Advanced Software-Engineering Programmentwurf

1. Anforderungen überprüfen

a. Karte

Anforderung	Erfüllt?	Bemerkung
Die Eigenschaft PIN wird beim Anlegen einer Karte gesetzt und kann danach nicht mehr verändert werden.	Ja	
Jede Karte verfügt über die Möglichkeit, eine am Automat eingegebene PIN mit der in der Karte gespeicherten PIN abzugleichen.	Ja	
Der Automat darf nur dann Geld auszahlen, wenn die korrekte PIN eingegeben wurde.	Nein	Der Wert von pinFalsch wird nie zurückgesetzt → Wird einmal das Limit an Falscheingaben überschritten kann danach mit keiner Karte mehr Geld abgehoben werden.
Eine PIN muss vierstellig sein und aus Ziffern bestehen (keine Buchstaben, keine Sonderzeichen).	Ja	
Es muss möglich sein, vierstellige PINs anzulegen, die mit einer oder mehreren Nullen beginnen.	Nein	Es kann Null nicht hinzugefügt werden, da bei der Erzeugung der PIN die while-Schleife bei der Zahl Null (0.0000…) nie verlassen wird (Endlosschleife).

b. Anwendung

Anforderung	Erfüllt?	Bemerkung
Die Klasse Anwendung steuert den Geldautomat. Sie ist als textbasierte Konsolenanwendung ausgeführt und bietet dem Benutzer alle Möglichkeiten, um am Geldautomat Geld abzuheben.	Ja	

c. Geldautomat

Anforderung	Erfüllt?	Bemerkung
Klasse implementiert die Funktionalitäten, die von der Klasse Anwendung gesteuert werden.	Ja	
Der Automat startet im Zustand "Kein Geld".	Jein	Zustände sind nicht implementiert. Funktionsweise wird aber über andere Variablen gesichert.
Durch das Aufrufen der Methode bestücken kann der Automat in den Zustand "Bereit" versetzt werden.	Jein	Zustände sind nicht implementiert. Funktionsweise wird aber über andere Variablen gesichert.
Der Automat kann nur bestückt werden, wenn sich keine Karte in ihm befindet.	Ja	
Wird eine Karte in einen leeren Automat eingeschoben, wird sie wieder ausgeworfen und eine Fehlermeldung ausgegeben.	Nein	Die Karte wird nicht ausgegeben, alle Funktionen außer dem Geld Abheben bleiben bestehen.
Wird eine Karte in einen betriebsbereiten Automat eingeschoben, dann ändert sich sein Zustand nach "Karte drin".	Jein	Zustände sind nicht implementiert. Funktionsweise wird aber über andere Variablen gesichert.
Befindet sich eine Karte im Automaten, muss die passende PIN eingegeben werden, damit mit der Auszahlung fortgefahren werden kann.	Nein	Wenn man mehr Geld abheben möchte als der Automat hat, kann dies auch ohne Eingabe der Pin durchgeführt werden.
Wird die korrekte PIN eingegeben, dann ändert sich der Zustand nach "PIN korrekt" und es kann so lange Geld ausbezahlt werden, wie der Automat Bargeldreserven hat.	Jein	Zustände sind nicht implementiert. Funktionsweise wird aber über andere Variablen gesichert.
Der jeweilige Auszahlungsbetrag muss dabei zwischen 5 und 500 Taler liegen.	Ja	
Da der Automat nur Scheine und keine Münzen hat, muss jeder Auszahlungsbetrag durch 5 teilbar sein.	Nein	Es kann ein beliebiger Betrag zwischen 5 und 500 Talern ausgezahlt werden.

Wird in einer Transaktion mehr Geld angefordert als der Automat Bargeld besitzt, zahlt der Automat das aus, was er noch hat und geht dann in den Zustand "Kein Geld" über.	Jein	Zustände sind nicht implementiert. Funktionsweise wird aber über andere Variablen gesichert.

2. Design-Pattern anwenden

Design-Pattern	Angewendet	Begründung
State	Ja	Geldautomat hat laut Anforderung unterschiedliche Zustände. Diese wurden allerdings durch verschachtelte if-else-Abfragen implementiert. Durch die Anwendung des Patterns können die Methoden des Geldautomats je nach aktuellem Zustand angepasst werden. Dadurch wird die Verständlichkeit, Wartbarkeit und Erweiterbarkeit des Codes erhöht.
Singleton	Nein	Innerhalb der Anwendung soll nur einzelner Geldautomat existieren. Somit ist es sinnvoll, dass von dieser Klasse nur ein einzelnes Objekt erzeugt werden kann. Die Klasse Anwendung ist ebenfalls nur für die Steuerung eines einzelnen Geldautomaten ausgelegt. Problematisch dabei ist, dass die Unit-Tests extrem angepasst werden müssten, damit Funktionalitäten überprüft werden können. Daher ist die Entscheidung gegen das Pattern gefallen.
Strategy	Nein	Die Methoden der Klasse Geldautomat verhalten sich je nach aktuellem Zustand anders. Würde man dieses Pattern anwenden, müsste man allerdings für jede Methode der Klasse ein Interface erzeugen, über das in Unterklassen die Strategie je nach Zustand gesetzt werden kann. Man müsste also extrem viele Klassen erzeugen und hätte dann eine schwer zu verstehende und zu wartende Codebasis.
Observer	Nein	Ein Objekt der Klasse Geldautomat existiert unabhängig von anderen Klassen. Daher macht es keinen Sinn, dass ein solches Objekt überwacht werden kann. Eine Zustandsänderung im Geldautomat hat keine Auswirkung auf andere Klassen der Anwendung.

3. Design-Prinzipien anwenden

Design-Prinzip	Angewendet	Begründung
Single-Responsibility	Ja	Die Methoden sollen je nach Funktionalität in der entsprechenden Klasse sein. Beispielsweise wurde daher die Funktion erzeugePin() von Anwendung nach Karte verschoben.
Interface-Segregation	Ja	Jeder Zustand des Geldautomaten befindet sich in einer eigenen Klasse und wird über ein Interface aufgerufen. Dadurch erhöht sich die Wartbarkeit und die Methoden müssen weniger komplex sein.

4. Anmerkungen

Beim Aufruf der Methoden des Geldautomaten in der Anwendungs-Klasse wird der aktuelle Zustand des Geldautomaten abgefragt. Das ist keine schöne Lösung, aber nur so kann sichergestellt werden, dass Eingaben nicht immer möglich sind. Außerdem muss dies so gemacht werden, da sonst die Benutzereingaben nicht mehr von der Anwendungs-Klasse verarbeitet werden (s. Anforderungen).