1. Creator:
AddCountroller is a creator that will create a record and store the record to the database.

2. Information Expert:
We don’t have this patten in order to avoid the problem of cohesion and coupling.
Example:
If we are shopping, one product only appears once in the shopping cart. So when we add product to the shopping cart we need to know if this produce is in the cart. And this responsibility should go to produce, because the produce has their unique id number.

3. Low Coupling:
This pattern is prefer to assign responsibilities and then reduce the impact change in depended upon elements on dependent elements. That means we should minimize the connection between classes.
In our class diagram, we use Low Coupling between amountController class and addController class. Because the amountController is to input value only so the only related class is addController.

4.Controller:
We have the controllers:
amountController:
   show a page like a calculator to read the input.
addController:
   show a page like a form to read the imformation of a record.
OptionController:
   There is 2 buttom in this page. One is add, another is view
CalendarCountroller:
   This page performs a calendar to set a filter.
viewController:
   This page shows the table of AccountingTable.

5. High cohesion:
This pattern is prefer to place the functionally closely related responsibilities in a class and let those responsibilities work together to accomplish limited functionality. High cohesion tries its best to guarantees that programs occur within each section will not affect other sections. Record class uses this pattern in our class diagram, for example, Record class has both of the setDate

and getDate, so if there are problems occur in date related functionality, because of the high cohesion, it will not influence other responsibilities.

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 OptionController class in our project is the "mediation object" between addController class and CalendarController class, which helps the isolating of those two classes.

7. Polymorphism:

We don't have Polymorphism on our project.

Example: If we want to calculate the tax of a produce, We will have a method call calculateTax(). However, different country has different tax. So in the calculateTax method, we need to use many if-else to calculate tax for different country. So we can use an interface with a calculateTax() and many country can implements from this interface to have their own calculateTax() method.

8. Protected Variations:

This pattern is prefer to provides flexibility and protection from variations. Pre-identify unstable change points, encapsulate with a unified interface, and if the future changes, new functionality can be extended through the interface without having to modify the old implementation. 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.