Questions and Exercises

These questions and exercises is an opportunity to see what you've learnt from the lecture as well as practice the new things we've been talking about. In other words, these questions and exercises are completely optional but it's recomended to do them. In the end of the document you will find the answers to the questions as well as possible solutions to the exercises, note that one can solve an exercise in different ways. There will also be some suggestions about what one could code if one want to continue with some more advanced things. These suggestions will not come with a possible solution and might include things that haven't been covered in the lecture.

Question 1
What are the the two only things can can put in an interface?

Question 2
What's the difference between an abstract class and an interface?

Question 3
What does it mean to extend multiple interfaces when creating an interface?

Exercise
Write an interface called IConsumable that is used in the classes Food and Poison. The interface should have a method called consumedBy that requires one Player paramater(you'll have to write a simple Player class as well). When food is consumed the Player's hungerbar should change while consuming posion should damage the player's health.

Further explorations
Start with the example from the lecture(can be found on the lecture page) and improve it. Use it to create a simple game where the Player can attack monsters and containers and open chests and similar things to get loot. This could be combined with the code from the last lecture, if not however, make the game of any type (i.e. How the player find monsters/chests is not important).

Answers and solutions

Answer to Question 1
The only things that an interface can consist of are public static final variables (constant values) and public abstract methods (methods to override). Since this is the case you don't have to specify the public static final and the public abstract in front of the variables and methods, that's done anyways.

Answer to Question 2
Abstract classes can have constructors, defined methods and any type of field, interfaces can't. To use an abstract class you'll have to extend it with your subclass, however when you have an interface you can just implement it from any class, it doesn't matter how your class structure looks like. Furtheremore you can implement multiple interfaces for the same class.

Answer to Question 3
When it comes to classes one can just extend one class, in other words a class can only have one superclass. However, for interface you can extend multiple ones. This will give your new interface all the methods from all the superinterfaces, basically forcing the coder to override all those methods when implementing your new interface. Of course the coder will also have to override any new methods written in the interface(no only the ones from the superinterfaces).

Possible solution to Exercise

```
public interface IConsumable {

    void consumeBy(Player player);

}

public class Player implements IConsumable {

    private String name;
    private int health;
    private int hunger;

    public Player(String name) {
        this.name = name;
        health = 100;
        hunger = 20;
    }

    public String getName() {
        return name;
    }


    public int getHealth() {
        return health;
    }

    public int getHunger() {
        return hunger;
    }

    public void damageHealth(int val) {
```

```
        health -= val;
    }

    public void restoreHunger(int val) {
        hunger -= val;
    }

    //just for fun, not part of the exercise
    @Override
    public void consumeBy(Player player) {
        if (player.equals(this)) {
            System.out.println("Don't eat yourself, are you mad?");
        }else{
            System.out.println("Don't eat me!");

            //self defence
            player.health = 0;
        }
    }


}

public class Food implements IConsumable {

    private int value;

    public Food(int value) {
        this.value = value;
    }

    @Override
    public void consumeBy(Player player) {
        player.restoreHunger(value);
    }

}

public class Poison implements IConsumable {

    private int value;

    public Poison(int value) {
        this.value = value;
    }

    @Override
    public void consumeBy(Player player) {
```

```
        player.damageHealth(value);
    }

}
```