Übung 03

JAN BEERBAUM [redacted] LOTTE UNCKELL [redacted]

REPORT

1. Klassendiagramm

Das Ziel dieser Übung war es, das Erzeugermuster der Fabrik, dass uns in der Vorlesung vorgetragen wurde, selbstständig an einem Beispiel zu implementieren. Die erste Schwierigkeit für uns war zu erkennen, dass wir hier nicht die eigentliche Fabrikmethode, sondern die abstrakte Fabrik wählen sollten. Im Gegensatz zu einer abstrakten Fabrik hätten wir bei der Fabrikmethode eine einzelne Fabrik für jede einzelne Produktklasse gebraucht. Da wir aber die abstrakte Fabrikmethode gewählt haben, brauchten wir nur drei Fabriken: eine für jede Firma.
Die nächste Entscheidung, die wir trafen, war die, das wir statt 9 konkreten Produktklassen nur 3 brauchten. Streng nach dem Erzeugermuster der abstrakten Fabrik hätten wir für jede Kombination aus Marke und Art des Schuhs je eine konkrete Produktklasse gebraucht. Da aber das einzige, in dem sich die Schuhe unterschieden, die Marke war entschieden wir uns, dies einfach in einem Field der klasse zu speichern. Theoretisch hätten wir so auch die Art des Schuhs in der spezifischen Instanz speichern können und die konkreten Produktklassen auch eine reduzieren können. Aber um das Muster der abstrakten Fabrik nachzuvollziehen, entschieden wir uns trotzdem für drei verschiedene Produktklassen, da sich sonst auch die drei verschiedenen Fabriken erübrigt hätten.
Nachdem wir diese Entscheidungen getroffen hatten, war es dann nicht mehr so kompliziert das Klassendiagramm zu erstellen. Wir orientierten uns an dem Klassendiagramm, dass uns in der Vorlesung gezeigt wurde. Allerdings mit einem Unterschied: unsere Klasse Schuh keine Interface, sondern abstrakt da die drei Implementierungen der Schuh Klasse viele gleiche Funktionen haben und wir so ein wenig code duplication vermeiden wollten.

2. Implementation

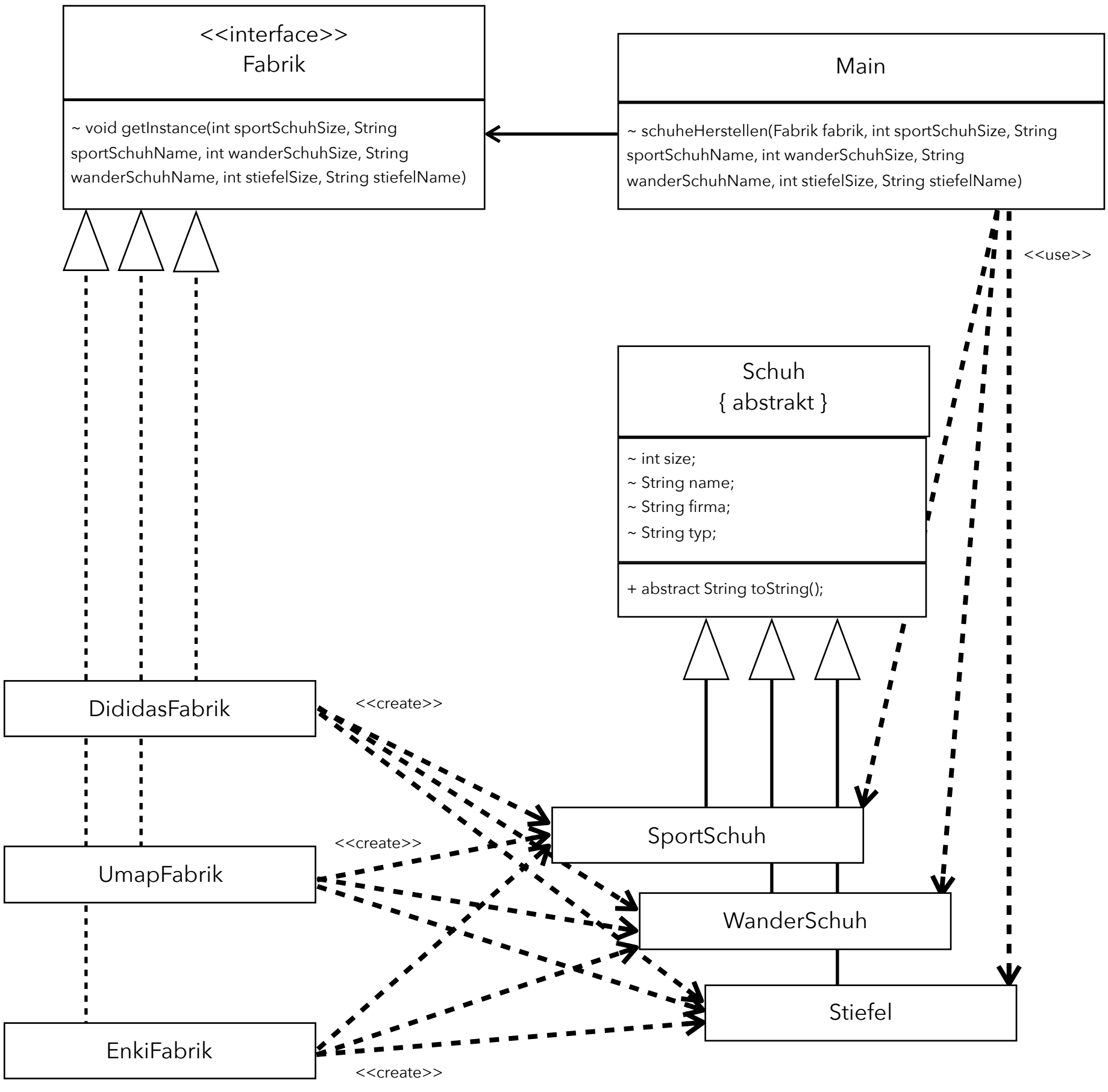
Das erstellen des Diagrams war definitiv die schwerere der beiden Aufgaben. Nachdem wir die wichtigsten Entscheidungen bereits getroffen hatten war dann unsere Implementation letzten Endes nur noch das umsetzen des Diagramms. Unser code findet sich unter: https://github.com/lotteunckell/Informatik03

3. Reflections

Obwohl wir recht ähnliche Erfahrungen gemacht haben, haben wir uns entschieden, unsere Erfahrungen jede*r in einem getrennten Teil festzuhalten um beide genauer reflektieren zu können.

Lotte Unckell:
Ich hatte zwar schon Erfahrungen mit Pair Programming, aber mit einer lockereren Form dieses Konzeptes. Zu zweit an einer Aufgabe zu arbeiten habe ich in dieser lockereren Form als sinnvoll empfunden, da ich der Meinung bin, dass so gerade bei komplizierteren Aufgaben mal die eine, mal der andere eine Idee hat, wie die weitere Vorgehensweise ist.
Was ich an dieser strengeren Art des Pair Programming persönlich nicht so toll fand war, dass wir alle 20 Minuten wechseln mussten. Zumindest bei mir führte das dazu, dass ich, als meine 20 Minuten programmieren um waren, komplett aus meinen Gedanken gerissen wurde. Mich dann auf einen komplett anderen Ansatz einzulassen und nur Fehler zu korrigieren fand ich persönlich schwierig und ich glaube ehrlich gesagt nicht, dass ich es hinbekommen habe nur korrigierend mitzuwirken, wenn ich nicht an der Tastatur saß. Vielleicht war das aber auch einfach meiner Schwäche geschuldet, Aufgaben die ich einmal angefangen habe wieder abzugeben.

Jan Beerbaum:
Ich kannte zwar das Konzept des Pair Programming, habe es aber selber vorher nie angewandt. Die ständigen, erzwungenen Unterbrechungen nach 20 Minuten empfand ich als sehr störend. Es wäre vielleicht sinnvoller, den Wechsel nicht streng nach Ablauf einer bestimmten Zeitspanne durchzuführen, sondern flexibler wenn es beiden Beteiligten passt.


<<interface>>
Fabrik
~ void getInstance(int sportSchuhSize, String sportSchuhName, int wanderSchuhSize, String wanderSchuhName, int stiefelSize, String stiefelName)
Main
~ schuheHerstellen(Fabrik fabrik, int sportSchuhSize, String sportSchuhName, int wanderSchuhSize, String wanderSchuhName, int stiefelSize, String stiefelName)
<<use>>
Schuh
{ abstrakt }
~ int size;
~ String name;
~ String firma;
~ String typ;
+ abstract String toString();
DididasFabrik
<<create>>
UmapFabrik
<<create>>
EnkiFabrik
<<create>>
SportSchuh
WanderSchuh
Stiefel