Teammates: Ben Courlang & Veda Jammula

1. (5 points) What does “design by contract” mean in an OO program? What is the difference between implicit and explicit contracts? Provide a text, pseudo code, or code example of each contract type.

“Design by Contract” means creating the elements in a contract for the main program, and elements being methods and classes. The result will not be as expected if the user does not go by the contract. There are preconditions, postconditions, and invariants. When thinking about implicit and explicit in programming, implicit means something that is already done for us behind the scenes. Explicit on the other hand has the instructions needed to make any changes. In terms of contracts, implicit contracts could be contracts that are hidden and not easy to find to understand the language being used while explicit will have the precondition clauses, postcondition clauses, and class invariants. An example of these two can be seen in the study done by Karine Arnout and Bertrand Meyer, “Finding Implicit Contracts in .NET Libraries”. An implicit class invariant could be “The default initial capacity for an ArrayList is 16”, which “implies that the capacity of the created object is greater than zero”(Meyer and Arnout). We can also do a comparison of implicit changing to explicit, where implicit is “the zero-based index of the first occurrence of value within the entire ArrayList, if found; otherwise, -1” and the explicit version is “If value appears in the list, the result is the index of the first occurrence, hence greater than or equal to zero (.NET list indexes are indexed starting at zero) and less than Count, the number of elements in the list” (Meyer and Arnout).

2. (5 points) What are three ways modern Java interfaces differ from a standard OO interface design that describes only abstract method signatures to be implemented? Provide a Java code example of each.

Three ways that modern Java interfaces differ from standard designs such as Abstract classes is 1. Within the modern interfaces, everything defined is assumed public while abstract classes can have these access modifiers that won’t be assumed public.

CODE BELOW TAKEN FROM GEEKSFORGEEKS:
Interface
—----------

```
class B {

    // Main driver method
    public static void main(String[] args)
    {

        // Creating object of type class A
        A b = new A();
```

```
        // Now calling the method m1()
        A.m1();
    }
}
```

Abstract Class

—----------

```
abstract class Vehicle {

    // Declaring an abstract method getNumberOfWheel
    abstract public int getNumberOfWheel();
}

// Class 2
// Helper class extending above abstract class
class Bus extends Vehicle {

    // Giving the implementation
    // to the  parent abstract method
    public int getNumberOfWheel() { return 7; }
}

// Class 3
// Helper class extending above abstract class
class Auto extends Vehicle {

    // Giving the implementation
    // to the  parent abstract method
    public int getNumberOfWheel() { return 3; }
}

// Class 4
// Main class
public class GFG {

    // Main driver method
    public static void main(String[] args)
    {

        // Creating Bus object
        Bus b = new Bus();

        // Creating Auto object
        Auto a = new Auto();
```

```
        System.out.println("Number of wheels in bus is"
                           + " " + b.getNumberOfWheel());
        System.out.println("Number of wheels in Auto is"
                           + " " + a.getNumberOfWheel());
    }
}
```

2. A interface can't declare constructors or deconstructors while abstract classes can

—----------

```
public abstract class Bike {

   int speed;

   public Bike(int speed) {
      this.speed = speed;
   }
}
```

3. The interface can inherit many interfaces but not a class, while an abstract class can inherit a class and as many interfaces as wanted.

CODE FROM [ORACLE DOCUMENTATION]

—----------

```
abstract class X implements Y {
  // implements all but one method of Y
}

class XX extends X {
  // implements the remaining method in Y
}
```

3. (5 points) Describe the differences and relationship between abstraction and encapsulation. Provide a text, pseudo code, or Java code example that illustrates the difference.

Encapsulation is the idea of bringing a bunch of data together under one unit and being able to manipulate the code and data as whole. It helps keep the data safe and not accessed from outside code. In short, it is data hiding while abstraction will only show the main or necessary information to the user. It will ignore the details that are not as important and the identity of an object is different from other ones of similar type. Abstraction deals with what should be done while encapsulation works towards answering how it should be done. Abstraction can be seen with abstract classes or

encapsulation while encapsulation is done using access modifiers. The relationship between the two is the idea of hiding certain or all information from other code.

CODE FROM JAVAtPoint

Encapsulation

—----------

```
public class EncapsulationDemo

{

public static void main(String[] args)

{

//creating instance of Account class

Account acc=new Account();

//setting values through setter methods

acc.setAcc_no(7560504000L);

acc.setName("Mark Dennis");

acc.setEmail("md123@gmail.com");

acc.setAmount(500000f);

//getting values through getter methods

System.out.println(acc.getAcc_no()+" "+acc.getName()+" "+acc.getEmail()+"
"+acc.getAmount());

}

}
```

Abstraction

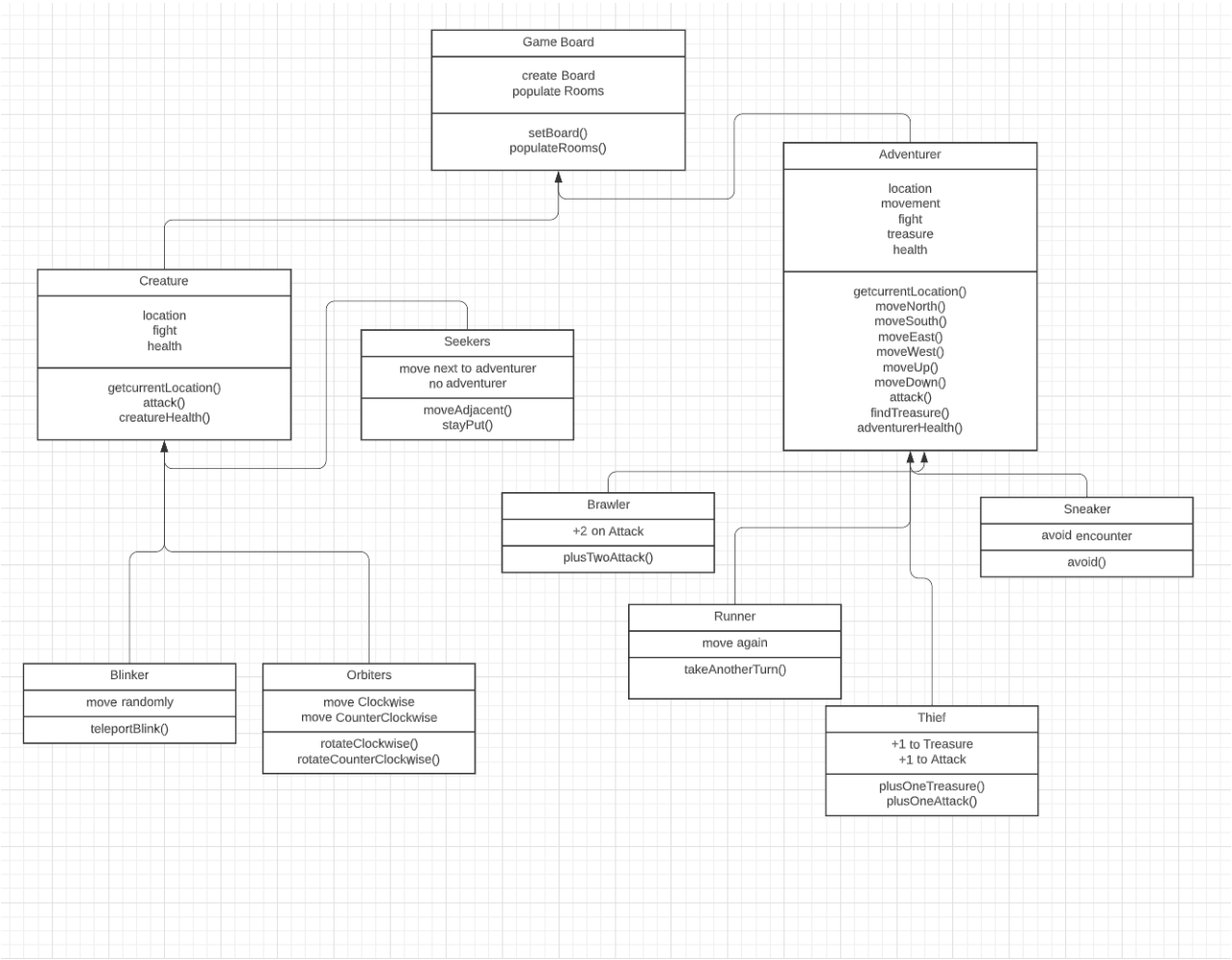
—----------

```
abstract class Shape
{
//abstract method
```

```
//note that we have not implemented the functionality of the method
public abstract void draw();
}
class Circle extends Shape
{
//implementing functionality of the abstract method
public void draw()
{
System.out.println("Circle!");
}
}
//main class
public class Test
{
public static void main(String[] args)
{
Shape circle = new Circle();
//invoking abstract method draw()
circle.draw();
}
}
```

4. (5 points) Draw a UML class diagram for the RotLA simulation described in Project 2.2. The class diagram should contain any classes, abstract classes, or interfaces you plan to implement to make the system work. Classes should include any key methods or attributes (not including constructors). Delegation or inheritance links should be clear. Multiplicity and accessibility tags are optional. Design of this UML diagram should be a team activity if possible and should help with eventual code development.

DIAGRAM PICTURE INCLUDED BELOW
completed as a team - Veda Jammula & Ben Courlang


Game Board
create Board
populate Rooms
setBoard()
populateRooms()
Adventurer
location
movement
fight
treasure
health
getcurrentLocation()
moveNorth()
moveSouth()
moveEast()
moveWest()
moveUp()
moveDown()
attack()
findTreasure()
adventurerHealth()
Creature
location
fight
health
getcurrentLocation()
attack()
creatureHealth()
Seekers
move next to adventurer
no adventurer
moveAdjacent()
stayPut()
Brawler
+2 on Attack
plusTwoAttack()
Sneaker
avoid encounter
avoid()
Runner
move again
takeAnotherTurn()
Blinker
move randomly
teleportBlink()
Orbiters
move Clockwise
move CounterClockwise
rotateClockwise()
rotateCounterClockwise()
Thief
+1 to Treasure
+1 to Attack
plusOneTreasure()
plusOneAttack()