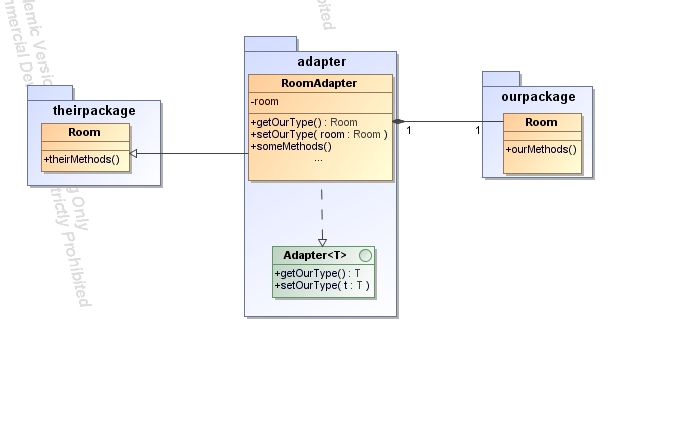
**Designentscheidung Adapter für Programmteile von Team F**

Eine Integration der anderen Programmteile durch einfache Implementierung von Interfaces in den Adapter-Klassen war in unserem Fall nicht möglich da uns keine Interfaces zur Verfügung gestellt wurden. Eine Änderung der Methoden-Schnittstellen von Team F, um diese Vorgehensweise umzusetzen, wurde von uns abgelehnt, da dafür umfangreiche Änderungen am fremden Code nötig gewesen wären. Diese Änderungen hätten wir dann bei eventuellen neueren Versionen der Programmteile wieder von neuem vornehmen müssen.

Das Problem wurde damit umgangen, dass nun unsere Adapter Klassen direkt von den Domänenobjekten des anderen Teams erben und ansonsten wie ganz normale Adapter-Klassen unser eigenes Interface beinhalten und die Methoden an ebendieses delegieren. Weiters implementiert jeder Adapter ein generisches „Adapter“-Interface welches Methoden bereitstellt um den inneren (also den von uns implementierten) Typ zu setzen und auszulesen. Dies ermöglichte uns dynamischere Programmierung, da alle unsere Adapter dieses Interface unterstützen und so ganz einfach von anderen Klassen erzeugt oder ausgelesen werden können.

Problematisch bei dieser Umsetzung ist, dass die Adapter auch die Methoden der Superklassen der adaptierten Klasse überschreiben müssen, da Java ja keine Mehrfachvererbung erlaubt und wir deshalb nicht von anderen Adaptern erben können.

Das folgende Diagramm visualisiert unser Adapter Design:



Als Einstiegspunkt für unser Programm wählten wir die „ManagerFactory“ von Team F, anstelle von konkreten Managern liefert unsere angepasste Factory nun Interfaces der Manager welche von uns konkret mit „ManagerAdapter“-Klassen umgesetzt werden. Die Schnittstellen der „ManagerAdapter“ verwenden die Klassen von Team F, intern wird aber auf unsere eigene Persistenzschicht zugegriffen indem mittels dem zuvor erwähnten Adapter-Interface unsere Typen ausgelesen und gespeichert werden oder im umgekehrten Fall eine neue Adapter-Klasse erzeugt und mit unserem inneren Typ ausgestattet wird. Die Übergabe kann dank der eingesetzten Vererbung trotzdem über die von Team F definierten Schnittstellen erfolgen.

**GUI Adaptierung**

Team F setzt auf ein SWT-GUI, welches sich nicht ohne weiteres in unser Swing GUI integrieren lässt.

Jedoch besitzen die von uns zu adaptierenden Use cases von Team F keine umfangreichen User Interfaces, der Tagesabschluss selbst wird einfach durch einer MessageBox bestätigt. Hier wurde anstelle der SWT MessageBox einfach das entsprechende Swing Pendant verwendet.

Die Akonto-Buchung selbst wurde von Team F in ihrem Habitation-Panel umgesetzt, da sich ein SWT-Panel nicht in eine Swing Anwendung integrieren lässt und wir das User Interface nicht umschreiben wollten, da Team F noch Änderungen daran vornahm, wird das entsprechende Panel nun in einem eigenen Fenster aus unserer Applikation heraus geöffnet.