Seven Design Patterns

Design Pattern: Singleton

Where the Design Pattern Is Used: Service Manager

Explanation: The singleton pattern is when only one instance of a class is created during the running of the program. This is achieved through a private class constructor, a static reference to the single instance, which is created when the class initializes, and a static method which accesses the single instance. The NETime Planner application is using the singleton pattern for the ServiceManager class because the functionality of this class needs to be accessed globally by other classes and because only one copy of this manager object needs to exist. The benefit of using the singleton pattern over making each managerial function a static method is that references to the single instance can be inputted into other methods, allowing methods to accept the ServiceManager object as a parameter.

Design Pattern: Visitor

Where the Design Pattern Is Used: Events

Explanation: The visitor pattern places the code for a certain operation outside of an object's class. A visitor object performs operations on certain objects which have accept methods which accept that visitor as a parameter. This method then calls the visitor's visit method, and this method accepts the reference to the object that is being visited. The result of this is that any objects that need a certain operation can simply inherit the interface that corresponds to that operation. Events in the NETime Planner application use an abstract EventVisitor class which can be extended for the creation of different operations which behave differently depending on whether they are accepting an Event, Activity, or Deadline object, where Activity and Deadline objects are Event subclasses.

Design Pattern: Prototype

Where the Design Pattern Is Used: Copying Events

Explanation: The prototype pattern involves the use of a clone method, where an instance of the object containing the method acts as a prototype, while the calling of the clone method creates a new instance of the object based on that prototype. Events in the NETime Planner application contain a clone method, which allows Event objects to be more easily created.

Design Pattern: Observer

Where the Design Pattern Is Used: Listening to Schedule Changes

Explanation: In this pattern, there are observer and subject objects. Observer objects contain the method, update, and a reference to a subject object. Subject objects contain references to their observer objects and can notify these observers by calling the update method. This pattern allows observers to be updated based on what occurs in the methods of the subject object. The NETime Planner application uses the observer pattern to execute code whenever changes in the schedule occurs. This allows information about the schedule which the program tracks to be dynamically updated as needed.

Design Pattern: Flyweight

Where the Design Pattern Is Used: Cache for DbDAO

Explanation: The flyweight design pattern reduces the number of created objects by allowing objects to be reused. This is achieved by causing an object to only store its intrinsic state, while allowing the object's extrinsic state to exist outside of the object. The intrinsic state is the information about an object which is not modified outside of the object. When an object is created, it has a key which corresponds to its intrinsic state. Rather than creating two objects with the same intrinsic state, a factory that creates an object refers to the existing object with that state. In the NETime Planner application, the DbDAO object has a cache which stores data access objects with certain intrinsic states, allowing for more efficient object creation.

Design Pattern: Adapter

Where the Design Pattern Is Used: Converting a DayOfWeek Object to an Integer and Back

Explanation: The adapter design pattern involves a class which wraps around a different object to provide the ability to use the object that is being wrapped but with a different interface to that of the wrapped object. The methods used for accessing the functionality of the wrapped object call the methods on that wrapped object and return the results. The NETime Planner application uses this pattern when converting a DayOFWeek object to an integer and back. A DayOfWeekAdapter object contains a method for converting a set of DayOfWeek objects to an integer as well as a method for converting an integer to a set of DayOfWeek objects.

Design Pattern: Command

Where the Design Pattern Is Used: Opening Someone's Schedule With a Button

Explanation: The command pattern involves command objects which have an execute method and the data required for executing the method. Receivers are the objects which will perform the operation after the execute method is called, and invokers are responsible for storing command objects and calling their execute methods. The NETime Planner application uses the command pattern for the operation of opening someone's schedule with a button, which allows for decoupling the object that is responsible for receiving the button input with the object that is responsible for executing the commands caused by that input.