Übung 8

# Aufgabe Component Hell

## Teilaufgabe 1: Erklärung und Erläuterung des Fehlverhaltens

Um einen Überblick über das Fehlverhalten zu gewinnen, testen wir zuerst einmal alle möglichen Kombinationen durch:

1. ViewA + InsertionSortP 🡪 Ok
2. ViewA + InsertionSortQ 🡪 Fehler
3. ViewB + InsertionSortP 🡪 Fehler
4. ViewB + InsertionSortQ 🡪 Ok

Wie von Archibald bereits festgestellt, gibt es also keine spezifische Komponente, welche den Fehler verursacht. Weiter ist festzustellen:

1. Obwohl das Updaten der Visualisierung nicht immer funktioniert, so ist doch der Endwert an Swaps und Comparisons realistisch 🡪 Alle Sortieralgorithmen rufen also die Methoden des SortNotifier auf
2. Gewisse Visualisierer erkennen, dass das Endresultat nicht stimmt, andere nicht

Da wir die Applikation nicht debuggen können, aber Teilausschnitte des Codes haben, werden wir wohl oder übel in den Sourcecode eintauchen müssen. Schauen wir uns alle Klassen an, stellen wir folgendes fest:

1. Die Klasse InsertionSortQ klont das Datenarray und führt die Sortierung auf dem geklonten Array aus. Dies führt dazu, dass das sortierte Endresultat nicht im durch den Aufrufer übergebenen Array vorzufinden ist, sondern durch den SortNotifier selber sortiert und aktualisiert werden muss. Aufrufender Code welcher die ursprüngliche Referenz verwendet wird am Ende auf das unsortierte Resultat stossen, falls es InsertionSortQ verwendet
2. Die Klasse ViewB aktualisiert in der Methode notifySwap ein selbstgeführtes Datenarray und tauscht dort Werte aus. Denkt man an die Klasse InsertionSortQ zurück erscheint es nun logisch, dass ViewB damit funktioniert, ViewA aber nicht (da ViewA auf dem referenzierten Datenarray repaintet)

Möglicherweise gibt es noch weitere Probleme, da ich den lauffähigen Sourcecode aber nicht habe, höre ich an dieser Stelle auf.

## Teilaufgabe 2: Mögliche weitere Implementierungsfehler

Die Klasse ViewA übergibt dem Sortieralgorithmus immer die Referenz mit den zu sortierenden Daten – diese Referenz enthält schlussendlich auch das sortierte Endresultat. Die Klasse ViewB klont hingegen das zu übergebende Datenarray vorher immer. Was passiert nun, falls der Benutzer mit ViewA während der Laufzeit die Daten plötzlich verändert (z.B. kann er nochmals die Werte mixen lassen, da der Programmierer fälschlicherweise den Button während der Laufzeit nicht deaktiviert)? Wie geht der Sortieralgorithmus damit um, dass plötzlich sein zu bearbeitender Datensatz verändert wird?

In diesem Fall ist diese Problematik eher hypothetisch, doch wirft sie die Fragestellung auf: Worauf soll der Sortieralgorithmus wirklich sortieren (Referenz oder Datenklon) und wie gehen wir mit den Resultaten/Endresultat um (Referenz mit sortierten Daten verwenden oder Resultat via notifySwap selber zusammenbauen)?

## Teilaufgabe 3: Verhinderung der Component Hell

Generell: Im aktuellen Beispiel ist die von Archibald beschriebe Schnittstelle architekturmässig nicht genügend aussagekräftig beschrieben, sodass plötzlich designbedingte Entscheidungen in die Implementierung einfliessen – und hier fängt eben das Problem an, dass das Design des einen plötzlich zur Architektur des anderen wird und es früher oder später zu einem Problem kommt.

Was könnte man also tun:

1. Archibald sollte versuchen, die Schnittstelle besser zu beschreiben und erläutern, wie und auf welche Art und Weise eine Schnittstelle funktioniert und Aufgaben ausführt. Ein reines «was sie macht» reicht dabei nicht, da zu viel Handlungsspielraum entsteht. Beispiel:
   1. Die Klasse SortAlg stellt das Interface eines Sortieralgorithmus dar, welcher ein konkreter Algorithmus implementiert. Die Methode run(int[] data) nimmt dabei ein sortiertes oder nicht sortiertes Datenarray entgegen und sortiert die Daten live auf dieser Datenstruktur. Das Endresultat ist nach erfolgreicher Sortierung im übergebenen Datenarray vorzufinden
   2. Alternativ könnte man sagen, dass SortAlg nur die Daten intern sortiert und die sortierten Werte selber via den SortNotifier gesammelt werden müssen
2. Archibald könnte eine Referenzimplementierung schreiben, gegen welche man seine jeweilige Implementierung testen kann. In vielen Fällen ist dies aber zu zeit- und geldaufwändig, zumal ja man Firmen den Auftrag gibt eben eine solche Implementierung zu schreiben
3. Archibald könnte ein «Technology Compatibility Kit», kurz TCK schreiben, so wie es zum Beispiel bei Java und dem JRE/JDK oder analog bei JEE zum Einsatz kommt. Dieses Kit testet dann die jeweilige Implementierung/Modul mittels diversen Tests auf ihre korrekte Funktionalität und zeigt Fehler auf. Dabei ist es aber wichtig, dass das Kit auch wirklich die ganze geforderte Funktionalität testet, was im Java-Umfeld leider nicht immer der Fall ist.