Assignment 2

Generic Rubric

- 50% task completion: all requirements implemented
- 20% code correctness: code is correct, demonstrated with user tests (screenshots of text UI)
- 20% style: code is well-organized, follows coding conventions, and is easy to read
- 10% presentation: code is adequately documented, UML class diagrams are formatted well

Questions (20 points)

Answer the following yes/no question with a brief justification, such as a working code snippet or a code snippet demonstrating a compile error.

1. An abstract class can have only abstract methods and no method with implementation.
2. When a class is marked as `final`, it can't be extended, but its methods can still be overridden.
3. The `this` keyword can be used to reference the current object of a class, but the `super` keyword can only be used in a constructor to call the superclass constructor.
4. An interface can have a constructor.
5. Polymorphism instructs a subclass to override all the methods of its superclass.
6. Inheritance is transitive.
7. The `static` keyword is used to dfine a variable or method that is unique to each instance of a class.
8. A class that is marked as `abstract` must have at least one abstract method.
9. A class can be marked as `final` and `abstract` at the same time.
10. An interface can extend another interface but not an abstract class.

Text-Based Dungeon Role Playing Game (60 points)

Implement a role playing game in which a player must find a path through a dungeon from a starting chamber to a goal chamber. The description of this task is intentionally vague to give you freedom in exploring object-oriented design and implementation. The player can choose a `Character` (e.g., a wizard or warrior); every character has experience in terms of strength and craft, as well as health points (the player loses when the character health becomes 0). Characters can carry `Item`s (e.g., a sword, a shield, or a magic wand) in an inventory, and designate up to two items as being in use (one per hand). A `Dungeon` consists of `Chamber`s; each chamber has one or several `Door`s that connect to other chambers (and connect back to the current chamber, i.e., the dungeon forms an undirected graph). A door may be guarded by a `Monster` (e.g., a `Goblin` or a `Spider`), which are adversarial characters. As characters, they have health and either strength or craft (but unlike the player's character not both, e.g., if craft is non-zero than strength is 0). When the player enters a chamber, a list of possible `Action`s is presented (e.g., `Fight` a monster, `Pick` an item, or `Move` through a door to an adjacent chamber, which is only possible if the guarding monster of the door is defeated in a fight). A fight is directed by the monster's skill: if the monster has strength (i.e., craft is 0), the fight uses the strength values of the player and the monster. Otherwise, if the monster is skilled in craft (strength 0), the fight uses the craft values. In a fight, the player and the monster each roll a die (random `int` between 1 and 6) and add it to the strength/craft value, plus the strength or craft value of any item that the player carries in hand. The difference gets subtracted from the

health of the monster or player. When the health of the monster drops to 0, the monster is defeated (gets removed from the door).

Your assignment is to implement and test the game.

1. Task 1: Use object-oriented programming principles to encapsulate data in classes, inheritance to factor commonalities into base classes, and polymorphism to choose the concrete behavior of methods depending on the dynamic type of objects at runtime (e.g., how items are printed).
2. Task 2: Demonstrate your game with user tests, taken as screenshots from the text UI outputs as game play evolves.

The code snippet below serves as a partial specification of the API of your classes and it provides the text-based user interface. Your implementation must work with this code.

Game Setup

```
public class Game {

  public static void main(String[] args) {
    // initialize some chambers
    Chamber[] chambers = new Chamber[] {
      new Chamber(),
      new Chamber(new Axe()),
      new Chamber(new Shield()),
      new Chamber(),
      new Chamber()
    };

    // connect the chambers with doors
    Door.connect(chambers[0], chambers[1]);
    Door.connect(chambers[1], chambers[2], new Monster("Goblin", 1, 0,
3));
    Door.connect(chambers[2], chambers[3], new Monster("Spider", 3, 0,
5));
    Door.connect(chambers[3], chambers[4]);

    Character player = new Wizard("Gandalf");

    // initialize the dungeon with player, entry chamber, and goal chamber
    Dungeon d = new Dungeon(player, chambers[0], chambers[4]);

    TextUI ui = new TextUI();
    ui.play(d);
  }

}
```

Text-Based User Interface

The code snippet below defines the text-based user interface. It prints the current room, then prints all the available actions, and asks the user to select one (by inputting a number). Then the UI executes the action and loops.

1. Task: **Address the TODO in `print` and print the monsters that are guarding doors**

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.*;

public class TextUI {
  public void play(Dungeon d) {
    while (!d.isFinished()) {
      print(d);
      Action a = ask(d);
      a.execute();
    }
  }

  /** Print the current room of the dungeon. */
  private void print(Dungeon d) {
    Room r = d.getCurrentRoom();
    StringBuilder s = new StringBuilder();
    s.append("You are in a room with " + r.getDoors().size() + "
doors\n");
    s.append("There are " + r.getItems().size() + " items in the room\n");
    // TODO: print for each door which monster is there, how strong it is,
how skilled in craft, and how healty
    System.out.println(s.toString());
  }

  /** Asks the user for an action. */
  private Action ask(Dungeon d) {
    StringBuilder s = new StringBuilder();
    s.append("Here are your options:\n");
    List<Action> actions = d.getActions();
    for (int i = 0; i < actions.size(); i++) {
      Action a = actions.get(i);
      s.append("\t" + i + ": " + a.toString() + "\n");
    }
    System.out.println(s.toString());

    // ask for action
    BufferedReader reader = new BufferedReader(
      new InputStreamReader(System.in));
    try {
      int command = Integer.parseInt(reader.readLine());
      return actions.get(command);
    } catch (IOException e) {
      return new PrintError(d, e);
    }
```

```
    }
  }
```

Documentation (20 points)

Document your software design with a UML class diagram.