Instructions

Below are four short descriptions of problems in specific contexts. Two solutions are required for each problem - the second an elaboration on the first. Each solution is to be structured as follows:

i. Briefly outline of the relevant design pattern(s) that are viable candidates in the ensuing answer.

(5 Marks)

ii. A solution to Version 1 expressed as a class diagram and with a high level sketch of the classes involved. Key methods in the class can be expressed in any suitable pseudocode.

(10 Marks)

iii. A solution to Version 2, which may be expressed as modifications to Version 1 with further classes or pseudocode if necessary.

(10 Marks)

iv. A short summary of the benefits of adopting the selected patterns.

(8 Marks)

Select *any three* of the problems and propose the outline solutions structured as described above. All problems are awarded equal marks.

Problem 1: Blogging Component

A new component is to be designed and integrated into an existing Content Management System. This component is to provide blogging functionality. In particular, the component is to provide a clean API for managing articles written by contributors - called Posts. Initially just plain text, posts can be created, deleted, edited, previewed, approved and published. There may be additional manipulations of posts proposed later in the development effort.

- Version 1: Propose a flexible API to encapsulate this aspect of the component. The API can be expressed as a set of Interfaces, which can be implemented incrementally as the project advances.

- Version 2: Selected features, initially edit, approve and publish should have the ability to be rolled back easily. Proposed extension to the API to flexibly support this capability.

Problem 2: Mobile "Push" capability

A mobile phone platform is to be augmented to provide "Push" functionality for registered applications. This will enable individual applications to register with a central component their interest in receiving specific messages. Messages are structured into topic/message body format - all text. Once messages arrive then they will be "pushed" to the relevant registered application. The phone platform is based on JDK 6, and will have the full set of java libraries available from that release.

- Version 1: Propose a design based on the java.util.Observer/Observable capability. This version should also provide a simple logging component.

- Version 2 : A more sophisticated implementation is to be proposed based on then PropertyChangeSupport libraries in java.beans library. This version should may be used by a Swing application and should elegantly address the single threaded nature that library.

Problem 3: Media Player

Two development teams are developing components for a new media player. A Media team is focusing on the core media library and player capabilities, and a UI team is building the user interface. The media team must evolve a model for the media library, including tracks, albums, playlists, cover art. In addition it must provide for different media categories including movie, tv programme and tv series. The media team would like to present to the UI team a stable model, but have the freedom to evolve alternative implementations of this model as the development effort proceeds and is ported to different platforms.

- Version 1: The mediator pattern is proposed for the first phase to allow the teams to start work. Outline a simple design that would use this pattern.

- Version 2: Once a mature understanding has been reached, the Media team propose a design based on the Bridge pattern. Show what an early draft of this might look like.

Problem 4: Components A, B, C & D.

A developer is present with the following problem. Class A has been written to operate with Class B, with a reference to B provided to A's constructor. The source for Class A is "closed", i.e. it cannot be modified. However, the source for class B is available. Note also that there are classes C and D in the problem space, which are unrelated to B.

- Version 1: Propose a design pattern that would enable class A to interoperate with class C or D. Use implementation inheritance in the proposal.

- Version 2: Propose an alternative configuration whereby B is in fact specified as a Java interface, not a class. What variation of the design pattern proposed in Version 1 can now be adopted. Explore the advantages of this approach.