Design Patterns

1. Factory design pattern
   a. ViewFactory: using enums we can construct a View with one controller and presenter. The ViewFactory is created in the AccountController, and then returns the next view of the program based on information about the previous view(s) and the state of the program. The View Factory allows us to create views and return them without specifying which kind of view they are.
   b. EventFactory: using enums we can generate different Events with no worry about type-specific constructors. The explanation for this is similar to the explanation for ViewFactory.
2. Visitor design pattern
   a. VipAttendee, Attendee: the interfaces VipAcceptor, VipVisitor and helper class VipHelper (which implements VipVisitor) implement the visitor design pattern and allow us to distinguish if an Attendee is VIP or not without typecasting. The two Attendee classes implement VipAcceptor and have their own implementations of determining their VIP status (ie. VipAttendee is a VIP and other Attendees are not) and the AccountManager method isVipAttendee() will call those methods to obtain this info without the usage of instanceof keyword and typecasting.
   b. Event (and its subclasses): the interfaces EventAcceptor, EventVisitor and helper class EventHelper (which implements EventVisitor) help distinguish Events by their types and obtain speakers of specific Events without typecasting. Each Event subclass implements the methods outlined by the EventAcceptor interface, returning either a string representation of the subclass type (for use in downloading the schedules as HTML), or a list of zero, one, or more speaker usernames of the Event (for use in viewing Events by Speaker).
3. Null object design pattern
   a. ViewEnum contains “VOID” constant and MenuEnums contain “INVALID” constants. These constants enables a default “do nothing” or “try again” behaviour in the collaboration between user-issue commands and the programs response.
   b. For example
      i. Several Views return ViewEnum.VOID which provides a default behavior for the program to complete a command and non-recursively return back to the user menu
      ii. User menus requests command input and if the user’s input does not match any functional command, returns MenuEnum.INVALID by default
   c. This way, the collaborators that use ViewEnums and MenuEnums can treat “Do nothing” or “Try again” commands as commands that actually tell them to do something.

4. Command design pattern
   a. View interface acts as a base ‘Command’ class containing an execute method, runView()
   b. ViewFactory maps implementations of View to client-issued actions represented by ViewEnums and MenuEnums, afterwards the Command execution is run
   c. Each view is encapsulated and does not know about any other view; each view only returns a VIEW.Enum after running so that the recipient of this return statement can decide which view is to be run next.
   d. The classes who are responsible for switching between views based on their output are the AccountView and AccountController. The getViewEnum method calls getView from AccountController who then calls the viewFactory to fetch the appropriate view based on the viewNum returned from the previous view.