

Lab: Dependency Inversion and Interface Segregation Principles

Problems for exercises and homework for the ["Java OOP Advanced" course @ SoftUni](#).

Part I: Dependency Inversion

1. System Resources

You are given a **GreetingClock**. It has the following behavior:

- if hour < 12, prints "Good morning..."
- if hour < 18, prints "Good afternoon..."
- else prints "Good evening..."

Refactor the code so that it conforms to the Dependency Inversion principle.

* Optional: Introduce **Strategy Design Pattern**

Solution

Create a new interface **TimeProvider** that has a single method **getHour():int**

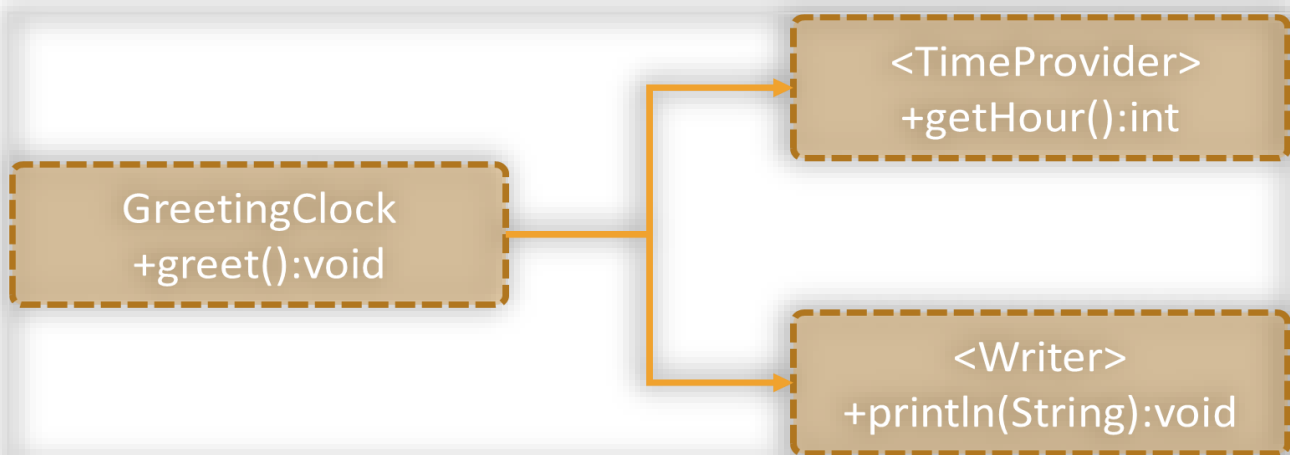
Create an implementation of the interface that can provide the hour (You can use **LocalTime** java class)

Inject **TimeProvider** through the clock's constructor

Create a new interface **Writer** that has a single method **println(String):void**

Create an implementation that uses a class that can print to the console

Inject **Writer** through the clock's constructor and use it to write to the console



2. Services

You are given some classes:

- **OnlineStoreOrder**
- **SmsNotificationService**

- **EmailNotificationService**

Once the order is **processed** it should **send** the proper **notifications** through the services, if they are active.

Refactor the classes so that they conform the Dependency Inversion principle

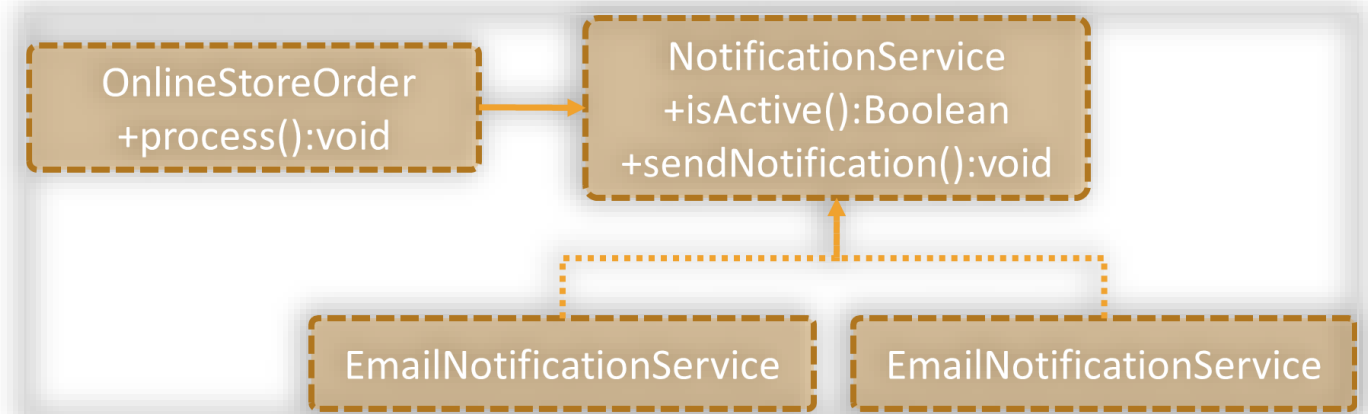
* Optional: Introduce **Composite Design Pattern**

Solution

Introduce a new interface called **NotificationService** and make all notification services to implement it

Either inject variable number of parameters in the **OnlineStoreOrder** or create a **CompositeNotificationService** implementation

CompositeNotificationService should implement **NotificationService**



3. Employee Info

You are given the classes:

- **Employee**
- **EmployeeInfoProvider**
- **EmployeeDatabase**
- **ConsoleFormatter**

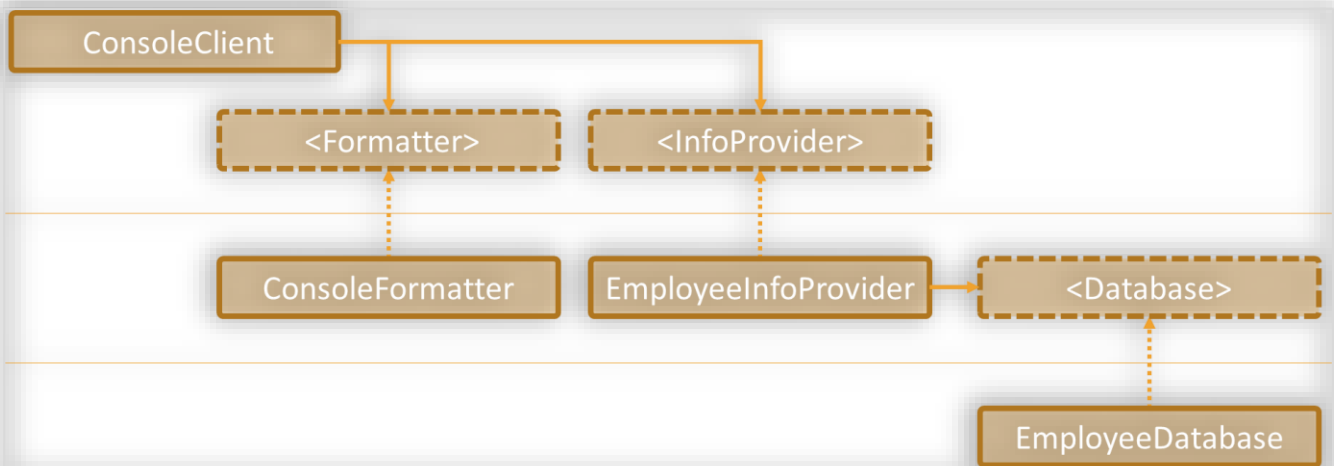
EmployeeInfoProvider provides a list of employees to be formatted to a String by the **ConsoleFormatter**. Then the string is printed to the console. Refactor the classes and the structure to conform to the **Dependency Inversion** principle.

Hints

Create an abstraction between every layer of the application

Create a **ConsoleClient** which will run the application. Define the abstractions by using as a guide the **ConsoleClient**'s needs (It needs some kind of **Formatter** and some kind of **InfoProvider**)

EmployeeInfoProvider needs a database, so define abstraction by looking at what methods **EmployeeInfoProvider** needs



Part II: Interface Segregation

4. Recharge

You are given a library with the following classes

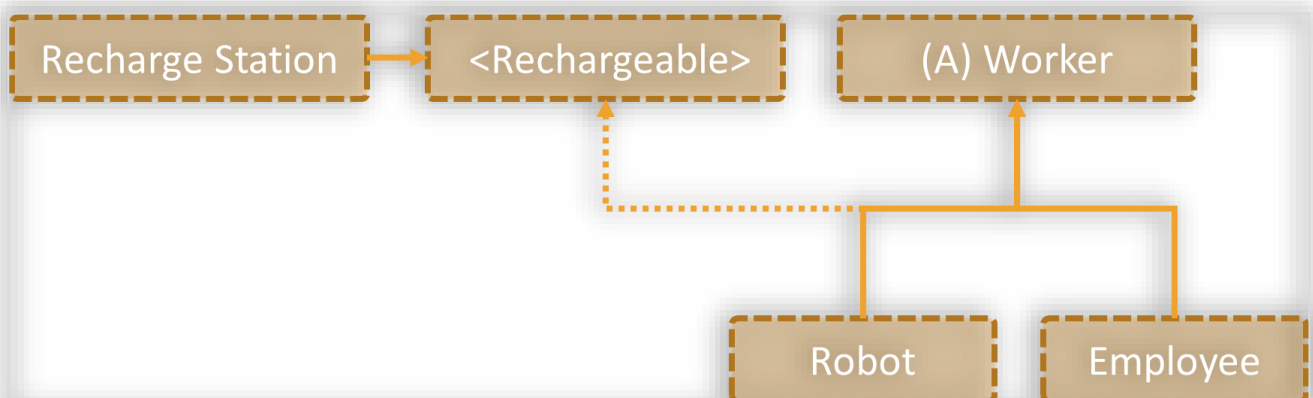
- Worker implements Sleeper
- Employee extends Worker
- Robot extends Worker
- RechargeStation

If you inspect the code, you can see that some of the classes have methods that they can't use (throw **UnsupportedOperationException**) which is clear indication that the code should be refactored.

Refactor the structure so that it conforms to the **Interface Segregation** principle.

Hints

Make the **Robot** to extend **Worker** and at the same time to implement **Rechargeable**



5. Security Door

You are given:

- **SecurityManager**
- abstract class **SecurityCheck**
- interface **SecurityUI**

SecurityManager which can interact with a user by validating his key card or by getting his pin code. Both methods are provided by an interface called **SecurityUI**. The validation is performed by the appropriate **SecurityCheck** class.

Refactor the structure so that it conforms to the **Interface Segregation** principle.

Hints

Split **SecurityUI** into smaller role interfaces, one for each **SecurityCheck** class

