Einführung in Software-Engineering – Übung 10

Aufgabe 1

a) Das Interface „RMIClientSocketFactory“ ist Teil eines Factory Method Design Pattern. Ein Objekt wird hier nämlich durch einen Aufruf einer Methode erzeugt (createSocket()), anstatt durch einen direkten Aufruf des Konstruktors. Die zu erzeugenden Objekte sind in diesem Fall Objekte der Klassen ServerSocket und Socket (steht für ClientSocket), also sind dies die ConcreteProducts, während die Klasse Socket das Product ist. Die Klasse RMIClientSocketFactory ist in diesem Zusammenhang also der Creator. Der ConcreteCreator ist dann die Klasse, in welcher die Fabrikmethode createSocket() implementiert wird und so ConcreteProducts erzeugt.

b) Der Iterator ist robust. Er wird informiert sobald bei der zugrunde liegende Liste modCount und expectedModCount ungleich sind. Dies führt zum Werfen der Concurrent-ModificationException.

c) In der Klasse java.lang.Runtime kann das Singleton Design Pattern erkannt werden, da es in dieser Klasse nur ein runtime-Objekt gibt, welches mit der statischen Methode getRuntime() ermittelt wird.

Aufgabe 2

a) Das Singleton Design Pattern sorgt dafür, dass von einer Klasse nur ein Objekt instanziiert wird. Dies hat natürlich den Nachteil einer schlechten Testbarkeit, wenn das Objekt interagierend mit seiner Umgebung getestet werden soll. Unter Umständen können in diesen Fällen Mock-Objekte für Testzwecke erzeugt werden, welche aber auch nicht immer hilfreich sind und zudem ist das Erzeugen von diesen mit mehr Aufwand verbunden.
Zudem kann es schwierig sein, mit nur einem Testobjekt alle Methoden zu überprüfen, da man diese Methoden dann nicht unabhängig voneinander testen kann. Hat man in einer früh getesteten Methode einen Fehler, so zieht sich dieses falsche Ergebnis durch das Testen bis zum Endergebnis und dies kann zu Verwirrung führen.

b) Zur Umsetzung nutzen wir das Factory Method Pattern in Verbindung mit dem Abstract Factory Pattern. Der Window-Manager stellt auf der obersten Ebene den Creator mit den ConcreteCreators FVWM und SawFish dar. Auf der gleichen Ebene steht das Window als Product mit den ConcreteProducts FVWMWindow und SawFishWindow.
Auf der darunterliegenden Ebene finden wir jetzt Window als Creator und FVWMWindow sowie SawFishWindow als ConcreteCreators vor. Die Klasse Scrollbar stellt das Product und FVWMScrollbar und SawFishScrollbar die ConcreteProducts dar.
Alle Creators und Products sind abstrakte Konstruktoren an die konkreten Objekte vererben. Klassen, die grundlegende Funktionen bereits implementiert haben und unimplementierte Methoden und Konstruktoren an die konkreten Objekte vererben. Um die Konsistenz der Klassen sicherzustellen, akzeptiert z.B. der WindowManager nur Objekte der gleichen konkreten Produkt-Klassen. Um im Hinblick auf neue GUI-Elemente das Design zu erweitern, können in den abstrakten Klassen Instanzen dieser Elemente erstellt werden. Diese neuen Objekte sind dann global in allen WindowManagern zu implementieren.

Aufgabe 3

a) Die GUI-Implementation des Fensters, durch welches der User letztlich aus verschiedenen Strategien die gewollte Lernstrategie auswählt, ist in der Methode learn() der Klasse FlashcardsWindow programmiert worden. Für jede Lernstrategie existiert eine eigene Klasse, in welcher die konkrete Implementierung vorgenommen wurde. Des Weiteren ist die Abstrakte Klasse Strategy dafür verantwortlich, dass jeweils die Methode changeCollection(ArrayList<Flashcard> fs) in den konkreten Lernstrategieklassen überschrieben wird und so die eigentliche Funktionalität bereitgestellt wird. Die Klasse Strategies hingegen ist eine Hilfsklasse, die zur Auswahl einer Lernstrategie durch den User benötigt wird.

b) Siehe SVN.

c) Dokumentation des Factory Method Patterns mit Hilfe eines UML-Klassendiagramms: