A1 User Stories: INVEST-Kriterien

Analysieren Sie die folgenden Anforderungen auf ihre Eignung als User Stories auf Grundlage der INVEST-Kriterien.

a) »Als Nutzer möchte ich mein Passwort zurücksetzen können.«

b) »Als Vertriebsmitarbeiter möchte ich neue Marketingkampagnen anlegen können, damit ich gezielt die Kundenbindung intensivieren kann.«

c) »Als Abonnent möchte ich auf der Einstellungsseite festlegen können, ob ich alle 5, 10, 15 oder 60 Minuten über die neuesten Meldungen benachrichtigt werde, damit ich die Benachrichtigungen nach meinen Bedürfnissen konfigurieren kann.«

d) »Als Bewerber möchte ich meine persönlichen Daten löschen können, damit die Datenschutzgrundverordnung erfüllt wird.«

e) »Als Administrator möchte ich das Authentifizierungsverfahren wechseln können, damit ich zukünftige Änderungen am Identitätsmanagement berücksichtigen kann.«

Lösung 1

Die INVEST-Kriterien sind Prinzipien, die sicherstellen sollen, dass User Stories im Agilen Software Development gut formuliert sind. INVEST steht für:

- Independent (Unabhängig): Jede User Story sollte unabhängig sein. Dies bedeutet, dass sie unabhängig von anderen Stories entwickelt und implementiert werden kann, was die Planung und Priorisierung erleichtert.
- Negotiable (Verhandelbar): Eine User Story ist keine starre Vertragsanforderung, sondern ein Ausgangspunkt für Diskussionen. Entwickler und Stakeholder sollten den Inhalt und die Anforderungen der Story verhandeln können.
- Valuable (Wertvoll): Jede User Story sollte einen klaren Wert für den Kunden oder Benutzer liefern. Der Fokus sollte darauf liegen, dass der Endbenutzer von der Umsetzung der Story profitiert.
- Estimable (Schätzbar): Eine User Story sollte so formuliert sein, dass das Entwicklungsteam in der Lage ist, den Aufwand für ihre Umsetzung zu schätzen. Eine unklare oder zu komplexe Story macht die Schätzung schwierig.
- Small (Klein): User Stories sollten klein genug sein, um in einem Sprint oder in einer kurzen Zeitspanne umgesetzt werden zu können. Dies erleichtert die Planung und ermöglicht schnelle Feedbackzyklen.
- Testable (Testbar): Jede User Story sollte testbare Kriterien enthalten, um festzustellen, ob die Story erfolgreich umgesetzt wurde. Ohne diese Kriterien ist es schwer zu beurteilen, ob die Anforderungen der Story erfüllt wurden.

a) *»Als Nutzer möchte ich mein Passwort zurücksetzen können.«*
Diese User Story ist indipendent, valuable, estimable, small, und auch testable. Verhandelbar ist sie jedoch nur im Hinblick auf die Ausgestaltung.
→ Dennoch ist es eine gut formulierte User Story.

b) *»Als Vertriebsmitarbeiter möchte ich neue Marketingkampagnen anlegen können, damit ich gezielt die Kundenbindung intensivieren kann.«*
Es ist schwer zu testen, ob sich Kundenbindungen intensiviert haben und die Story gibt auch keine überprüfbaren Kriterien dafür her. Bei Marketingkampagnen ist es auch ziemlich ausgeschlossen, dass sie einen Mehrwert für die Menschheit bieten, auch wenn der Vertriebsmitarbeitende womöglich davon profitieren könnte.
→ Diese Anforderung ist schlecht formuliert und ungeeignet als User Story.

c) *»Als Abonnent möchte ich auf der Einstellungsseite festlegen können, ob ich alle 5, 10, 15 oder 60 Minuten über die neuesten Meldungen benachrichtigt werde, damit ich die Benachrichtigungen nach meinen Bedürfnissen konfigurieren kann.«*
Diese User Story ist mglw. von einem Abo-System, Benachrichtigungssystem und einer Einstellungsseite abhängig. Dafür bietet sie jedoch Spielraum für Verhandlung, ist klein, testbar und der Aufwand kann in einem bestehenden System gut abgeschätzt werden.
→ Als Feature für eine App, welche bereits über solche grundlegenden Systeme verfügt, ist die Anforderung als User Story geeignet.

d) *»Als Bewerber möchte ich meine persönlichen Daten löschen können, damit die Datenschutzgrundverordnung erfüllt wird.«*
Diese User Story ist fehlerhaft, da zur Erfüllung der DSGVO die Bewerbungen nach Beendigung des Bewerbungsverfahrens nicht vom Bewerber, sondern vom Arbeitgeber gelöscht werden müssen. Ein Mehrwert wäre unter dieser Voraussetzung also nicht gegeben; die anderen Kriterien sind erfüllt.
→ Die User Story könnte die Kriterien erfüllen, sofern der Use Case real sein sollte.

e) *»Als Administrator möchte ich das Authentifizierungsverfahren wechseln können, damit ich zukünftige Änderungen am Identitätsmanagement berücksichtigen kann.«*
Diese User Story ist nicht klar genug formuliert und wohl kaum test- oder schätzbar. Ob Sie klein und unabhängig ist, lässt sich bei so ungenauer Formulierung kaum sagen, aber dafür lässt sie genug Freiraum zu verhandeln.
→ Es liegt keine geeignete User Story vor.

A2 Hybrid Mobile Application

Szenario

Sie wollen objektorientiert ein hybrides Application Framework entwickeln, mit dem Sie einheitlich graphische Benutzerschnittstellen in Android und iOS steuern können. Sie benötigen die folgenden Klassen:

- Eine generalisierte Klasse `MobileElement` mit dem schreibgeschützten Attribut `getTitle()` vom Datentyp String sowie den rückgabelosen Operationen `render()` und `register(MobileForm form)`.
- Eine generalisierte Klasse `MobileForm` mit folgenden Klassenmitgliedern:
  - Liste der enthaltenen mobilen Elemente vom Typ `MobileElement`. Das Verhältnis ist eine Teile-Ganze-Beziehung.
  - Operation `addElement(string title)`: es wird ein neues mobiles Element mit dem angegebenen Titel erzeugt. Der Titel kann nur bei der Erzeugung festgelegt werden. Während der Erzeugung wird die zugehörige `MobileForm` registriert.
  - Operation `render()`, die die gleichnamige Operation in allen enthaltenen mobilen Elementen aufruft.
- Von `MobileElement` und `MobileForm` sollen keine Instanzen erzeugt werden können.
- Konkrete Elemente und konkrete Formen für Android und iOS, die erzeugt werden können. Die Betriebssysteme einer konkreten Form und den darin enthaltenen konkreten Elementen passen immer zusammen.
- Alle oben genannten Klassen sind im Paket `mobileapplication.ui` enthalten. Aus Sicherheitsgründen sollen neue konkrete Elemente von außerhalb des Pakets nur über die Operation `addElement` von `MobileForm` erzeugt werden können.
- Die Klasse `App` im Paket `mobileapplication` mit den folgenden Eigenschaften:
  - Attribut `form` vom Datentyp `MobileForm`.
  - Private Operation `getOS()`, die das aktuelle Betriebssystem als String zurückgibt.
    Mögliche Werte sind zurzeit »Android« und »iOS«.
  - Private Operation `initialize()`, die dem Attribut `form` eine neue Instanz passend zum Betriebssystem zuweist.
  - Operation `execute()`, die `initialize()` aufruft, drei neue Elemente der konkreten Form hinzufügt und anschließend alle Elemente rendert.

Aufgaben

a) Entwerfen Sie eine Lösung als Klassenverband. Nutzen Sie dafür das Entwurfsmuster Factory Method, wobei `MobileForm` die Rolle von Creator und `MobileElement` die Rolle von `Product` übernehmen soll. Modellieren Sie den Klassenverband als UML-Klassendiagramm.

b) Modellieren Sie die Abfolge der Aufrufe als UML-Sequenzdiagramm. Beginnen Sie mit der Operation `execute()` der Klasse App als gefundene Nachricht. Nutzen Sie dabei die abstrakten und nicht die konkreten Klassen.

c) Implementieren Sie den Rumpf der Lösung in Java.

Lösung 2

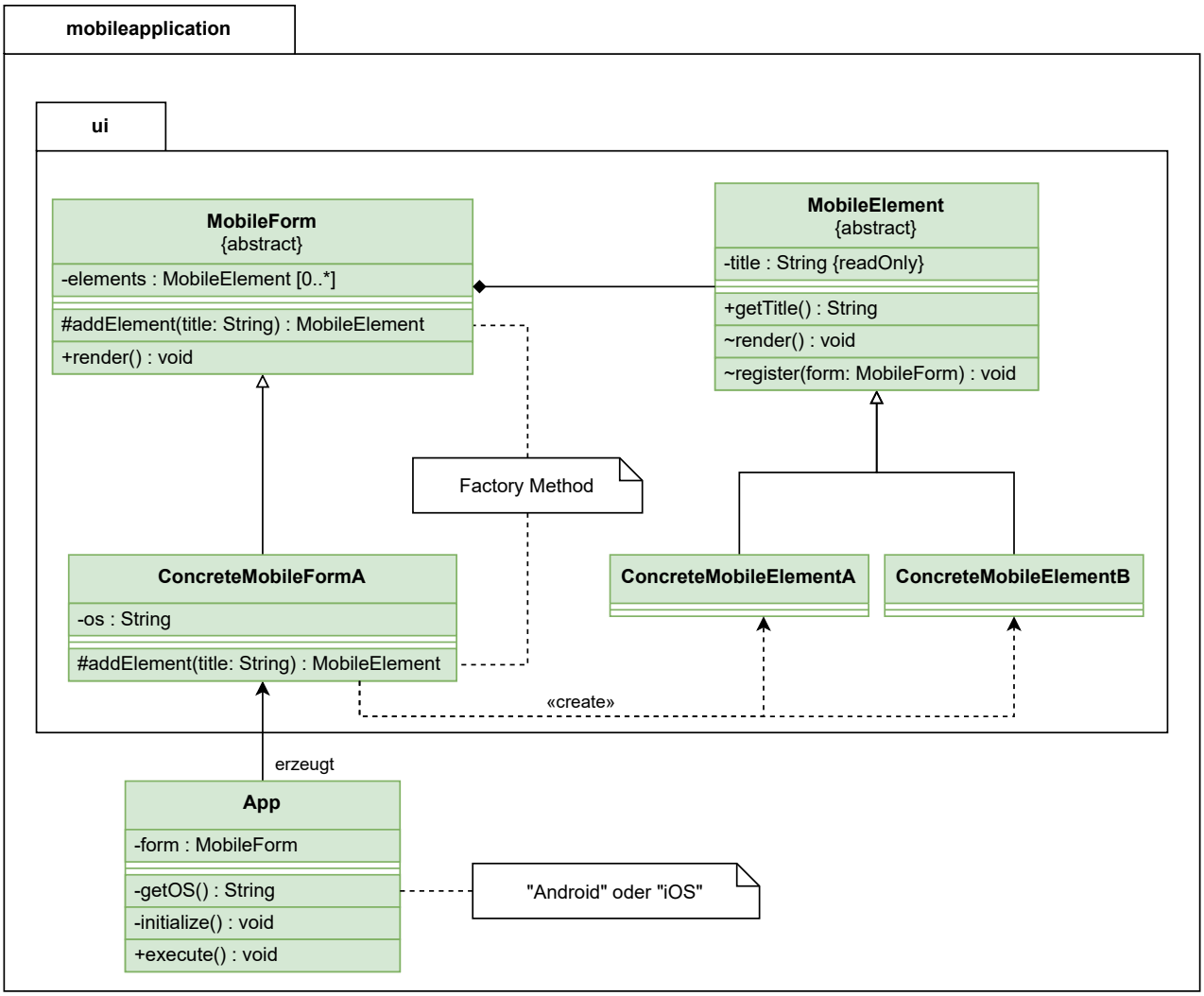

Abbildung 1: Lösung der Aufgabe 2a

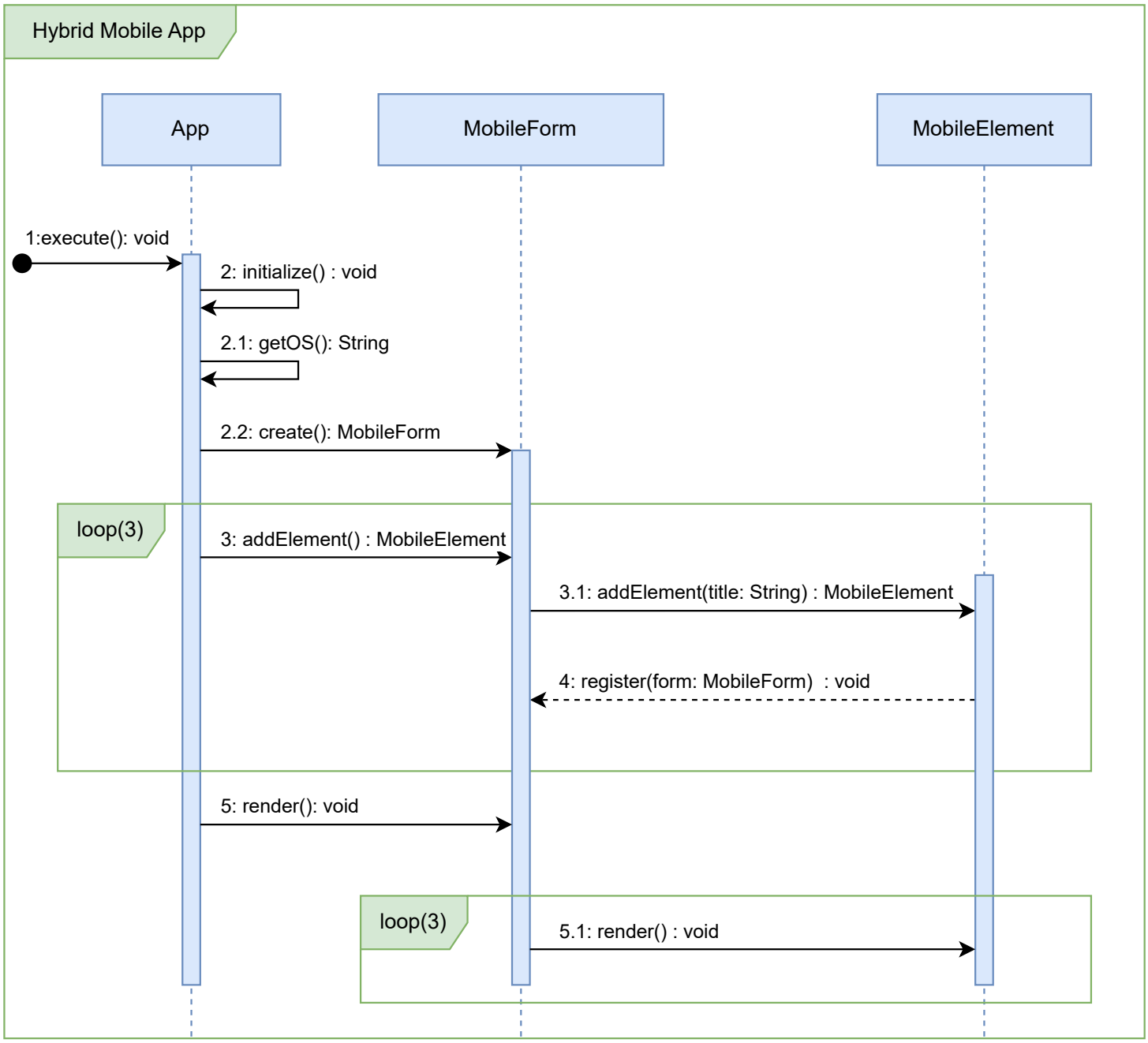

Abbildung 2: Lösung der Aufgabe 2b

Lösung 2c

```java
// UI Package
package mobileapplication.ui;

public abstract class MobileForm {
    protected List<MobileElement> elements;

    protected MobileForm();

    public abstract MobileElement addElement(String title);

    public void render();
}

public abstract class MobileElement {
    private final String title;

    protected MobileElement(String title);

    public String getTitle();

    protected abstract void render();
    protected abstract void register(MobileForm form);
}
```

```java
// Beispiel für eine ConcreteMobileForm
public class AndroidMobileForm extends MobileForm {
    private String os = "Android";

    @Override
    protected MobileElement addElement(String title);
}

// Beispiel für ein ConcreteMobileElement
public class AndroidMobileElement extends MobileElement {
    public AndroidMobileElement(String title);

    @Override
    protected void render();

    @Override
    protected void register(MobileForm form);
}
```

```java
// MobileApplication Package
package mobileapplication;

import mobileapplication.ui.MobileForm;

public class App {
    private MobileForm form;

```

```
9       private String getOS();
10      private void initialize();
11      public void execute() {
12          initialize();
13      }
14  }
```