1. Creator:

createAndShowGUI is a creator that will create a homepage interface.

2. Information Expert:

The responsibility for HomePage class is to display a homepage interface. It has all the informations needed inside its class and has little interaction with other classes. Therefore the information should be stored within this class.

3. Low Coupling:

Low coupling has the benefits that lower dependency between the classes, change in one class having a lower impact on other classes, higher reuse potential. This pattern implies that we should minimize the connections between classes.

In our design model, we use low coupling between HomePage and other classes. That's because a HomePage serves only as an interface to connect to other actions. Therefore it can interact with other classes without concerning other classes' internal implementation. This way, change to other classes does not significantly impact HomePage class, and HomePage class is highly reusable.

4.Controller:

createAndShowGUI is a controller that show a homepage interface to get user actions. It assigns tasks to other objects based on the user's actions.

5. High cohesion:

High cohesion is an evaluative pattern that attempts to keep objects appropriately focused, manageable and understandable. This implies that each of our classes should focus on related responsibilities rather than distinct ones.

HomePage class has no other responsibility than to present the homepage interface. This enables internal changes to the HomePage class to not affect the rest of the classes.

6. Indirection:

This pattern is prefer to assign responsibilities to a mediation object, isolating the object from other artifacts or services so that they do not have direct coupling. So the indirection pattern’s benefit always with low coupling.

For example, The LoginChoose class in our project is the “mediation object” between LoginFormCustomer class and LoginFormEmployee class, which helps the isolating of those two classes.

7. Polymorphism:
We don’t have Polymorphism on our project at this point. But later, we will use polymorphism after we finish the customer payment method. Like in the lecture, payment method contains many if-else loop like cashPayment(), cardPayment() and checkPayment(). So we can use an interface with a payment() and customers can implements from this interface to have their own payment() method.

8. Protected Variations:

This pattern is prefer to provides flexibility and protection from variations.

Firstly we need to find the unstable change points in the system and encapsulate them with a unified interface. If changes occur in the future, new functions can be extended by extending the interface without modifying the old code. This achieves the purpose of easy expansion.   For example, providing a well defined interface so that the there will be no affect on other units.

9. Pure Fabrication:

Example:
The system operates on the database. If we want to persist some objects in the system. If we use information expert pattern, it will assign this responsibility to each class. However, it will violent the high cohesion and low coupling. So we will make a new class such as DAO. And assign this responsibility to the DAO class.