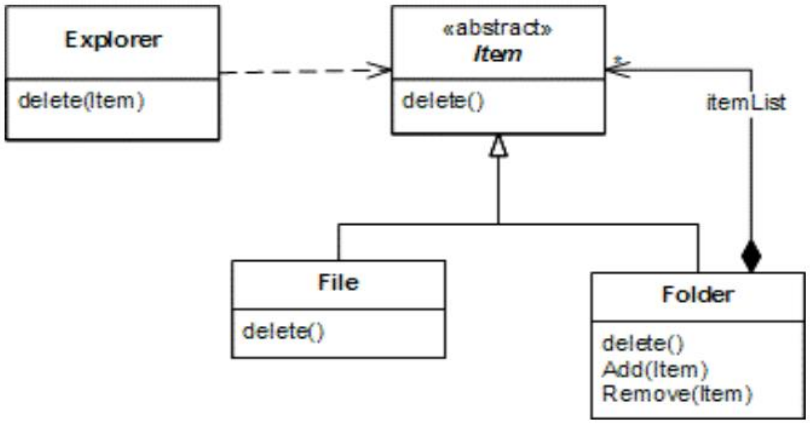
Identify the appropriate structural design pattern and implement it using Java.

1.



2. Create a simple serialization application which allows you to serialize objects into JSON or XML format. For example, create PersonInfo class which will be serialized. For serialization use JavaScriptSerializer and XmlSerializer classes. These classes play in this example role as Adaptees with incompatible interfaces. ISerializerAdapter (ITarget) is interface which must be implemented by concrete adapter. It has one method called Serialize which serializes object into appropriate format. JSONSerializerAdapter and XMLSerializerA dapter classes implement ISerializerAdapter interface. These two classes are adapters.

3. Create a simple messaging application. For this purpose Message class was created. This class acts as Abstraction for UserEditedMessage class. This class has one protected filed of type MessageSenderBase. MessageSender base is implementor and abstract class from all concrete implementations of message senders. Three message senders were created: EmailSender, MsmqSender and WebServiceSender. This is only demonstrative example and hence no realistic functionality of these senders is implemented.

4. In this example prepare your custom favourite sandwich. In this case assume Sandwich class which is an component base class. This class has two public abstract methods. GetDescription method returns full name of sandwich with all ingredients. Second method called GetPrice returns price of current sandwich. Create two sandwiches: TunaSandwich and VeggieSandwich. Both of these classes inherit from Sandwich class. SandwichDecorator class is a base class for all decorators. This class inherits from Sandwich class too. Three decorators were prepared: Olives, Cheese, Corn. This decorators override GetDescription and GetPrice methods of the Sandwich class. The first step is to create a concrete sandwich. You can choose from two

options: VeggieSandwich and TunaSandwich. If it is done, you can add some ingredients by creating of appropriate decorator and wrapping your sandwich by this decorator. If you want to add another ingredient, just create a new instance of decorator and wrap sandwich by this new decorator. By calling of method GetDescription you will get name of sandwich with all ingredients.

5. In a war game example UnitFactory can create military units. Create two types of units: Soldier and Tank. These classes are concrete flyweights which inherits from Unit flyweight base class. UnitFactory class contains method called GetUnit which accept one parameter that specify type of unit to create. This class has one private dictionary of unit types. When new unit is required, the first step is to look into this dictionary whether this type of unit was created earlier. If yes, program returns reference to this unit. If no, it just creates a new unit and places it into dictionary. You can control the number of instances of each unit type by static field NumberOfInstances.

6. Create Order application which displays information of selected order. Create the OrderRepositoryBase class which is an abstract class for ProxyOrderRepository a nd RealOrderRepository classes. RealOrderRepository class is a RealSubject which we want to consume by our application using ProxyOrderRepository. Also have a set of entities which represents data of RealOrderRepository class. In this example Order can have a multiple OrderDetails but one Customer. These properties could be filled by methods of RealObjectReporsitor (GetOrderDetails and GetOrderCustomer). That means if we want to get all of the information about the order, we must call three different methods of RealOrderRepository class (GetOrder, GetOrderDetails and GetOrderCustomer). ProxyOrderRepository has all of these methods too. In case of GetOrderDetails and GetOrderCustomer it calls directly methods of RealObjectRepository class. In case of GetOrder the situation is a little bit different. It doesn't only call method of RealObjectRepository but it sets OrderDetails and Customer properties of Order returned by GetOrder method of RealObjectRepository. When all if the properties of Order object are populated, Order is returned to the caller application.