# Text Adventure

Your name:

By the time you are done with this activity, you should be able to:

- use objects to build more concisely write complex programs.
- define simple classes of objects.

After you complete this activity, please fill out the short survey at

http://goo.gl/forms/HXjyuUb2ou

to improve this project for future users.

## Playing the game

Included in the Text Adventure project is a file TextAdventure.jar. This is a compiled version of the working game. To play it, use a terminal to navigate to the directory containing the file and type this on the command line:

```
java -jar TextAdventure.jar
```

Note that this game runs in the console, rather than in a graphic window.

Text adventure games go back to 1970s, beginning with *Colossal Cave Adventure*, which was in turn inspired in part by the tabletop roleplaying game *Dungeons & Dragons*. The player controls an adventurer who explores a network of caves, fighting monsters and collecting treasures. Commands are given by typing simple English text (e.g., “go north”). Part of the challenge is figuring out what works.

This is a one-player game. You should play until you’re able to find all of the treasures in this very small maze. Ask your classmates for help if you gets stuck.

## Overview

You have only been given some of the files needed to play the game. Your first job is to complete the program so that it behaves *exactly* like TextAdventure.jar. After that, you will have considerable creative freedom to design your own adventure.

Take notes as you work. What bugs and conceptual difficulties did you encounter? How did you overcome them? What did you learn?

For each of the classes you need to create there is a corresponding test class. These test classes won’t compile yet because they depend on classes you need to write! For each class below:

1. Write a bare skeleton of the class so that the test will compile. Don’t bother providing any instance variables or putting any content in the methods (other than a return statement that might be necessary for the class to compile). You will have to examine the tests to see what methods are needed.

2. Fail the tests.

3. Work through the tests one at a time *in the order in which they appear in in the test class*, writing the necessary code in your class to pass them. Pass one test before moving on to the next one.

You should not modify any of the given tests. If it is helpful for debugging, you may write additional tests.

## Monster

Use Treasure as a template when writing this class.

Note, in the constructor for Treasure, lines like:

```
this.name = name;
```

An instance variable `name` can always be referred to as `this.name`. Usually the `this.` can be omitted, but in this case it is necessary to distinguish between the instance variable `name` and the argument `name`.

Don't forget to include Javadoc comments for all of your classes, instance variables, and methods. Commenting becomes increasingly important as programs get larger.

## Weapon

This one is very similar to Treasure and Monster.

## Room

This object is somewhat more complex. As a hint, here is a list of instance variable declarations you might use:

```
    private String description;

    private ArrayList<String> exits;

    private Monster monster;

    private String name;

    private ArrayList<Room> neighbors;

    private Treasure treasure;

    private Weapon weapon;
```

Notice that not all of these are specified as arguments to the constructor. What are reasonable initial values for the others?

It may be helpful to have the API for the ArrayList class open in a web browser.

## Design your own adventure.

If your classes are passing all of the tests, you should now have a working adventure (but run it to be sure).

Design your own adventure involving up to 12 rooms. You are free to change the setting to science fiction, western, etc. You *may* add new features (such as an `inventory` command that lists everything the player is carrying), but this can be tricky.

Please fill out the survey at http://goo.gl/forms/HXjyuUb2ou.

This work was supported by the Google Education and Relations Fund's CS Engagement Small Grant Program grant #TFR15-00411.

The principal investigator, Peter Drake, wishes to thank the following for their useful comments: the members of the CS-POGIL project, specifically Clif Kussmaul and Helen Hu; Maggie Dreyer; and various anonymous reviewers.