Fachhochschule Nordwestschweiz
Hochschule für Technik

Software Architecture

Exercise – Gaming Platform (iteration #1)

BSc

Ingo Arnold

Copyright Notice

Exercise Opening
Overall Motivation

In this unit, we will **practically design a small system** as a solution to a given problem. The focus is less on functionality or final maturity of the system than on its architecture design.

Purpose is to understand how you **systematically follow a path from a given problem to a corresponding solution** and that you understand the evolutionary nature of architecture (i.e., architecture as an iterative process that is accompanied by uncertainties).

This solution will accompany us in **going through the architecture process** – albeit rudimentarily. This also means that at the end of the process we will have an executable version of the system in Java.

Exercise Opening
Motivation

A gaming platform is to be developed—i.e. a platform to which players can log in and subsequently play the games available.

- The gaming platform provides players with a unified gaming ecosystem and allows players to access all games registered with the platform.
- At the same time, the gaming platform provides game providers with access to a large group of players.
- Players must first register on the gaming platform.
- Registration requires the player to enter their first and last name, a unique account name, a password of their choice, and a role (i.e., either player or administrator or game provider).
- After players have registered with the platform, they can enter it.
- For this purpose, a player or administrator logs in to the platform by providing account name and password.
- After entering the platform, players will find a catalogue that they can view and select games from.

Exercise Opening
Motivation

A gaming platform is to be developed—i.e. a platform to which players can log in and subsequently play the games available.

- Before this catalogue shows games, they must have been previously registered for the platform.
- If an actor finds an interesting game in the catalogue, he can select it and register for the game.
- In doing so, the game is entered into the actor's favourites list so that the player can start the game directly from there in the future.
- After a game is started, players play the game.
- We limit the games for now so that all games are played against the computer.
- A game ends when one of the players (human or computer) wins the game or the human player ends the game early.
- For the beginning, it is enough for us to simulate games.

Exercise Opening
Motivation

A gaming platform is to be developed—i.e. a platform to which players can log in and subsequently play the games available.

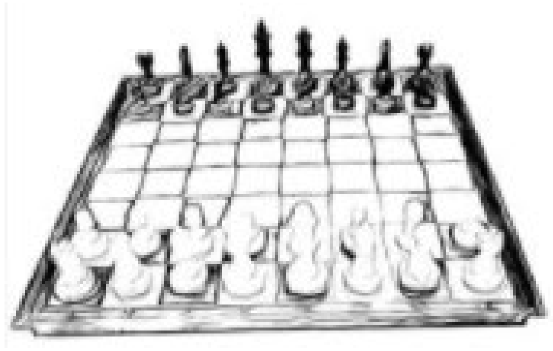
- A game simulation is to randomly determine a winner (i.e., player or computer). The duration of the simulated game (between 1 and 100 minutes) and the score resulting from the victory (between 1 and 100 points) should also be determined randomly.
- For each game there is a high score list.
- At the end of a (simulated) game, the winner, his opponent, the score as well as the duration are entered into the high score list of the respective game.
- The high score list of a game can be viewed by all players at any time.
- It is sufficient for screen outputs as well as possible screen inputs to direct these into a console output.
- Initially, a first prototype of the game platform is to be developed in a very short time.
- At the same time, it is expected that the original expectations for the game platform will be successively expanded.

Exercise Opening
Motivation

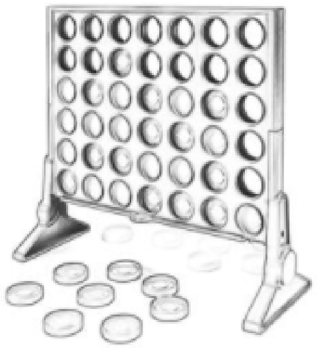
A gaming platform is to be developed—i.e. a platform to which players can log in and subsequently play the games available.

- Three games are to be initially registered on the platform.
- However, the games do not have to be implemented algorithmically – i.e., we assume the games to be closed entities.

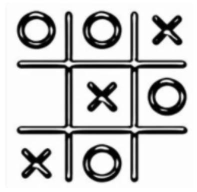
chess



connect four



tic tac toe



Requirements Elicitation
Use Case Overview and Use Cases

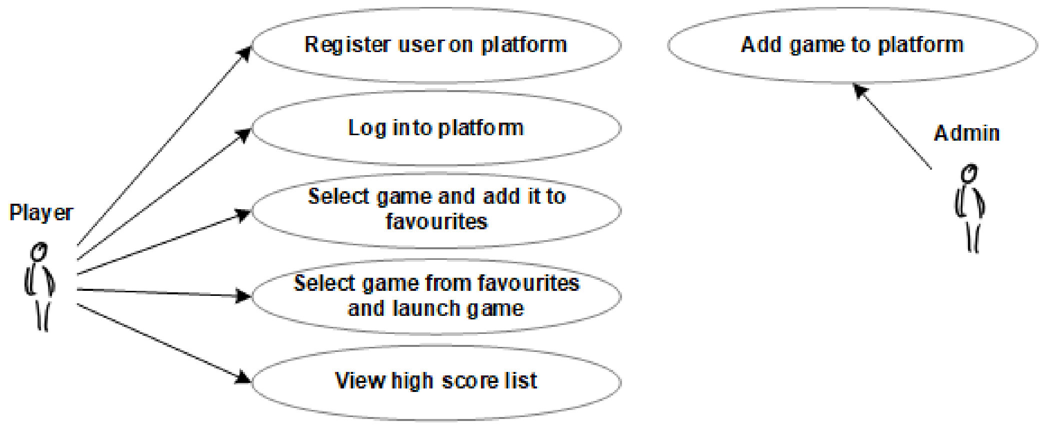
A first step has already been performed an initial set of functional requirements inherent in the previous description has been derived. In other words, we have created descriptions for *actors* and *functionalities* (i.e., *use cases*) expected from the platform.

A *use case overview* and descriptions have been created based on the schema below. Main focus has been on each Use Case's respective *flow of events*.

Use Case Name
Actor
Flow of Events
Precondition
Postcondition
Qualities

Requirements Elicitation
Use Case Overview and Use Cases

Below you see the **use case overview diagram** for the gaming platform. A use case overview diagram includes all major actors and use cases.


Register user on platform
Log into platform
Select game and add it to favourites
Select game from favourites and launch game
View high score list
Player
Add game to platform
Admin

Requirements Elicitation
Use Case Overview and Use Cases

Use case **add game to platform**.

Use Case Name	Add game to platform
Actor	Admin
Flow of Events	1. Actor creates game and sets its name. 2. Actor adds game to the gaming platform.
Precondition	Actor is logged into platform as admin.
Postcondition	New game is added to the gaming platform.
Qualities	

Requirements Elicitation
Use Case Overview and Use Cases

Use case **register user on platform**.

Use Case Name	Register user on platform
Actor	Player, Admin
Flow of Events	1. Actor initiates self-registration. 2. Actor registers with platform providing his name, first name, unique id, password (must be entered twice), and role (either «player» or «admin»). 3. Platform validates attributes and registers actor.
Precondition	
Postcondition	New player or admin is registered on platform.
Qualities	

Requirements Elicitation
Use Case Overview and Use Cases

Use case **log into platform**.

Use Case Name	Log into platform
Actor	Player, Admin
Flow of Events	1. Actor initiates login. 2. Actor provides gaming platform credentials (i.e., unique id and password). 3. Platform validates credentials and establishes user session.
Precondition	Actor is successfully registered on platform.
Postcondition	User session for actor is established.
Qualities	

Requirements Elicitation
Use Case Overview and Use Cases

Use case **select game and add it to favourites**.

Use Case Name	Select game and add it to favourites
Actor	Player, Admin
Flow of Events	1. Actor opens the game catalogue. 2. Actor views the game catalogue contents. 3. Actor selects game from game catalogue. 4. Selected game is added to actor's favourites list.
Precondition	Player is successfully logged into platform. Game catalogue is not empty. Selected game is not yet in the player's favourites list.
Postcondition	Game is added to actor's favourites list.
Qualities	

Requirements Elicitation
Use Case Overview and Use Cases

Use case **select game from favourites and launch game**.

Use Case Name	Select game from favourites and launch game
Actor	Player
Flow of Events	1. Actor opens the favourites list. 2. Actor views the favourites list contents. 3. Actor selects game from favourites list. 4. Actor launches a selected game. 5. Actor plays the game till it ends. 6. Game results are added to the game's high score list.
Precondition	Player is successfully logged into platform. Game was played and high score list is not empty.
Postcondition	Favourites list was viewed by an actor.
Qualities	

Requirements Elicitation
Use Case Overview and Use Cases

Use case **view high score list**.

Use Case Name	View high score list
Actor	Player
Flow of Events	1. Actor opens the high score list. 2. Actor views all game results in the high score list.
Precondition	Player is successfully logged into platform. High score list is not empty.
Postcondition	High score list was viewed by an actor.
Qualities	

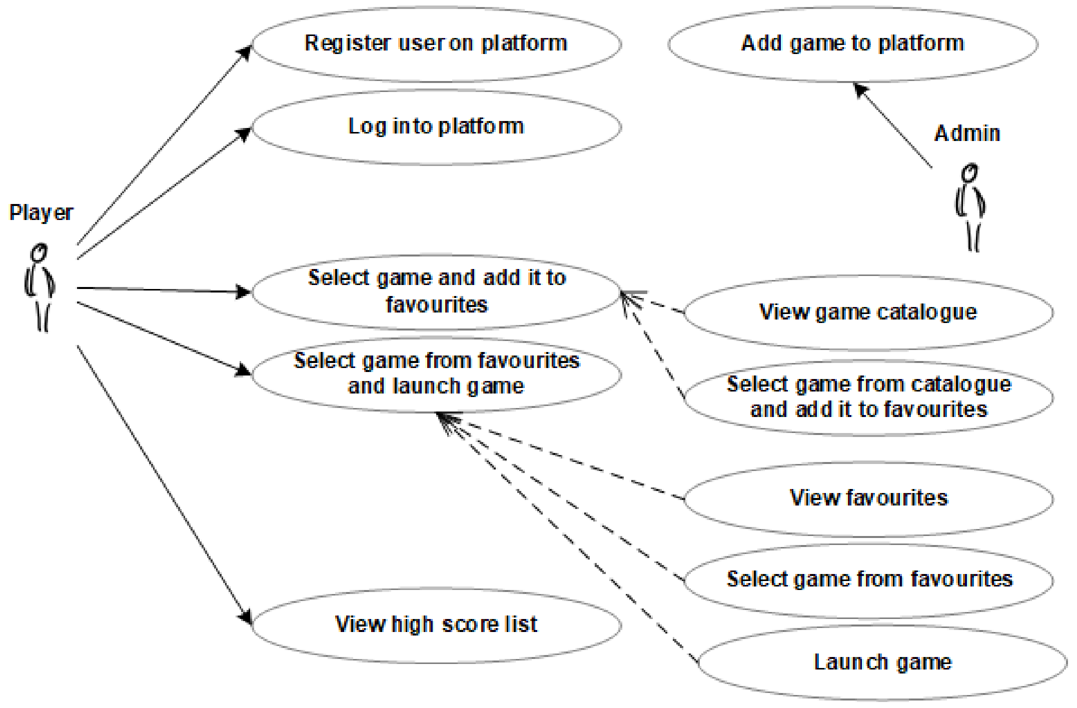
Requirements Elicitation
Use Case Overview and Use Cases

In a second round the **initial set of requirements** have been further refined where use cases were too coarsely grained, and hence should be broken down into smaller ones.

We'll see the results of this refinement on the following slides.

Requirements Elicitation
Use Case Overview and Use Cases

Below you see the **use case overview diagram** including the refined set of use cases.


Player
Admin
Register user on platform
Log into platform
Select game and add it to favourites
Select game from favourites and launch game
View high score list
Add game to platform
View game catalogue
Select game from catalogue and add it to favourites
View favourites
Select game from favourites
Launch game

Requirements Elicitation
Use Case Overview and Use Cases

Use case **view game catalogue**.

Use Case Name	View game catalogue
Actor	Player, Admin
Flow of Events	1. Actor opens the game catalogue. 2. Actor views the game catalogue contents.
Precondition	Player is successfully logged into platform. Game catalogue is not empty.
Postcondition	Game catalogue was viewed by an actor.
Qualities	

Requirements Elicitation
Use Case Overview and Use Cases

Use case **select game from game catalogue and add it to favourites**.

Use Case Name	Select game from catalogue and add it to favourites
Actor	Player
Flow of Events	1. Actor selects game from game catalogue. 2. Selected game is added to actor's favourites list.
Precondition	Player is successfully logged into platform. Game catalogue is not empty and selected game is not yet in the player's favourites list.
Postcondition	Game is added to actor's favourites list.
Qualities	

Requirements Elicitation
Use Case Overview and Use Cases

Use case **view favourites**.

Use Case Name	View favourites
Actor	Player
Flow of Events	1. Actor opens the favourites list. 2. Actor views the favourites list contents.
Precondition	Player is successfully logged into platform. favourites list is not empty.
Postcondition	Favourites list was viewed by an actor.
Qualities	

Requirements Elicitation
Use Case Overview and Use Cases

Use case **select game from favourites**.

Use Case Name	Select game from favourites
Actor	Player
Flow of Events	1. Actor opens favourites list. 2. Actor selects game from favourites list.
Precondition	Player is successfully logged into platform. Favourites list is not empty.
Postcondition	Game is selected by player.
Qualities	

Requirements Elicitation
Use Case Overview and Use Cases

Use case **launch game**.

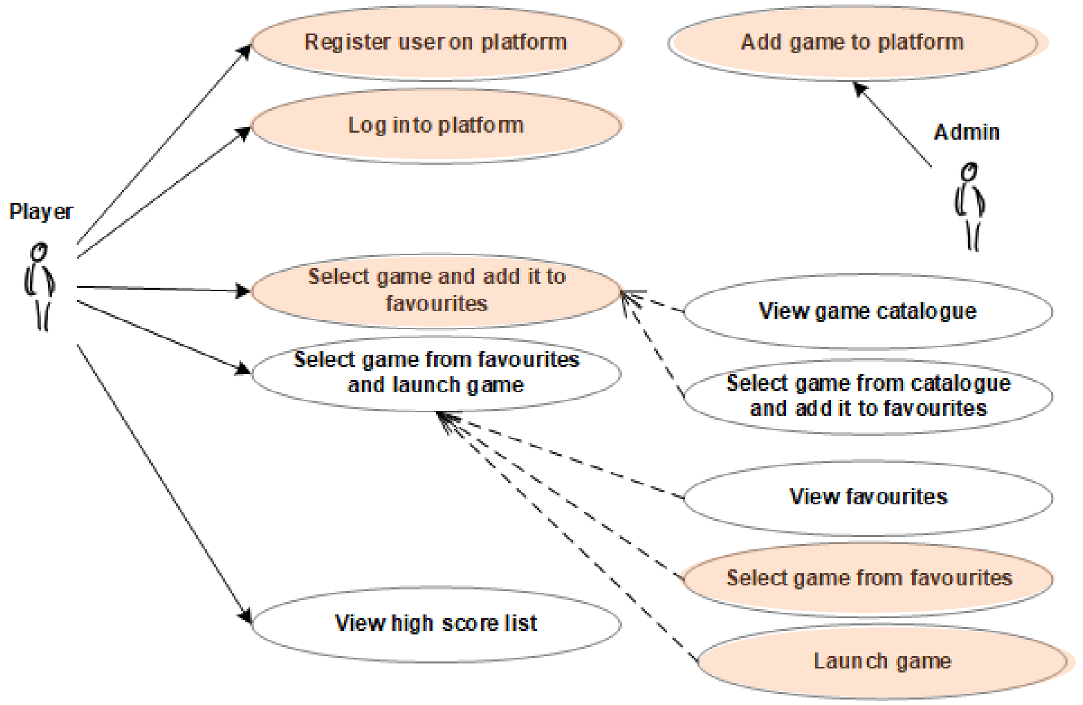
Use Case Name	Launch game
Actor	Player
Flow of Events	1. Actor launches a selected game. 2. Actor plays the game till it ends. 3. Game results are added to the game's high score list.
Precondition	Player is successfully logged into platform. A game was previously selected (e.g. From the favourites list).
Postcondition	Game results are added to high score list.
Qualities	

elaborate

System Analysis
Scenarios → Logical View

Next, **develop an Analysis Model** in the **Logical view** (i.e., *use case realizations*) **based** on these **6 Scenarios** (i.e., *use cases*).

You will use the **Miro-Board** i.1-BSc_SWA_GamePlatform-LogicalView to **elaborate your results**.


Register user on platform
Add game to platform
Log into platform
Admin
Player
Select game and add it to favourites
View game catalogue
Select game from favourites and launch game
Select game from catalogue and add it to favourites
View favourites
Select game from favourites
View high score list
Launch game

elaborate

System Analysis
Logical View → Development View

Next, **develop an Analysis Model** in the **Development view** (i.e., *classes, class diagram*) **based** on the **Logical View** (i.e., the 6 Use Case Realizations from the previous exercise).

You will use the **Miro-Board** i.1-BSc_SWA_GamePlatform-DevelopmentView to **elaborate your results**.

elaborate

System Design

Development View (Analysis Model → System Design)

In a next step, **develop a System Design Model** in the **Development view based** on the **Analysis Model** (i.e., further enrich the Analysis Model with technical details).

You will use the **Miro-Board** i.1-BSc_SWA_GamePlatform-DevelopmentView to **add your results** to the **existing Analysis Model**.

elaborate

System Implementation
Code Model in Java

Finally, you build on the architecture design and implement an initial version of the gaming platform in Java.

When you have finished your implementation, realize the following scenario in a driver class (call the class *Client*).

1. The following three users register with the gaming platform: *Jonas Arnold*, *Luis Arnold*, *Ingo Arnold*
2. All users log into the gaming platform
3. Instantiate three *games* – a *chess* game, a *tic tac toe* game, and a *connect four* game.
4. Add three games to the *gaming platform* by *Luis*
5. Display the gaming catalogue.
6. User *Ingo* selects the games *tic tac toe* and *chess* by which they are registered with his *favourite list*.
7. *Ingo* displays his *favourite list*.
8. *Ingo* selects the *tic tac toe* game from his *favourite list* and launches it. *Remember: a launched game is simulated and the game result is added to the game's high score list automatically after ending the game.*

elaborate

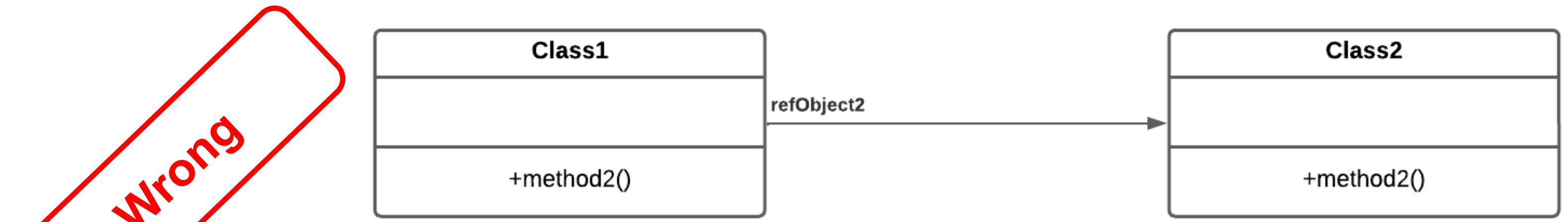
System Implementation
Code Model in Java

When you have finished your implementation, realize the following scenario in a driver class (call the class *Client*).

9. *Ingo* selects the *chess* game from his *favourite list* and launches it.
10. Next, user *Jonas* selects the game *connect four* by which it is registered with his *favourite list*.
11. *Jonas* displays his *favourite list*.
12. *Jonas* selects *connect four* from his *favourite list* and launches it.
13. Finally, user *Luis* selects all three games (i.e., connect four, chess, tic tac toe) by which they are registered with his *favourite list*.
14. *Luis* displays his *favourite list*.
15. *Luis* selects *connect four* from his *favourite list* and launches it.
16. *Luis* selects *chess* from his *favourite list* and launches it.
17. *Luis* selects *tic tac toe* from his *favourite list* and launches it.
18. *Luis* selects *connect four* from the *game catalogue* and displays its *high score list*.
19. *Luis* selects *chess* from the *game catalogue* and displays its *high score list*.
20. *Luis* selects *tic tac toe* from the *game catalogue* and displays its *high score list*.

System Implementation
Code Model in Java

To convert the development view into a corresponding Java programme, we follow the following systematic. Classes that have a use relationship never bind directly to each other, but always indirectly via a corresponding interface.



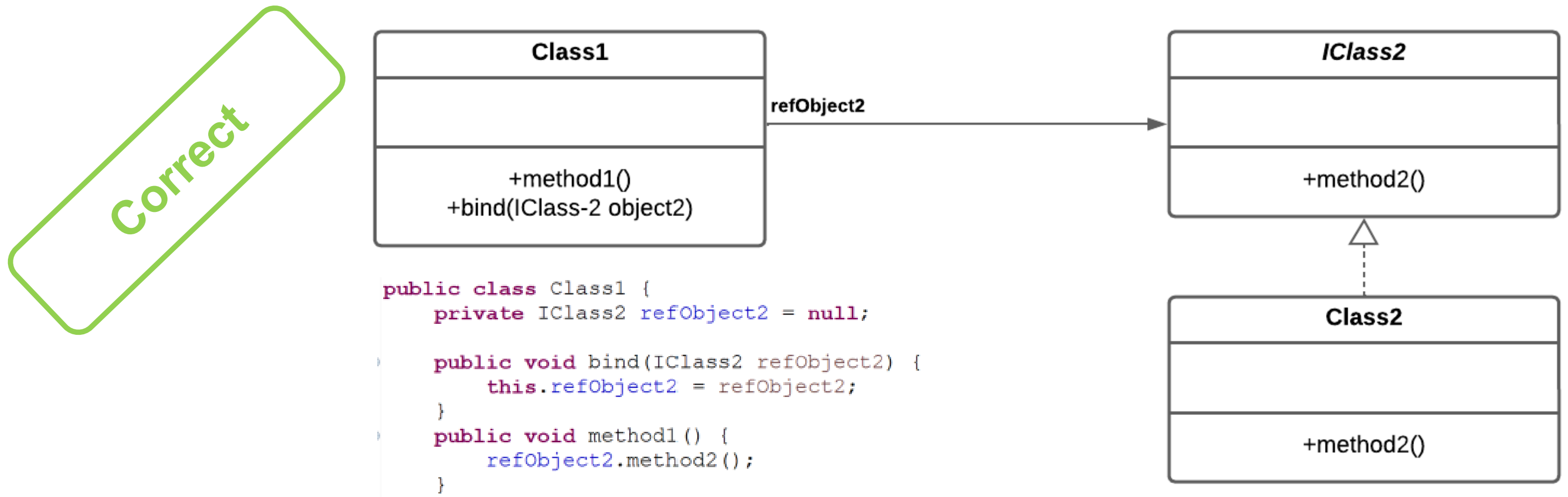
```
public class Class1 {
    Class2 refObject2 = new Class2();

    public void method1() {
        refObject2.method2();
    }
}
```

```
public class Class2 {
    public void method2() {}
}
```

System Implementation

Code Model in Java

To convert the development view into a corresponding Java programme, we follow the following systematic. Classes that have a use relationship never bind directly to each other, but always indirectly via a corresponding interface.



```
public interface IClass2 {
    public void method2();
}
```

```
public class Class1 {
    private IClass2 refObject2 = null;

    public void bind(IClass2 refObject2) {
        this.refObject2 = refObject2;
    }
    public void method1() {
        refObject2.method2();
    }
}
```

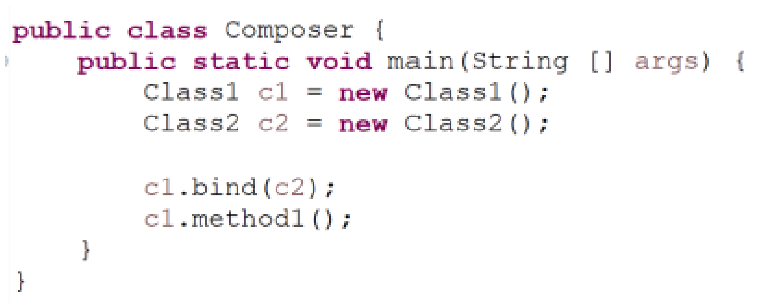
```
public class Class2 implements IClass2 {
    public void method2() {}
}
```

System Implementation
Code Model in Java

To convert the development view into a corresponding Java programme, we follow the following systematic. Classes that have a use relationship never bind directly to each other, but always indirectly via a corresponding interface.

Correct

To execute the entire programme, we will later create corresponding objects outside each implementation class (in a dedicate Composer class) and connect all objects according to our design.



```
public class Composer {
    public static void main(String [] args) {
        Class1 c1 = new Class1();
        Class2 c2 = new Class2();

        c1.bind(c2);
        c1.method1();
    }
}
```

Questions