MVC: Model View Controller

Aufgabe 1)

|  |  |  |
| --- | --- | --- |
| **Beschreibung** | **richtig** | **falsch** |
| Die View verwaltet die Geschäftslogik. |  | X |
| Bei der Pull-Methode holt sich die View vom Model per Getter die aktuellen Daten. | X |  |
| Wenn eine Änderung am Model durchgeführt werden, schickt der Controller Daten an die View. |  | X |
| Controller sind das Bindeglied zwischen View und Model. | X |  |
| Das Model beinhaltet die Logik der Daten und deren interne Strukturen . | X |  |
| Die Push-Methode ist immer die bessere Variante wenn das Subjekt alle Details der Observer kennt. | X |  |
| Es gibt eine eindeutige, allgemeingültige Definition von MVC. |  | X |

MVP

Aufgabe 1)

Lösungsvorschlag:

|  |  |  |
| --- | --- | --- |
| **Beschreibung** | **richtig** | **falsch** |
| Model-View-Presenter ist eine Weiterentwicklung von MVC. | x |  |
| MVP ist nicht für moderne Programmiersprachen ausgelegt. |  | x |
| Bei der Festlegung eines Designs nach dem MVP-Musters sollte zuerst der Presenter implementiert werden. | x |  |
| Es ist nicht sinnvoll für den Presenter eine Schnittstelle (Interface) zu spezifizieren. |  | x |
| Eine View in einer MVP-Anwendung kennt alle anderen Komponenten, wie z. B. das Model. |  | x |
| Die MVP-Anwendung ist nach der Implementierung nicht mehr erweiterbar. |  | x |
| Bei jedem Anforderungsprofil für eine Software ist das MVP-Pattern geeignet. |  | x |
| Eine passive View verwaltet selbstständig Ihre GUI-Handler (bezüglich der Anwendungslogik). |  | x |
| Ein Presenter ist eine Klasse, die als Bindeglied zwischen Benutzeroberfläche und der Datenhaltung verstanden werden kann | x |  |
| Das MVP-Pattern ist nur eine mögliche Lösung zur Implementierung einer komponentenbasierenden Softwarelösung | x |  |
| Die Realisierung einer Server-/Clientanwendung ist mit MVP nicht möglich |  | x |
| Die Implementierung einer MultiThreading-Umgebung ist in MVP nicht möglich |  | X |

Aufgabe 2)

Lösungsvorschlag:

