Facade Design Pattern:

**Other name (if any)**

placeholder

**What it does**

Facade hides the complexity from the client. It takes different methods from different classes to make a subsystem. It creates a sub system. Facade can have different facade. Its purpose is to hide internal complexity behind a single interface that appears simple on the outside.

**Where to use**

1. When you want to provide simple interface to a complex sub-system.  
2. A facade can combine multiple APIs into a single interface, streamlining interactions and reducing code duplication.  
3. Facades minimize the impact of changes to underlying systems by maintaining a stable interface.  
4. Facades define clear boundaries between subsystems, reducing dependencies and promoting modularity.  
5. If a class has huge number of functions but only few functions are important to the client. With these few functions facade pattern can be created.

**Steps**

1. Facade Class: Create object from different classes which are necessary for the sub system. Call the important functions of the objects which are needed for that sub system.  
2. Client Code: Create an object using facade class and call the function of that class.

**Special cases (if any)**

placeholder

**Advantages**

1. It shields the clients from the complexities of the sub-system components.  
2. Allows for refactoring and optimization of the subsystem without impacting client code.  
3. Promotes better code readability and maintainability.  
4. Facilitates independent development and testing of different parts of the system.

**Disadvantages**

placeholder

**Code**

Coding Concept

**Difference with similar pattern**

Proxy Design Pattern: Proxy handles only one similar type of object.  
Adapter Design Pattern: Adapter makes adaptable.

**Diagram**

Tutorialspoint