Supervision 2

1.

A class can have instances, but an abstract class cannot, it only can be used by being inherited by some non-abstract class.

An interface is a group of abstract methods. If a class implements a interface, it must override all the methods in this interface.

For example, List is an abstract class, so if you try to compile code contains

List<MyClass> list = new List<>(); The compiler will tell you ‘List is abstract, cannot be instantiated’ . But ArrayList is an extension of List and it is a non-abstract class, hence you can write

List<MyClass> list = new ArrayList<>(); and that will be compiled.

If a class implement an interface, let’s say,

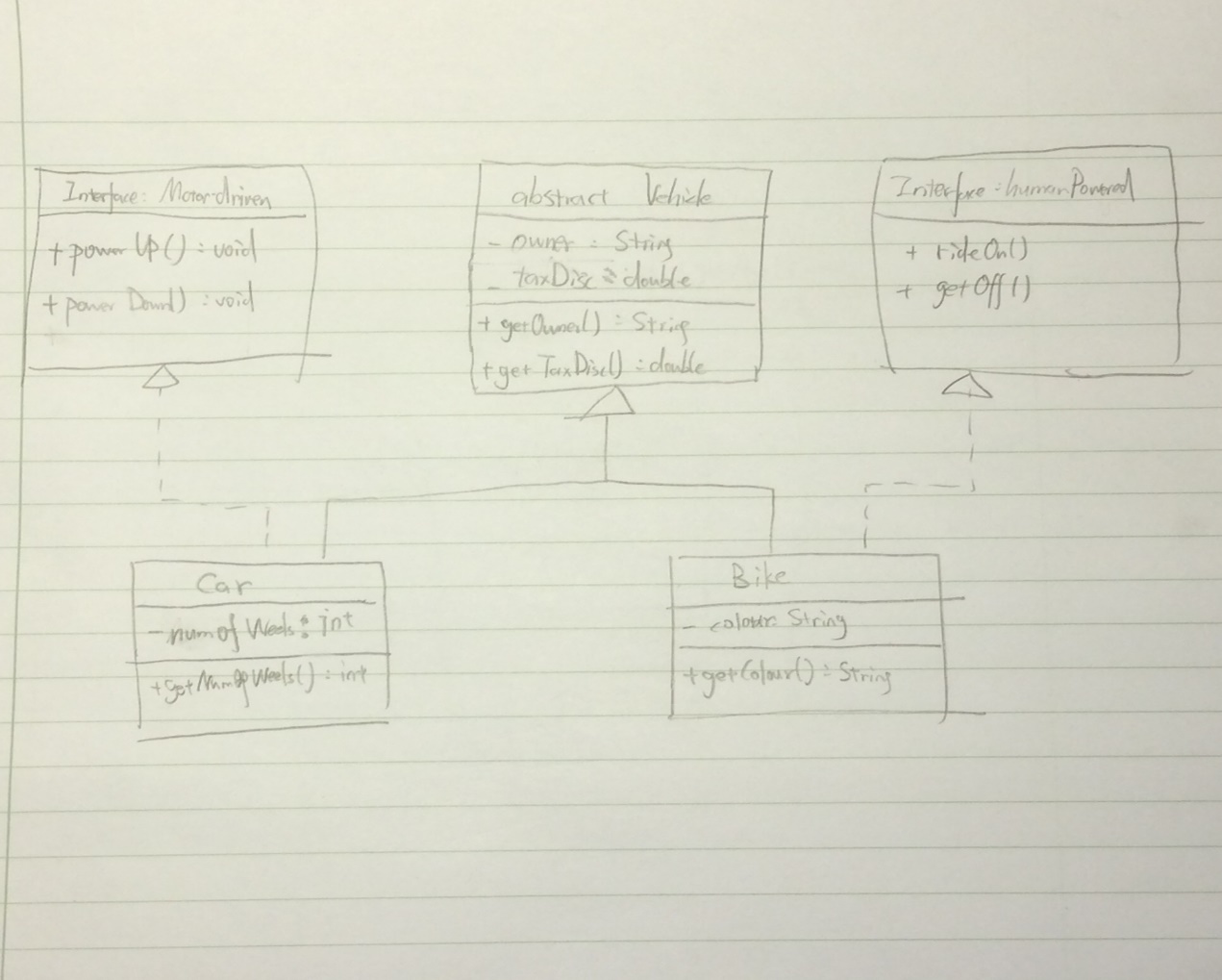
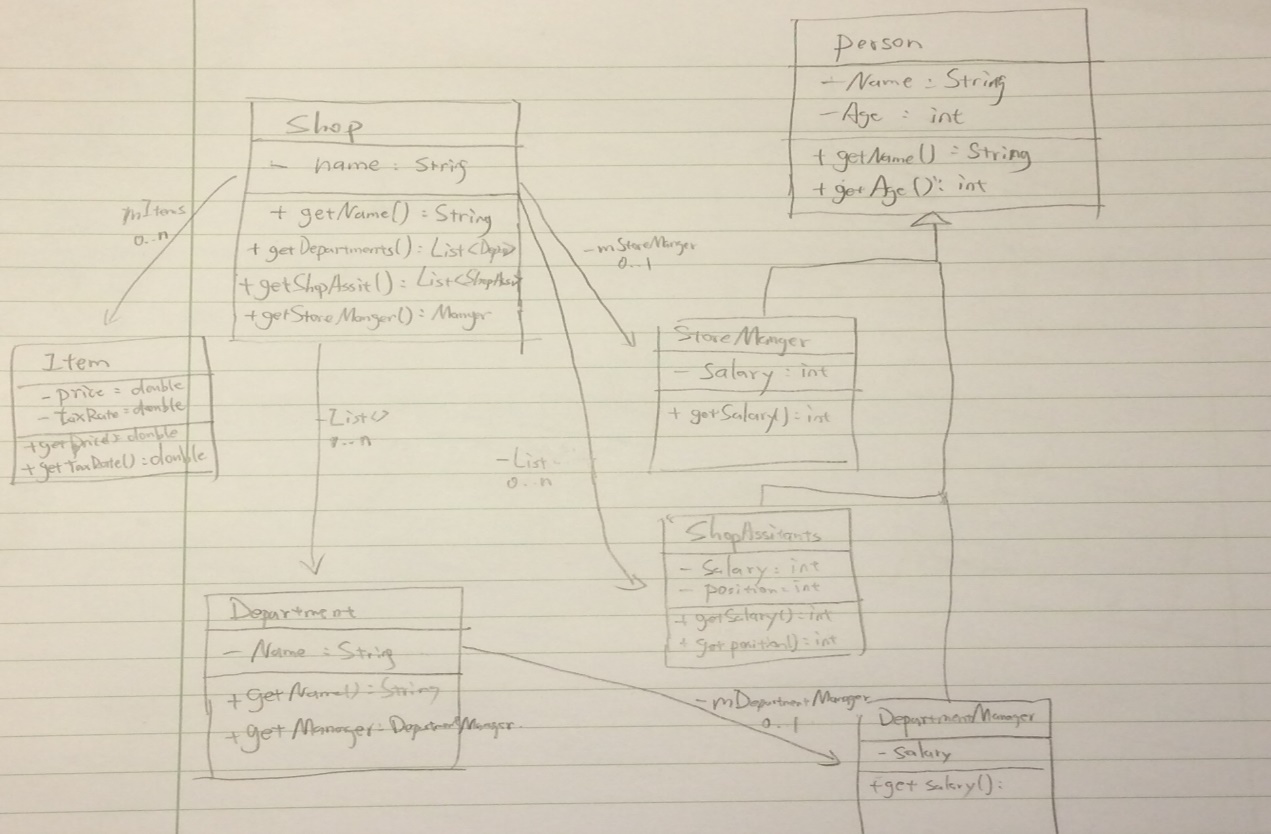
public class MyClass implements MyInterface{

………

………

}

It must override the abstract methods in MyInterface.

2.

3.

Modularity means that your programme is separated into different blocks connected by methods provided by themselves. For example, in question 2a, code about ‘Person’ and code about ‘Department’ are separated into two classes.

Code re-use means don’t repeat your code, just put it into a higher hierarchy and extend it. In question 2b, both motor-driven vehicle and human-powered vehicle have an owner and a tax disc, hence, these states and methods are written in a vehicle class.

Encapsulation means that every class is a black-box, you can only use the methods provided by the class but never touch the inner part. In question 2, almost every state is set private.

4.

Given that there is a class MyClass and its subclass MySubClass , with some methods, for example, myMethod() overridden by the subclass, and there is a MyClass reference mClass point to a MySubClass instance. Once we call mClass.myMethod(), in JVM it runs a MySubClass.myMethod()

Example code:

public class MySubClass extends MyClass {  
 public void print(){  
 System.*out*.println("MySubClass");  
 }  
}

public class MyClass {  
 public void print(){  
 System.*out*.println("MyClass");  
 }  
 public static void main(String args[]){  
 MyClass mClass = new MySubClass();  
 mClass.print();  
 }  
}

this programme prints “MySubClass”

One of the good things of polymorphism is that you don’t have to figure out which kind of subclass it is when you are dealing with many instances of same class but different subclasses. For example you have a array of Animal class which can be Cat or Dog or Human, and each of these subclass has different method of eat() you can write

for( Animal i : animals){  
 i.eat();  
}

and everybody eats well.

5.

public interface NinjaInt {  
 public void ninjaBehaviour();  
}

public class Ninja implements NinjaInt {  
 @Override  
 public void ninjaBehaviour() {  
 System.*out*.println("I'm a Ninja");  
 }  
}

public class NinjaEmployee extends Employee implements NinjaInt {  
 private Ninja ninjaSoul = new Ninja();  
  
 @Override  
 public void ninjaBehaviour() {  
 ninjaSoul.ninjaBehaviour();  
 }  
}

6.

The garbage collector will trace every reference from the main method, if an object cannot be reached, i.e. it’s not used anymore, the garbage collector will set this piece of memory “reusable” , i.e. a new class can overwrite this piece of memory.

7.Override the toString() method in Person class.

8 and 9.

package Q9;  
  
*/\*\*  
 \* Created by Anchor on 2016/11/24.  
 \*/*public class Game {  
 private int mDifficulty;  
 private int mTarget;  
 private int numOfGuess = 0;  
 private int mMaxNumOfGuess = 0;  
 public Game(int difficulty, int maxNum){  
 mDifficulty = difficulty;  
 java.util.Random r = new java.util.Random();  
 mTarget = r.nextInt() % mDifficulty + 1;  
 mMaxNumOfGuess = maxNum;  
 }  
 public boolean response(int guess) {  
 numOfGuess++;  
 if (guess > mTarget) {  
 if (numOfGuess == mMaxNumOfGuess) {  
 System.*out*.println("Game Over!");  
 return false;  
 }  
 System.*out*.println("go lower");  
 } else if (guess < mTarget) {  
 if (numOfGuess == mMaxNumOfGuess) {  
 System.*out*.println("Game Over!");  
 return false ;  
 }  
 System.*out*.println("go higher");  
 } else  
 {  
 System.*out*.println("correct");  
 return false;  
 }  
 return true;  
 }  
}

package Q9;  
  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
  
*/\*\*  
 \* Created by Anchor on 2016/11/24.  
 \*/*public class HiLo {  
  
 private Game mGame;  
  
 public HiLo(Game game) {  
 mGame = game;  
 }  
  
 public void play() throws IOException {  
 String response = "";  
 BufferedReader in = new BufferedReader(new InputStreamReader(System.*in*));  
  
 while (!response.equals("q")) {  
 response = in.readLine();  
 if (response.startsWith("g")) {  
 if (mGame == null) System.*out*.println("Please set difficulty and max guesses");  
 else {  
 int guess = Integer.*parseInt*(response.substring(2, response.length()));  
 if (!mGame.response(guess)) {  
 System.*out*.println("Please set difficulty and max guesses");  
 mGame = null;  
 }  
 }  
 } else if (response.startsWith("s")) {  
 String[] mArgs = response.split(" ");  
 mGame = new Game(Integer.*parseInt*(mArgs[1]), Integer.*parseInt*(mArgs[2]));  
 }  
  
 }  
 }  
  
 public static void main(String args[]) {  
 try {  
 Game g = null;  
 HiLo hl = new HiLo(g);  
 hl.play();  
 } catch(IOException e) {  
 System.*out*.println(e.getMessage());  
 }  
 }  
}