Software Engineering Übung 10:

Gruppe:

1. Michael Scholz (matr.: 1576630, rbg: mi48azih)
2. Ulf Gebhardt (matr.: 1574373, rbg: hu56nifa)
3. Sebastian Weicker (matr.: 1625099, rbg: we87obyq)

Aufgabe 1: Verwendung von Design Patterns in JDK:

a)

Die Klasse ist das Interface für die SocketFactory des RMIClient Frameworks. Es wird das Factory Pattern implementiert. Dieses Interface ist die Vorlage für den Erzeuger.

b)

Es handelt sich um einen robusten Iterator. Dieser prüft, ob sich die Daten des Aggregats geändert haben. Ist dies der Fall, so wird eine "ConcurrentModificationException()" durch die Funktion "checkForComodification()" geworfen. Somit ist der robuste Iterator gegen Modifikationen des Aggregats gesichert.

c)

Es ist das Singletone Pattern zu erkennen. Es kann immer nur eine Runtime-Instanz pro Java Aplikation existieren. Die aktuelle Instanz kann mit getRuntime() abgefragt werden. Zudem ist das Factory Pattern zu erkennen. Die Methoden exec(...) erstellen jeweils einen Prozess, welcher in der Runtime-Instanz ausgeführt wird.

Aufgabe 2: Design Patterns diskutieren:

a)

Wenn die zu testende Klasse ein Singleton ist oder verwendet, so existiert nur eine Instanz. Zum Testen werden jedoch meistens mehrere Instanzen der Klasse erzeugt, um einzelne Verhaltensweisen vom Defaultzustand zu testen. Dies ist jedoch nicht möglich, da Singletons nur eine Instanz zulassen. Beispiel: Java Junit @Before hat in diesem Fall keine Funktion mehr. Es kann nur noch @BeforeClass benutzt werden.

b)

Wir nutzen das Abstract Factory Pattern wie folgt: Die Klassen WindowManager, Window und Scrollbar sind Interfaces, welche von den jeweiligen Framework implementiert werden. Hierbei beinhalten die Frameworks Templates mit ihren eigenen Typen für Window und Scrollbar. Dadurch sind Objekte verschiedener Frameworks nicht mehr miteinander kompatibel, da sie ein anderen Templateparamter nutzen.
Die Interface WindowManager erfordert eine Methode zum erstellen von Windows. Das Interface Window erfordert wiederum eine Methode zum erstellen von Scrollbars(bzw. Für die Erweiterbarkeit eine Methode zum Erstellen von WindowObjects, siehe Teil2). Die einzelnen Frameworks müssen diese Methoden implemntieren. Dadurch wird sichergestellt, dass die Framework-Komponenten nicht untereinander kompatible sind.

Um das Framework flexibel und erweiterbar zu halten sollte man ein Interface WindowObject forden, welches eine allgemeine Schnittstelle für Objekte in einem Fenster darstellt. Scrollbar würde dann dieses Interface implemntieren und das Interface Window würde z.B. folgende Funktion erfordern: addObject(<? Extends WindowObject> O).

Aufgabe 3: Übung im Umgang mit Design Patterns:

a)
Die Verantwortlichkeit eine Lernstrategie auszuwählen und die entsprechenden Lernstrategie zu instanziieren hat die Methode learn() in der Klasse FlashCardsWindow.

b)
➡ *siehe Code*

c)

AllLearnStrategies ist der "ConcreteCreator"; FlashcardsWindow ist das "ConcreteProduct", welches den Erzeuger aufruft.

In unserem Design gibt es keine "Product" oder "Creator", von dem das "ConcreteProduct" bzw der "ConcreteCreator" erbt, da das für unser Design nicht notwendig ist.

AlllearnStrategies enthält die Funktion "CreateAllLearnStrategies", welche ein Array von LearnStrategies zurückgibt. Diese Funktion wird von learn() in FlashcardsWindow aufgerufen um die Liste der Learnstrategies zu erhalten.

Im Diagramm ist die Trennung zwischen Model und UI nun deutlich zu erkennen.