TEAM: John Aldrete, Catherine Xiao

1. Design by contract means an agreement between classes/objects and how they will be used to make sure that state of objects is always valid
   - Implicit contracts are ones put in place by the public methods and class design in the program. There is no rule for compliance to these patterns.
   - Explicit contracts force compliance on their victims, where the specification for a particular routine can be hard coded.
2. In traditional object oriented design, interface methods are only declared and not implemented. This changes in modern Java though, since we are now allowed to have default methods associated with an interface; these tell the implementing class that no implementation is required for the method. Also, Java employs the use of static methods to allow implementing classes direct access to the method from the interface. Furthermore, static methods in an interface cannot be overwritten by the implementing class. Finally, Java uses functional interfaces, which are interfaces with a single abstract method. These are useful because they allow for lambda expressions, which are functional interfaces defined in-line, and don't require the whole class to be implementing the interface to use its functionality.

```java
public interface skeleton{
    //default methods example:
    //to be implemented
    void doSomething();
    default void doesSomething(){
    //need not be implemented in the implementing class.
        System.out.println("does something already!");
    }
}
```

```java
public interface Gandalf{
    //static methods example:
    //unchanging implementation
    static void emote(){
        System.out.println("You shall not pass!!");
    }
}

public class Frodo implements Gandalf{
    public static void scene(){
        Gandalf.emote() //calling static interface method
        System.out.println("Gandalf NO!!!!");
    }
    //@override <- this would result in a compiler error
    //emote()   <-
}
```

```
//functional interfaces example:
@FunctionalInterface
interface circle{
    int doSomething(int r);
}
class Area{
    public static void main(String[] args){
        int r = 2;
        //declaring lambda expression using functional interface
        circle c = (int r) -> pi * (r**2);
        c.doSomething(2);
    }
}
```

3. Abstraction is hiding unwanted information whereas encapsulation is the process of containing the information. So abstraction makes things easier to understand for the user while encapsulation protects important information.

```
Source: https://linuxhint.com/abstraction-java/
public abstract class PersonExample {

    int age;
    int name;

    Person(int age, String name) { //constructor
        this.age = age;
        this.name = name;
    }

    public void display() { // method
        System.out.println("person name:" + name + " " + "person age:" +
age);
    }

    public abstract void add(); // abstract method
}
```

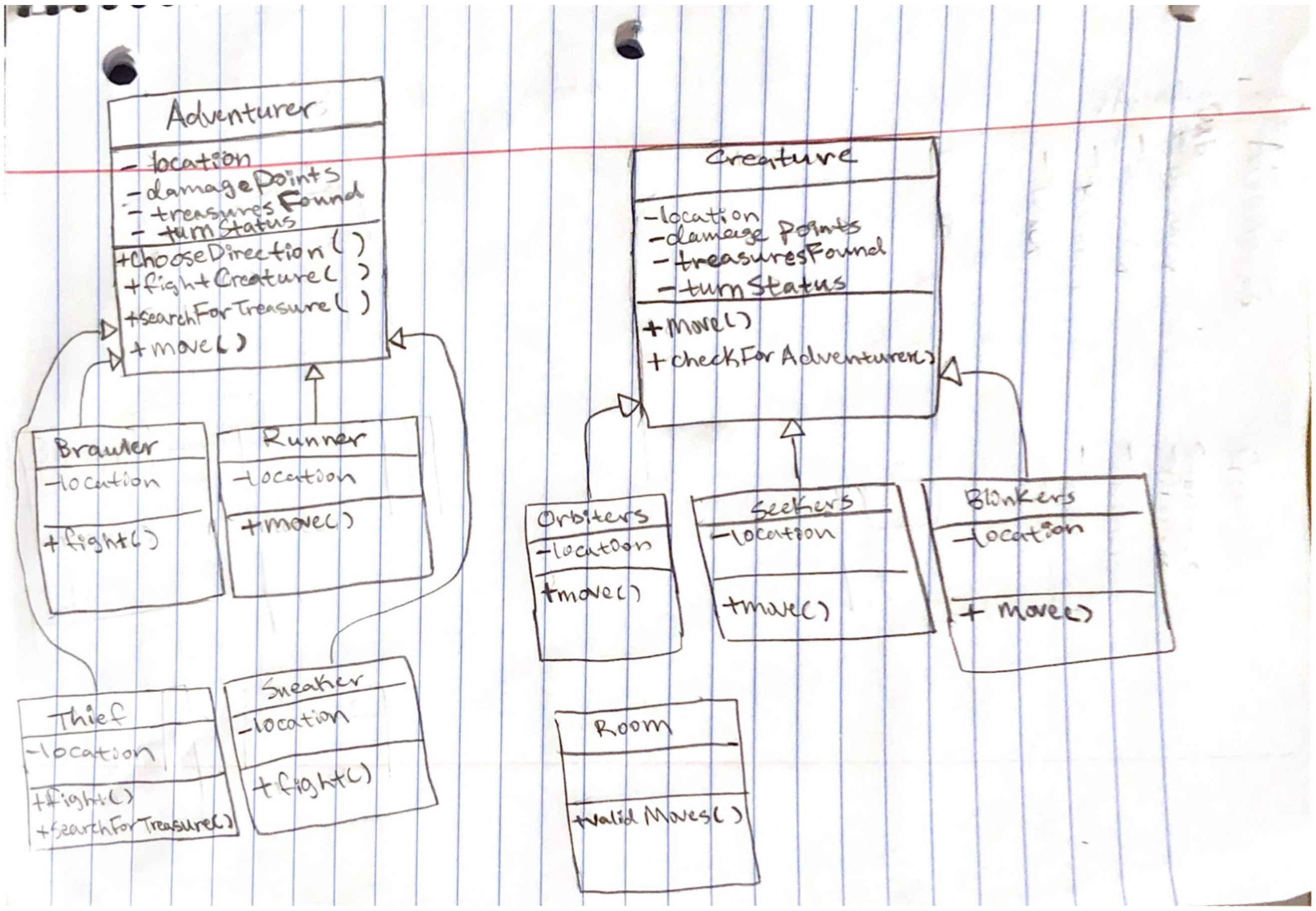
```
public class Encapsulate {

    // private variables can only be accessed inside of class
    private String s;

    // get method for age to access
    // private variable geekAge
    public int getString() { return s; }

    // method to set private variable s
    public void setStr(String newS) { s = newS; }
}
```

4.


Adventurer
-location
-damagePoints
-treasuresFound
-turnStatus
+chooseDirection( )
+fightCreature( )
+searchForTreasure( )
+move( )
Creature
-location
-damage Points
-treasuresFound
-turn Status
+move( )
+checkForAdventurer( )
Brawler
-location
+fight( )
Runner
-location
+move( )
Orbiters
-location
+move( )
Seekers
-location
+move( )
Blinkers
-location
+ move( )
Thief
-location
+fight( )
+searchForTreasure( )
Sneaker
-location
+fight( )
Room
+valid Moves( )