1. (1) Classes are the ones don't contain abstract methods. Abstract classes are the ones that part of or all of the methods are abstract methods. Interfaces can only have static final fields, methods without body, and default methods.

   (2) One class can only inherit one abstract class, but can implement multiple interfaces.

   (3) Fields and methods in abstract methods can have different accessibilities but those in interfaces are all public.

   (4) Fields in interfaces must be given right in interfaces and cannot be re-declared or modified in classes implementing interfaces, which is not the case in abstract classes.

   (5) Classes implementing interfaces must give implementations of all the methods in interfaces but classes extending abstract classes can implement only some of the abstract methods in parent class and leave the rest of them abstract.

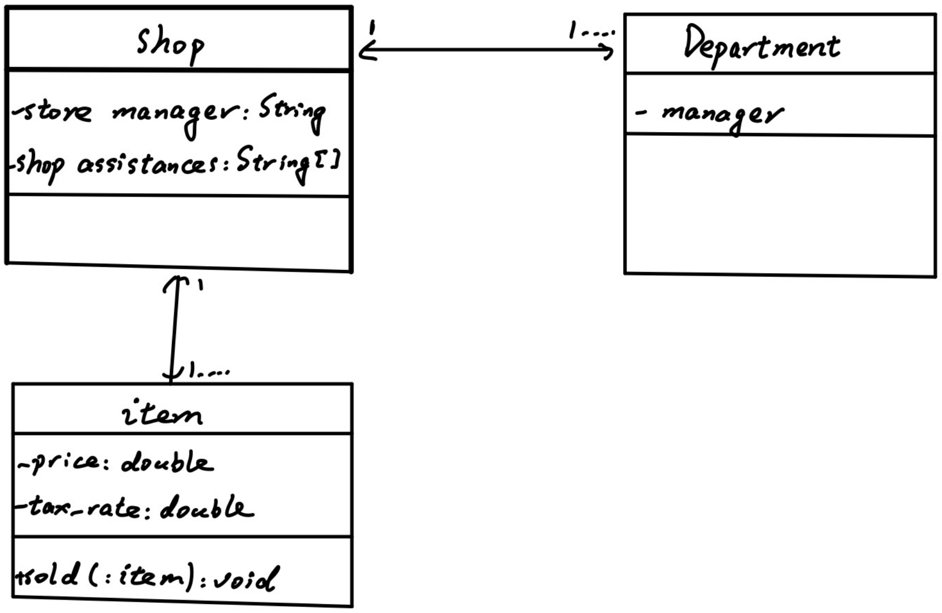
```
class Ex1{
    int x;

    int y(){
        return 1;
    }
}
```

```
abstract class Ex1{
    int x;

    abstract int y();

    private int z(){
        return 1;
    }
}
```

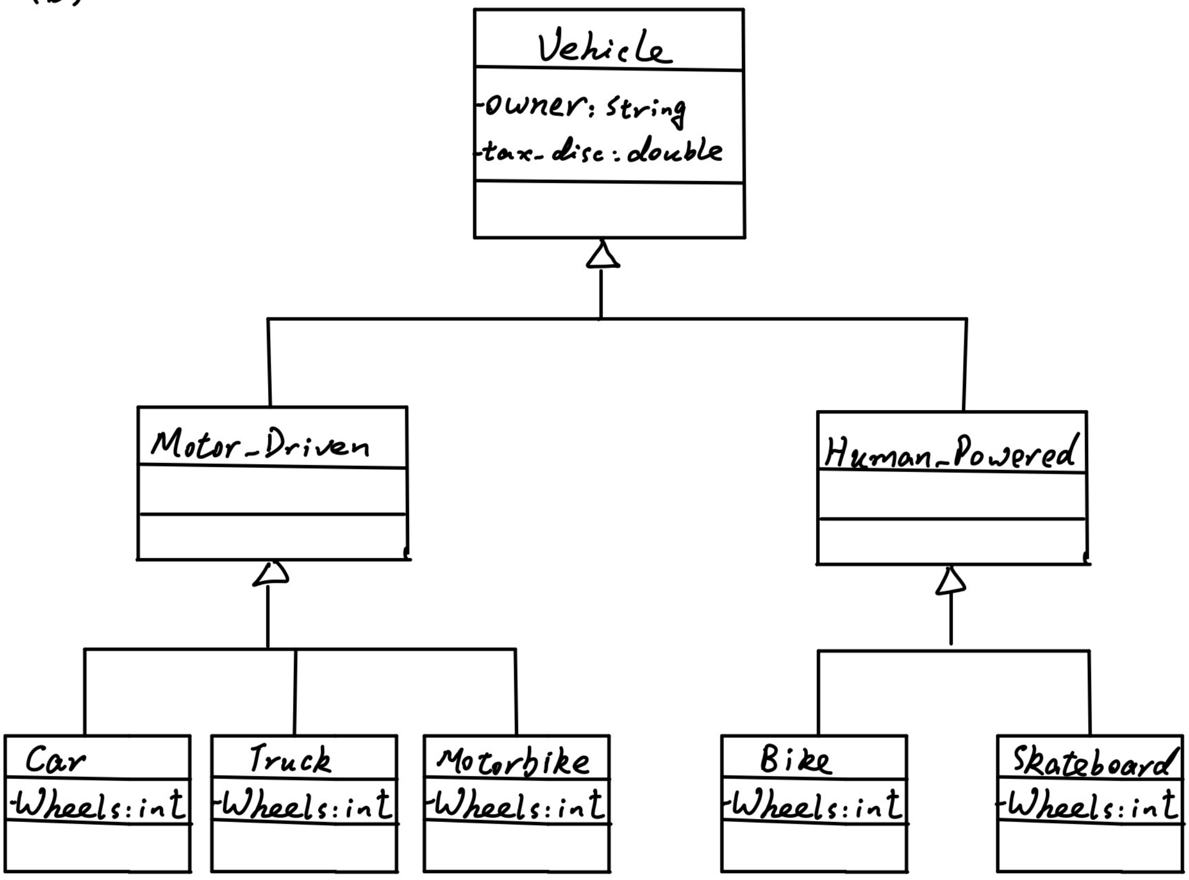
```
interface Ex1{
    public static final int x = 1;

    int y();

    default int z(){
        return y() + 1;
    }
}
```

2.

(a)


Shop
-store manager: String
-shop assistances: String[]
1
1....
Department
- manager
1
1....
item
-price: double
-tax_rate: double
+sold (: item): void

(b)


Vehicle
-owner: String
-tax_disc: double
Motor_Driven
Human_Powered
Car
-Wheels: int
Truck
-Wheels: int
Motorbike
-Wheels: int
Bike
-Wheels: int
Skateboard
-Wheels: int

3. (a) Each class represents a sub-unit of code that can be developed, tested and updated independently. For example, I can easily add another kind of vehicle in 2. (b) without changing other code.

   (b) The same code can be used in different programmes without modification.

   (c) Encapsulation means programmes hide its internal state (e.g by setting fields as private) and bundles methods with state to access state from methods.

4. Polymorphism means an object can have different types.

   Dynamic polymorphism means running the method or accessing the field in the child class.

   It's useful because it enables us to give different implementations of a single method. For example, we want to give different implementations of returning the number of wheels of different types of vehicles. We can have subclasses of Car and Bike and so on of the superclass Vehicle. Dynamic polymorphism lets us to run the method in the respective subclass.

5. (1) Let NinjaEmployee inherit Employee class.

   (2) Create a Ninja interface.

(3) Let Ninja class implement Ninja interface.

(4) Let NinjaEmployee implement Ninja interface as well.

(5) Create a Ninja object inside NinjaEmployee class.

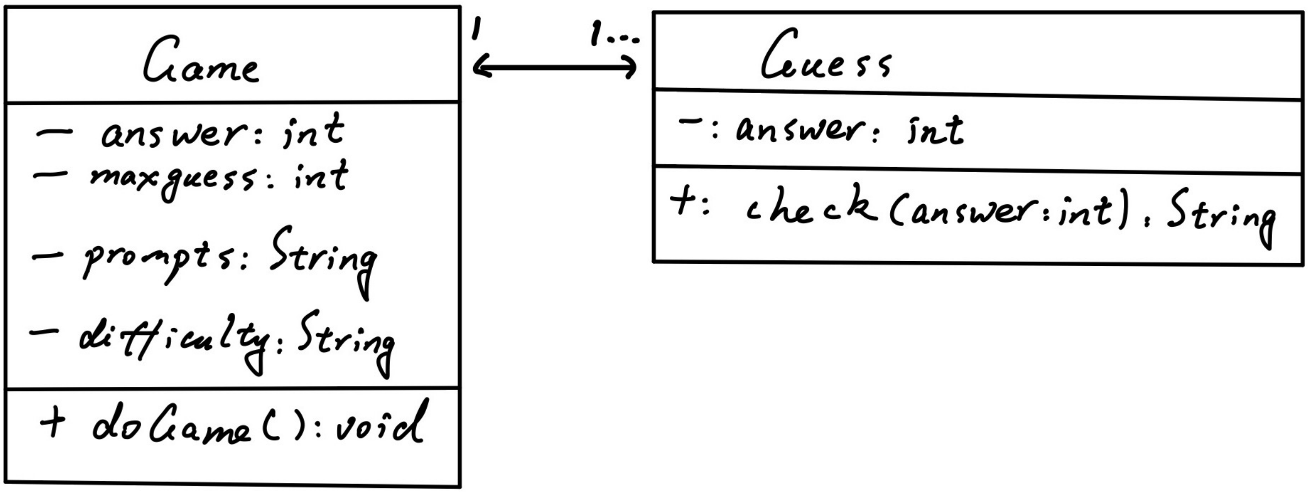
(6) Call methods of Ninja object for NinjaEmployee class to implement Ninja interface.

6. (a) Garbage Collection starts with a list of all references you can get to, follows all references recursively, marks every object and deletes all objects that were not marked.

   (b) Finalizer method is called when an object is about to collected. The problem is that finalizer may not be called and is unpredictable at all so you can't use it as a deconstructor.

7. By changing the implementation of overriding of toString method.

8.



```
//Guess.java
public class Guess {
    private int ans;

    public Guess(int ans){
        this.ans = ans;
    }

    public String check(int v){
        if (ans == v)
            return "Correct!";
        return v < ans ? "Go higher." : "Go lower.";
    }
}
```

```
//Game.java
```

```java
import java.util.Scanner;

import javax.script.ScriptContext;

public class Game {
    int ans, maxGuess;
    String difficulty;

    static final String prompt1 = "Please enter your name.";
    static final String prompt2 = "Game lost. Do you want one more game?";
    static final String prompt3 = "Game wins. Do you want one more game?";

    static Scanner scanner = new Scanner(System.in);

    public Game(String difficulty){
        this.ans = (int)(Math.random() * 100);
        this.maxGuess = difficulty == "Easy" ? ans : (int)(2 * Math.log(ans) / Math.log(2));
        this.difficulty = difficulty;
    }
    public int doGame(){
        Guess g = new Guess(ans);

        while(maxGuess > 0){
            maxGuess--;

            int response =
Integer.parseInt(scanner.nextLine());
            String info = g.check(response);

            if(info == "Correct!"){
                System.out.println(prompt3);
                return 1;
            }
            else if(maxGuess > 0)
                System.out.println(info);
            else System.out.println(prompt2);
        }

        return 0;
    }

    public static void printInfo(String name,
int wins1, int wins2){
        System.out.println("Name:  " + name);
        System.out.println("Number of wins at easy mode: " + Integer.toString(wins1));
        System.out.println("Number of wins at difficult mode: " + Integer.toString(wins2));
    }

    public static void main(String[] args){
        System.out.println(prompt1);

        int winsEasy = 0, winsDiff = 0;
        String name = scanner.nextLine();
        String mode, response;

        do{
            System.out.println("Please pick a mode");

            mode = scanner.nextLine();
            Game game = new Game(mode);

            System.out.println("Please make a guess");

            if(game.doGame() == 1){
                if(mode.equals("Easy"))
                    winsEasy++;
                else winsDiff++;
            }

            printInfo(name, winsEasy,
winsDiff);

            response = scanner.nextLine();
        }while(response.equals("Yes"));
    }
}
```