Exercise 1:

Apply appropriate design pattern to design an HR management application that manages employees' information. Every employee has a title. The title may be manager, developer… Each employee may have different type of information need to be stored, depends on his/her title.

Exercise 2:

Suppose you plan to manage address and telephone information as part of a personal information manager (PIM) application. The PIM will act as a combination address book, personal planner, and appointment and contact manager, and will use the address and phone number data extensively.

You can initially produce classes to represent your address and telephone number data. Code these classes so that they store the relevant information and enforce business rules about their format. For example, all phone numbers in North America are limited to ten digits and the postal code must be in a particular format.

Shortly after coding your classes, you realize that you have to manage address and phone information for another country, such as the Netherlands. The Netherlands has different rules governing what constitutes a valid phone number and address, so you modify your logic in the Address and PhoneNumber classes to take the new country into account.

Now, as your personal network expands, you need to manage information from another foreign country... and another... and another. With each additional set of business rules, the base Address and PhoneNumber classes become even more bloated with code and even more difficult to manage. What's more, this code is brittle—with every new country added, you need to modify and recompile the classes to manage contact information.

What design pattern can be used to solve the problem?

Exercise 3:

One of the functionalities of your existing project manager application is to manage project's meetings. The problem is that whenever there is a meeting between specific members a notification message is sent to all the members of the project regardless of their interesting. Please use appropriate design pattern to solve the problem.

Exercise 4:

Enhance HR management application to support undo/redo function.

Exercise 5:

You’ve decided to enhance the Project Management application to let users manage a complex project. Features of this enhancement include defining the project as a group of tasks and related subtasks, and associating deliverables with tasks. A natural way to accomplish this from a programming perspective is to define a tree structure, where the root task (which represents the project itself) branches out to subprojects, subsubprojects, and so on. Therefore, you define a Task class that holds a collection of other Task and Deliverable objects. Since both the Task and Deliverable relate to the project, you define a common parent for them—the ProjectItem class.

However, what happens if users need to perform an action that depends on the whole tree? For example, a project manager wants a time estimate for tasks and deliverables for one project. To accommodate this, you write code to traverse the tree and call the appropriate methods at each branch. That's a lot of work, however, involving separate code to walk the tree, call the methods, and collect the results. And with different classes (Task and Deliverable) at each branch of the tree, you might need to handle them differently when getting time estimates.

For a large number of classes or a complex tree, the code quickly becomes difficult to manage.

Please use appropriate design pattern to solve the problem.

Project

Design TMAPaint application (see the picture) with these functionalities:

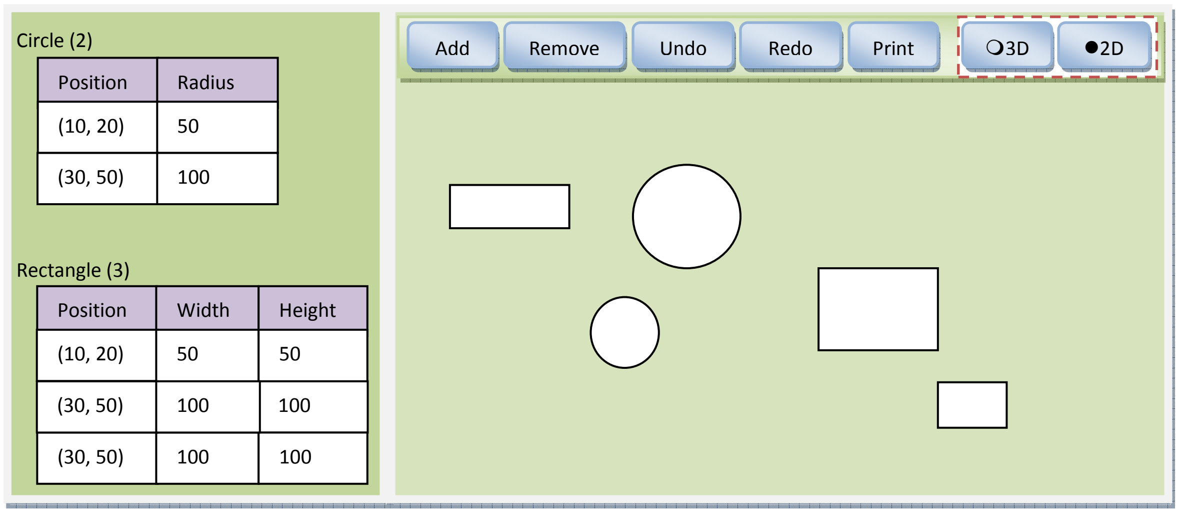
1. Supports 2 look-and-feels (2D, 3D)
2. Allows user to add, remove shape objects. Shape object can be either Rectangle or Circle
3. Has a panel that list out current objects added. This panel will be updated once an object is added/removed. User can select the objects in this panel to change its parameters. The change will be immediately reflected on the main panel.
4. Supports undo/redo
5. Supports printing function which implements this predefined interface:

```
interface Printer() {
    // start up the physical device
    // load paper
    // ...

    // print the object
    // printedObject.print();

    // stop the physical device
    public void print(PrintObject printedObject);
}
```

```
class PrintObject{
    public void print(Graphics g) {…}
}
```



Circle (2)

Position	Radius
(10, 20)	50
(30, 50)	100

Rectangle (3)

Position	Width	Height
(10, 20)	50	50
(30, 50)	100	100
(30, 50)	100	100