# Unsere Architektur

Unser Ziel ist es, die Anwendung nach dem Model-View-Controller Pattern aufzubauen. Hiermit wollen wir

* Wartbarkeit garantieren. Die Funktionalität soll möglichst streng von der Ansicht getrennt sein.
* sicherstellen, dass man jede Komponente testen kann. Durch die Trennung in MVC lassen sich einfacher Unit-Tests schreiben, da man kein ganzes Umfeld inklusive View generieren muss um zu testen. Stattdessen kann man einfach die Abläufe in den Controllern testen.

unsere Anwendung skalierbar machen. Der Aufwand eine neue Datenbank anzubinden oder diese auszutauschen ist minimiert.

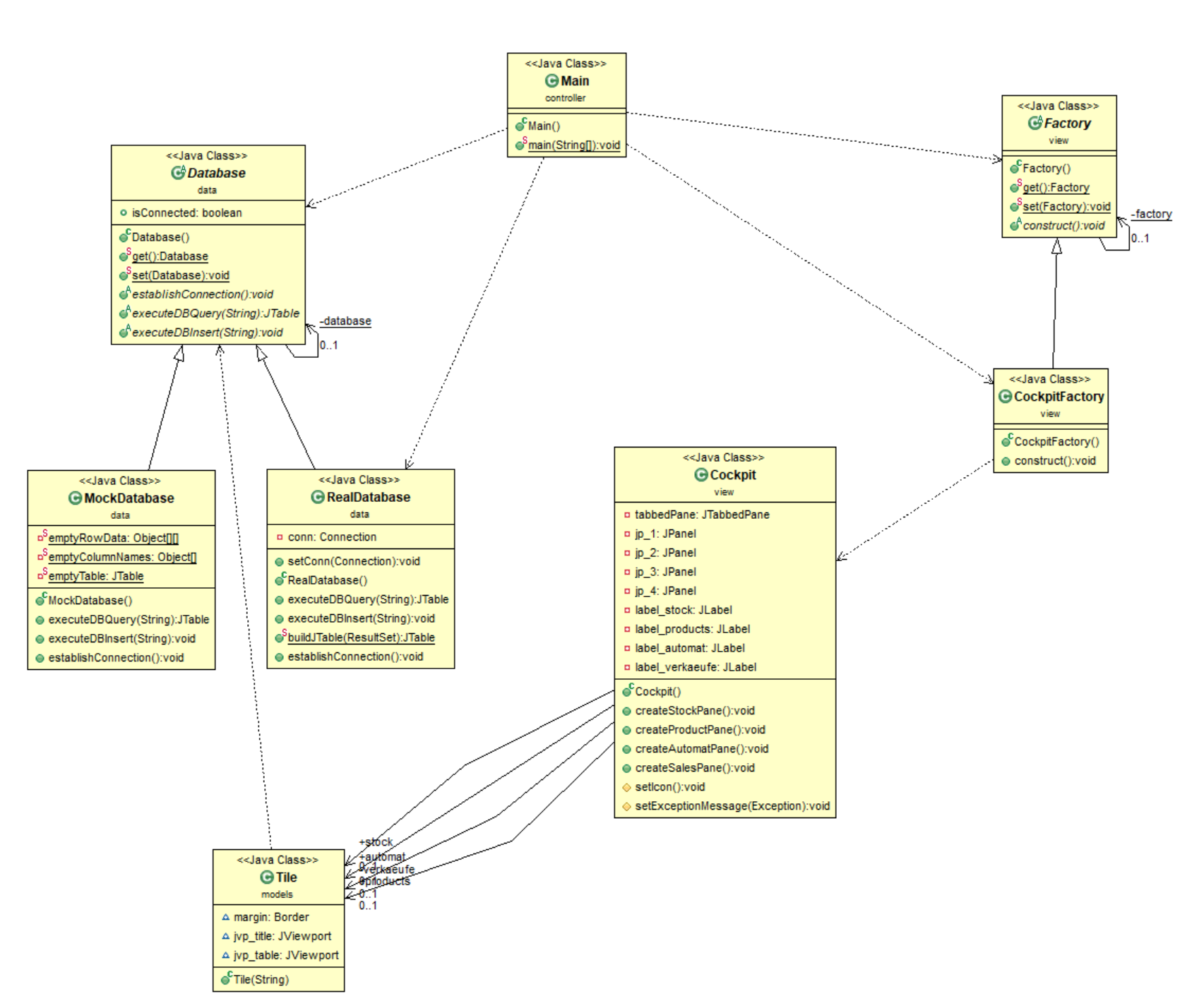
Die „Austauschbarkeit“ der Komponenten (Bezahlvorgang, Datenbank, Anmeldeprozess, Fehlerbehandlung) wird durch abstrakte Klassen und Vererbung sichergestellt. Die Konfiguration, welche Vorgänge bzw. Datenquellen verwendet werden, findet vor Starten der Anwendung in der Klasse ‚Main‘ statt. Die 4 Optionen, die Anwendung den eigenen Wünschen anzupassen, sind alle problemlos austauschbar.

Einem Vordringen auf den internationalen Markt steht nichts entgegen. Das *ButcherCockpit* kann entweder auf Deutsch oder auf Englisch verwendet werden.

Das *ButcherCockpit* muss nicht immer eine Metzgeranwendung bleiben. Die Anfragen auf die Datenbank können in den Select\_Statements angepasst werden. Eine Umstellung von Fleisch als Portion zu Gemüse als Portion ist mit keinem zusätzlichen Aufwand verbunden, sofern die Datenbank für Gemüse alle erforderlichen Attribute für die Klasse Portion bereitstellt.

Einem Handover an ein anderes Entwicklerteam steht nichts im Weg: Wir stellen ausführliche Dokumentation und einen Debug-Modus in der Fehlerbehandlung bereit.

# Erzeugung des Cockpits mit Tiles



Das ist gut:

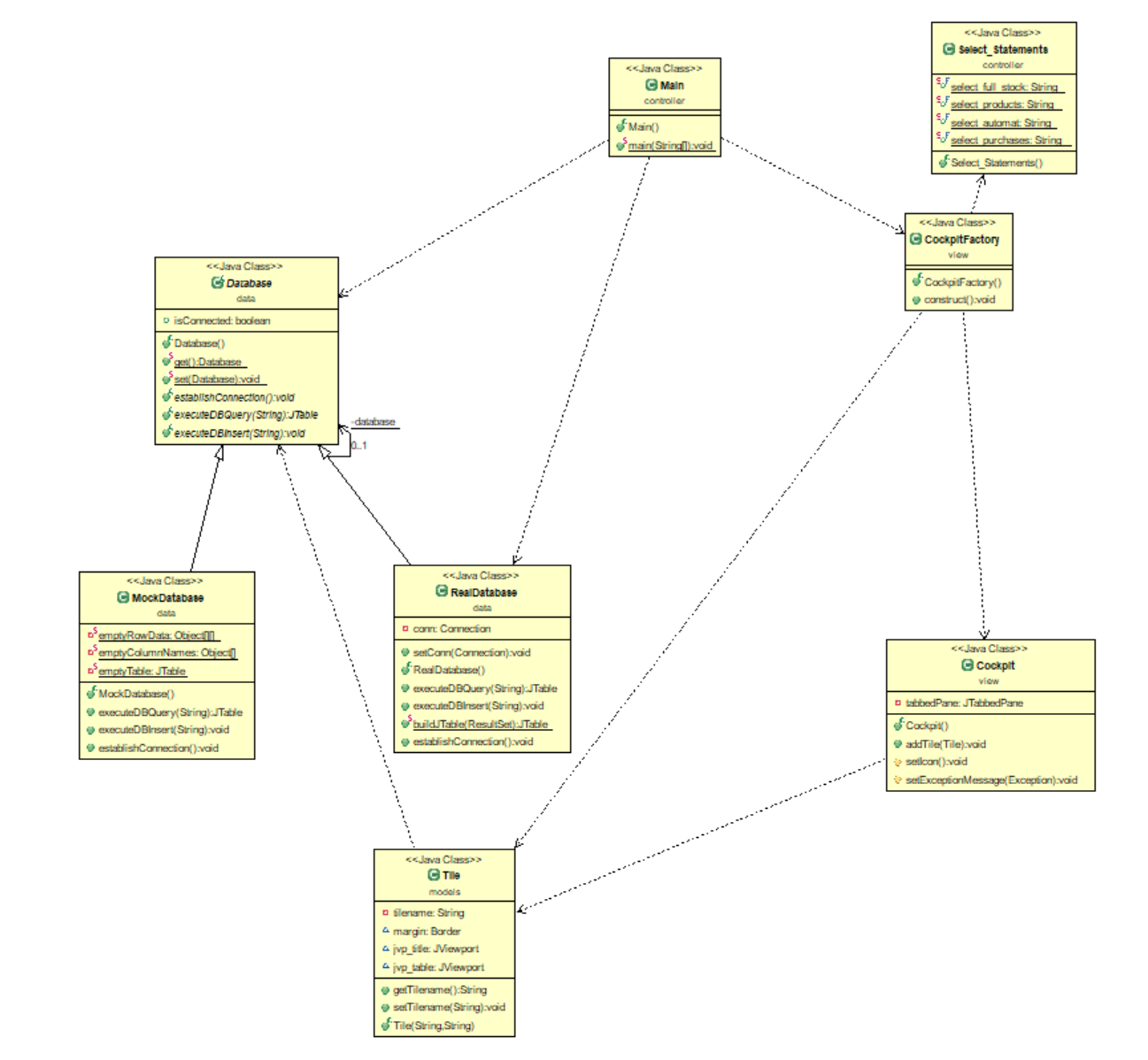
* Automat wird von einer Factory erstellt -> die kann für ihn alles steuern

Probleme:

* Anzahl der Tiles ist fest
* feste Knüpfung an die Datenbank
* Tiles sollten die DB nicht kennen müssen bzw nicht mit ihr interagieren!

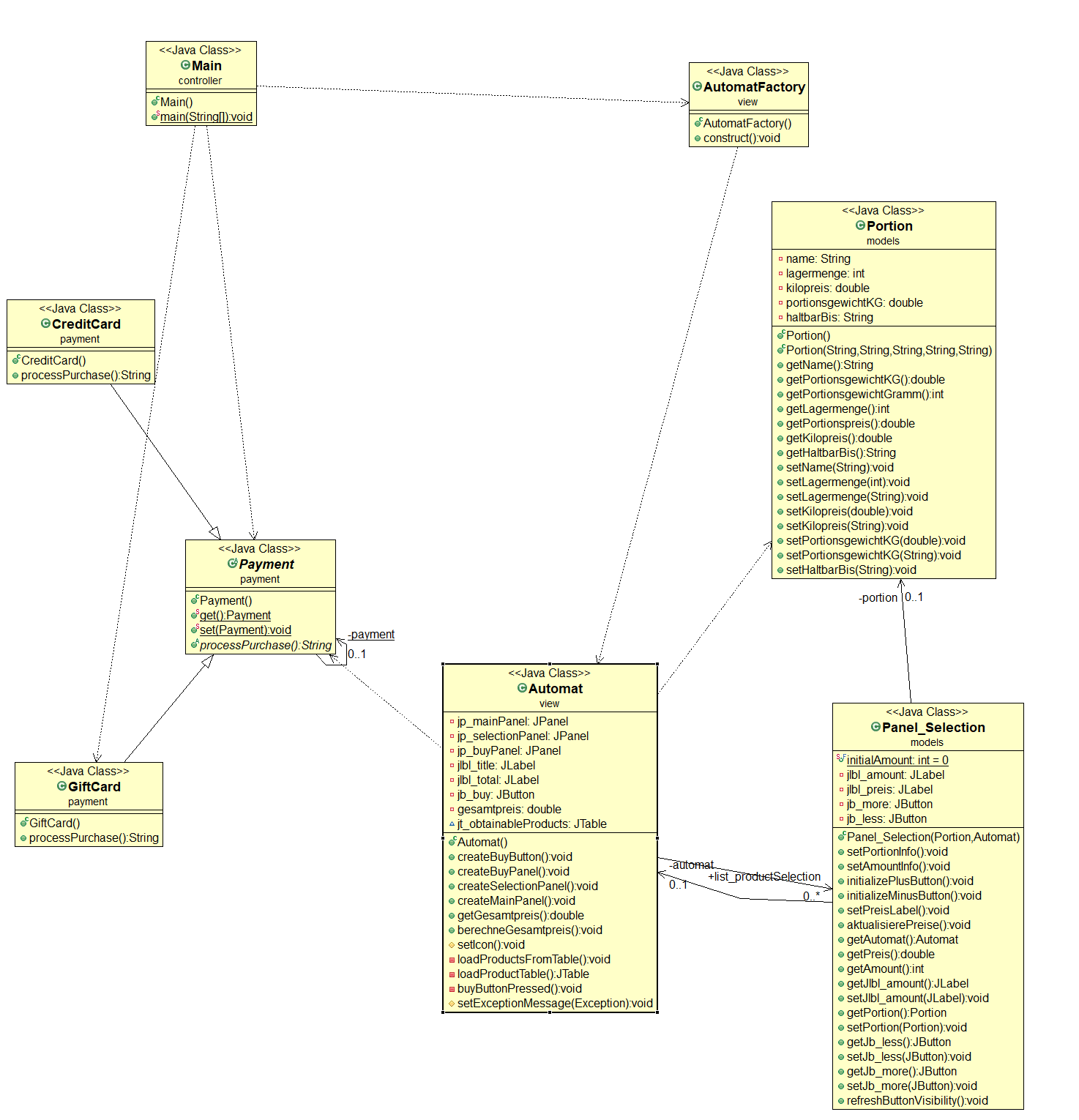
Lösung:

* Wir führen eine Methode addTile() im Cockpit ein
* die Factory erzeugt die Tiles für das Cockpit

Jetzt muss das Cockpit nicht mehr selbst die Tiles erzeugen, das macht die Cockpit-Factory.

Die Anzahl der Tiles und der Inhalt kann in der Factory bestimmt werden.

# Erzeugung des Automats



Das ist gut:

* Automat hat die Panels

Das ist nicht gut:

* Die Panels kennen den Automaten! Ganz mies

Was will ich umsetzen?

1. Die Panel-Selection sollte nicht den Automaten kennen müssen.
2. ~~der Automat soll die Tabellen gegeben bekommen (Factory)~~
3. Benachrichtigen des Automaten mithilfe eines Observers?

Gestaltung der Benachrichtigung:

Das Panel\_Selection muss seinen passenden Automaten benachrichtigen, falls die ausgewählte Menge verändert wurde.

Was jetzt geändert wurde:

Automat ist der Property Listener, Panel fired das Event falls die Ausgewählte Menge verändert wird.

Die Automat Factory erzeugt die Selection Panels, nicht der Automat selber.

