE: In your `DragonTreasureGame` class, you will need to add two **private instance** data fields of type `File` named `roomInfo` and `mapInfo`. Make sure to `import java.io.File` to your class if not yet done.

- In `setup()`, initialize those file objects to their respective text files. Then, call `loadRoomInfo()` and `loadMap()`.
- Your program should be able to successfully load all the cave information and rooms to your `ArrayList` so it should be possible to draw any of those rooms to the screen. It might take a minute to load all the images in when you launch the program but we've included a print statements to help you know which of the two loading processes it is doing!

5 Creating Characters

If you remember back to P03 both the `Dragon` and `Player` had a lot of shared and redundant code, this is where inheritance comes in handy!

- We'll start with a base class named Character. Go ahead and create a new Java class of that name. Using this we can model different basic characters and then create more detailed ones! Now add the following data fields:

```
private Room currentRoom; //current room the character is in
private String label; //a label giving a basic description of the character
```

- Now implement the remaining methods as described in the Javadocs of the Character class!
- We want some things in our game to move between rooms (e.g. like the player).
- We provide a Moveable interface to have a standardized way of moving objects around the game. Go ahead download it and add it to your project (but don't add it to the Character class – not all Characters are moveable!).

5.1 Player Class

Time to make a new Java class with a name of Player and add it to the project.

- The `Player` class should inherit from `Character` and implement the `Moveable` interface.
- Add the only extra **instance** data field that is **private boolean** `hasKey` to your `Player` class.
- Now go and write the methods described in the Javadocs of the Player class **except** `isDragonNearby()`. Note that most of these methods should work analogously to P03.
- Also this is now the time to go back and write the `playerCanGrabTreasure()` method in the `TreasureRoom` class.

5.2 Dragon Class

Time to make yet another new Java class this time with a name of `Dragon` and add it to the project.

- The Dragon class should inherit from `Character` and implement the `Moveable` interface.
- The dragon in 2.0 has the same movement pattern as it did in the previous version!
- Add the following data fields to the class:

```
private Random randGen; //random num generator used for moving
private static final String DRAGON_WARNING = "You hear a fire breathing nearby!\n";
private static final String DRAGON_ENCOUNTER = "Oh no! You ran into the fire
    breathing dragon!\n";
```

- Now go and write the methods described in the Javadocs of the Dragon class.
- Then, finish implementing the `isDragonNearby()` method that remains in the Player class.

5.3 Final Character Stuff

- Now, add the following `loadCharacters` helper method to your `DragonTreasureGame` class. This method will load in the three required characters into the game.
- NOTE: You will also need to add a **private ArrayList<Character>** `characters` data field to your `DragonTreasureGame` class, initialize it to an empty ArrayList in `setup()`, and then call the `loadCharacters()` method from `setup()`.

```
private void loadCharacters() {
  System.out.println("Adding characters...");
  characters.add(new Character(getRoomByID(5),"KEYHOLDER"));
  characters.add(new Player(getRoomByID(1)));
  characters.add(new Dragon(getRoomByID(9)));
}
```

- If you want to check you can have the `draw()` method draw the player's current room and it should be the starting room. Similar thing for the KEYHOLDER and dragon.

6 Finishing Game Logic

All that's left to now code is a bit more of the game logic. Since `draw()` is a callback method we can utilize that fact to have it be our gameplay loop! Do the following steps to complete the game logic:

1. Add a **private instance boolean** `isDragonTurn` data field. At the start of the game this should be false so the player gets to move first.
2. Add a **private instance int** `gameState` data field. At the start of the game this should be 0. If the value is 0 the game should continue, if the value is 1 that means the player won, and if the value is 2 the player lost.
3. In the `draw()` method do the following steps:
   (a) Draw the `currentRoom` of the Player. (Make this generic for any ordering in the ArrayList.)
   (b) Check for any warnings (e.g. portals, treasure, dragon) that need to be given to the player. Print those messages to the console. (We'll avoid thinking about putting them to the screen and making them look nice... Trust us a lot of extra work.)
   (c) Check if the player can grab the key, if they can let them get it. They can obtain the key if they are in the same room as the "KEYHOLDER" character. If they obtain the key, print "KEY OBTAINED" to the console.
   (d) Check if the player needs to teleport because they are in a room with a portal. If they do teleport successfully, print the message to the console.
   (e) If it is the dragon's turn to move and the game should continue have it `changeRoom()`. If the change is successful, make it that is no longer the dragon's turn.
   (f) Check and update the `gameState` data field. If the player is in the treasure room and can open the treasure they win. If the dragon and player are in the same room then the player loses. Additionally, print a message to the console if they lost or won.
4. In your `DragonTreasureGame` class, **override** the `PApplet`'s **public void** `keyPressed()` method. Here is how we will let the user move the player.
   (a) Use the key pressed value (provided automatically by PApplet as a variable called `key`) as the ID of the room the player wants to move into and `changeRoom()`. For example, if the user wants to move to room ID 2, they would press key `2`. You **do not** need to consider cases where the ID is more than one digit.
   (b) If the change is successful, make it the dragon's turn to move.
   (c) If it is not, then print out to the console for the user to pick a valid room.
   (d) The player should not be able to move if the `gameState` is either lost or won.

From here you should have a functioning version of the game with some very simple graphics!

7 Assignment Submission

Congratulations on finishing this CS300 assignment! After verifying that your work is correct, and written clearly in a style that is consistent with the CS300 Course Style Guide, you should submit your final work through Gradescope. The only EIGHT files that you must submit include: `Room.java`, `PortalRoom.java`, `TreasureRoom.java`, `StartRoom.java`, `Character.java`, `Player.java`, `Dragon.java`, and `DragonTreasureGame.java`. Your score for this assignment will be based on your **"active"** submission made prior to the hard deadline. The second portion of your grade for this assignment will be determined by running that same submission against additional offline automated grading tests after the submission deadline. Finally, the third portion of your grade for your submission will be determined by humans looking for organization, clarity, commenting, and adherence to the CS300 Course Style Guide.