Facade Design Pattern

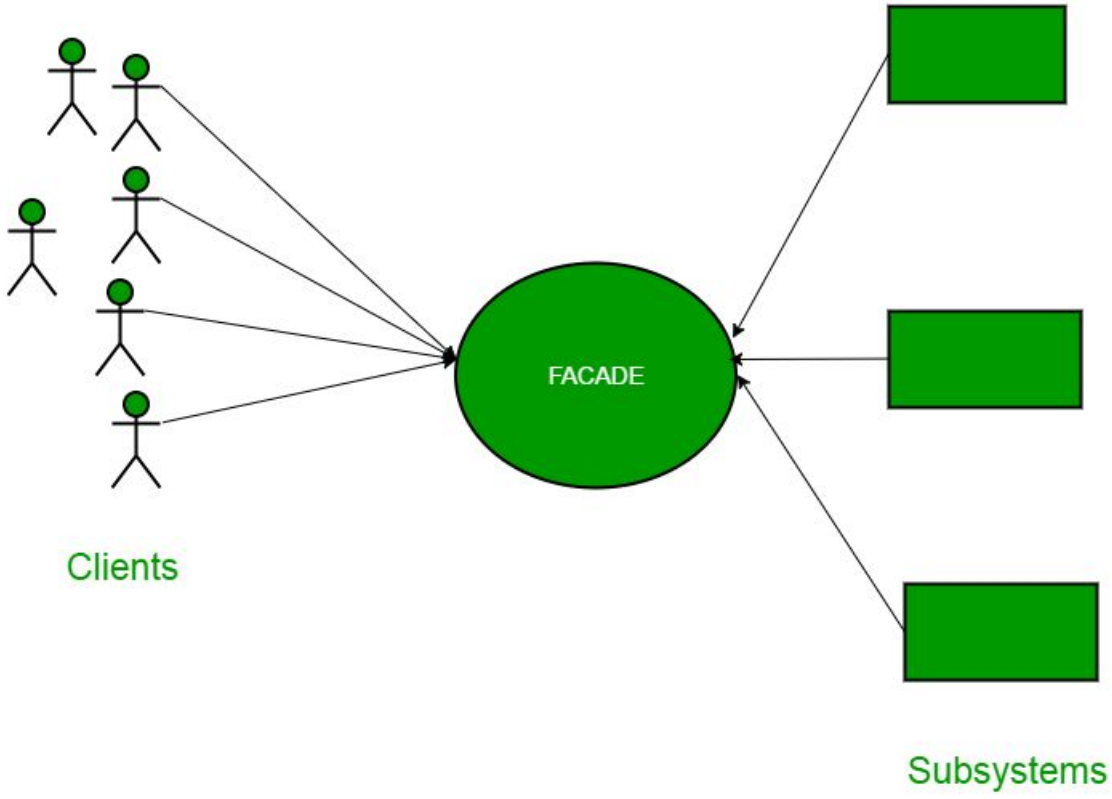
Facade is a part of Gang of Four design pattern and it is categorized under Structural design patterns. Before we dig into the details of it, let us discuss some examples which will be solved by this particular Pattern.

So, As the name suggests, it means the face of the building. The people walking past the road can only see this glass face of the building. They do not know anything about it, the wiring, the pipes and other complexities. It hides all the complexities of the building and displays a friendly face.

In Java, the interface JDBC can be called a facade because, we as users or clients create connection using the “java.sql.Connection” interface, the implementation of which we are not concerned about. The implementation is left to the vendor of driver.

Another good example can be the startup of a computer. When a computer starts up, it involves the work of cpu, memory, hard drive, etc. To make it easy to use for users, we can add a facade which wrap the complexity of the task, and provide one simple interface instead.

Same goes for the **Facade Design Pattern**. It hides the complexities of the system and provides an interface to the client from where the client can access the system.



Now Let's try and understand the facade pattern better using a simple example. Let's consider a hotel. This hotel has a hotel keeper. There are a lot of restaurants inside hotel e.g. Veg restaurants, Non-Veg restaurants and Veg/Non Both restaurants. You, as client want access to different menus of different restaurants . You do not know what are the different menus they have. You just have access to hotel keeper who knows his hotel well. Whichever menu you want, you tell the hotel keeper and he takes it out of from the respective restaurants and hands it over to you. Here, the hotel keeper acts as the **facade**, as he hides the complexities of the system hotel.

Let's see how it works :

Interface of Hotel

```
package structural.facade;
public interface Hotel
{
        public Menus getMenus();
}
```

The hotel interface only returns Menus.

Similarly, the Restaurant are of three types and can implement the hotel interface. Let's have a look at the code for one of the Restaurants.

```
package structural.facade;

public class NonVegRestaurant implements Hotel
{
	public Menus getMenus()
	{
		NonVegMenu nv = new NonVegMenu();
		return nv;
	}
}
```

```
package structural.facade;

public class VegRestaurant implements Hotel
{
	public Menus getMenus()
	{
		VegMenu v = new VegMenu();
		return v;
	}
}
```

```
package structural.facade;

public class VegNonBothRestaurant implements Hotel
{
	public Menus getMenus()
	{
		Both b = new Both();
		return b;
	}
}
```

Now let's consider the facade,

```
package structural.facade;

public class HotelKeeper
{
```

```
	public VegMenu getVegMenu()
	{
		VegRestaurant v = new VegRestaurant();
		VegMenu vegMenu = (VegMenu)v.getMenus();
		return vegMenu;
	}

	public NonVegMenu getNonVegMenu()
	{
		NonVegRestaurant v = new NonVegRestaurant();
		NonVegMenu NonvegMenu = (NonVegMenu)v.getMenus();
		return NonvegMenu;
	}

	public Both getVegNonMenu()
	{
		VegNonBothRestaurant v = new VegNonBothRestaurant();
		Both bothMenu = (Both)v.getMenus();
		return bothMenu;
	}
}
```

From this, It is clear that the complex implementation will be done by HotelKeeper himself. The client will just access the HotelKeeper and ask for either Veg, NonVeg or VegNon Both Restaurant menu.

How will the client program access this façade?

```
package structural.facade;

public class Client
{
	public static void main (String[] args)
	{
		HotelKeeper keeper = new HotelKeeper();

		VegMenu v = keeper.getVegMenu();
		NonVegMenu nv = keeper.getNonVegMenu();
		Both = keeper.getVegNonMenu();

	}
}
```

In this way the implementation is sent to the façade. The client is given just one interface and can access only that. This hides all the complexities.

When Should this pattern be used?

The facade pattern is appropriate when you have a **complex system** that you want to expose to clients in a simplified way, or you want to make an external communication layer over an existing system which is incompatible with the system. Facade deals with interfaces, not implementation. Its purpose is to hide internal complexity behind a single interface that appears simple on the outside.