

Aufgabe A1 – Mobile Application

a) Klassenverband (Factory Method)

Klassen im Paket mobileapplication.ui

MobileElement (abstrakt, Product) • schreibgeschütztes Attribut `title : String`

- Referenz auf die zugehörige MobileForm
- Operationen: `render()` (abstrakt), `register(MobileForm form)`, `getTitle()`

MobileForm (abstrakt, Creator) • Aggregation von `List<MobileElement> elements` (Teile-Ganze-Beziehung)

- Factory Method `createElement(String title)` (*protected abstract*)
- konkrete Methode `addElement(String title)`: ruft `createElement`, registriert das Element bei der Form und fügt es `elements` hinzu
- `render()` delegiert an `render()` aller enthaltenen Elemente

AndroidForm / IOSForm (konkrete Creator) • erben von MobileForm

- implementieren `createElement(...)` und erzeugen jeweils passende konkrete Elemente

AndroidElement / IOSElement (konkrete Products) • erben von MobileElement

- Konstruktoren mit Paket-Sicht, sodass Instanzen nur innerhalb von mobileapplication.ui erzeugt werden können
- implementieren `render()` betriebssystemspezifisch

Klasse im Paket mobileapplication

App • Attribut `form : MobileForm`

- private Operation `getOS()` liefert „Android“ oder „iOS“
- private Operation `initialize()` erzeugt passende konkrete Form (AndroidForm oder IOSForm) und setzt `form`
- `execute()` ruft `initialize()`, fügt drei neue Elemente über `form.addElement(...)` hinzu und ruft anschließend `form.render()`

UML-Beziehungen (textuelle Skizze)

- `MobileElement` \triangleleft - `AndroidElement`, `IOSElement` (Vererbung; abstraktes Product).
- `MobileForm` \triangleleft - `AndroidForm`, `IOSForm` (Vererbung; abstrakter Creator mit Factory Method `createElement`).
- `MobileForm` \blacklozenge - `MobileElement` (Komposition über `elements`).
- App - `MobileForm` (Assoziation über Attribut `form`); App nutzt nur die abstrakten Typen und ist nicht an konkrete OS-Klassen gekoppelt.

b) Sequenzdiagramm (Android)

Gefundene Nachricht: `App.execute()`.

Mobile Application - UML-Klassendiagramm (Factory Method)

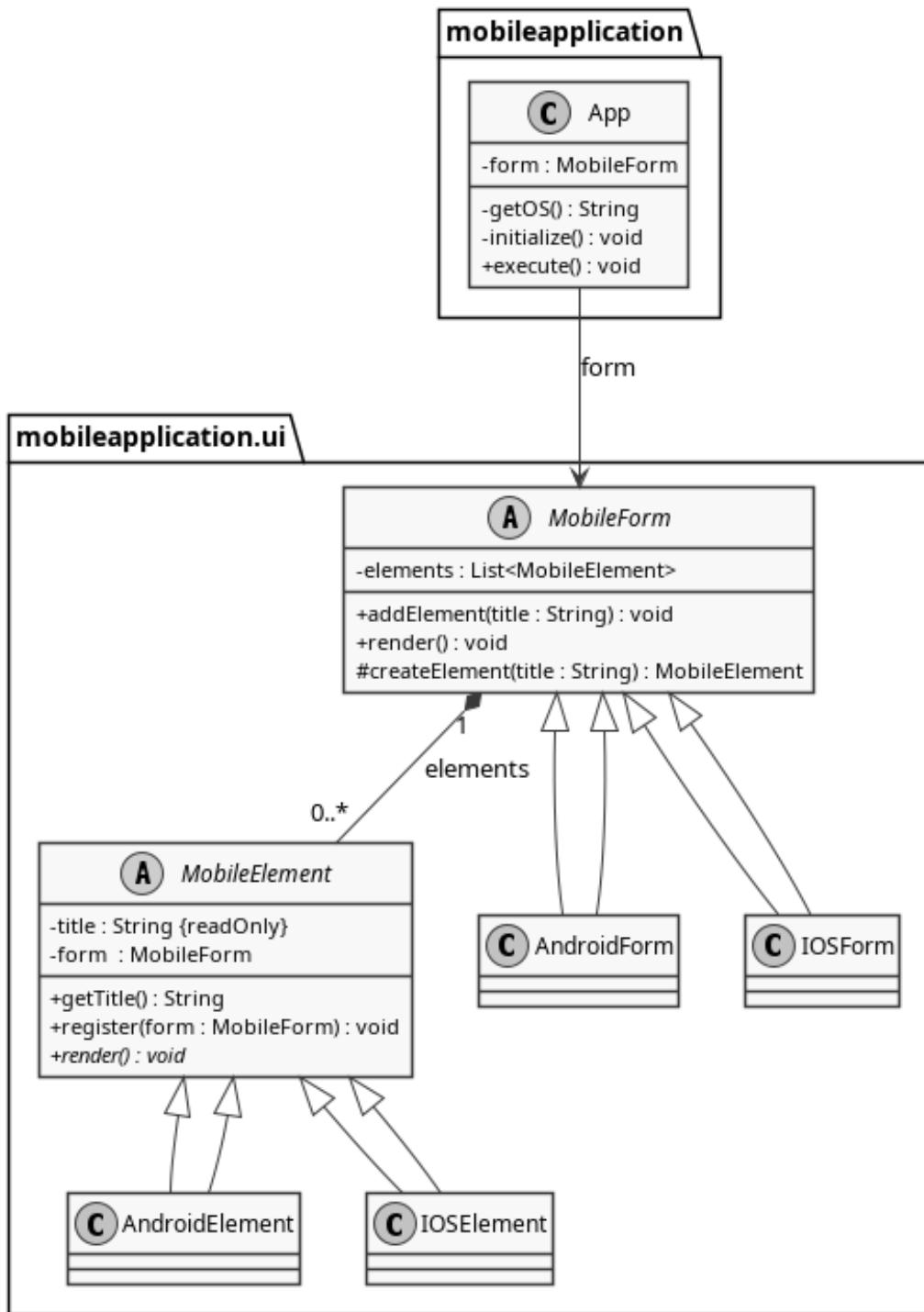


Abbildung 1: UML-Klassendiagramm für das Mobile-Application-Framework (modelliert mit PlantUML)

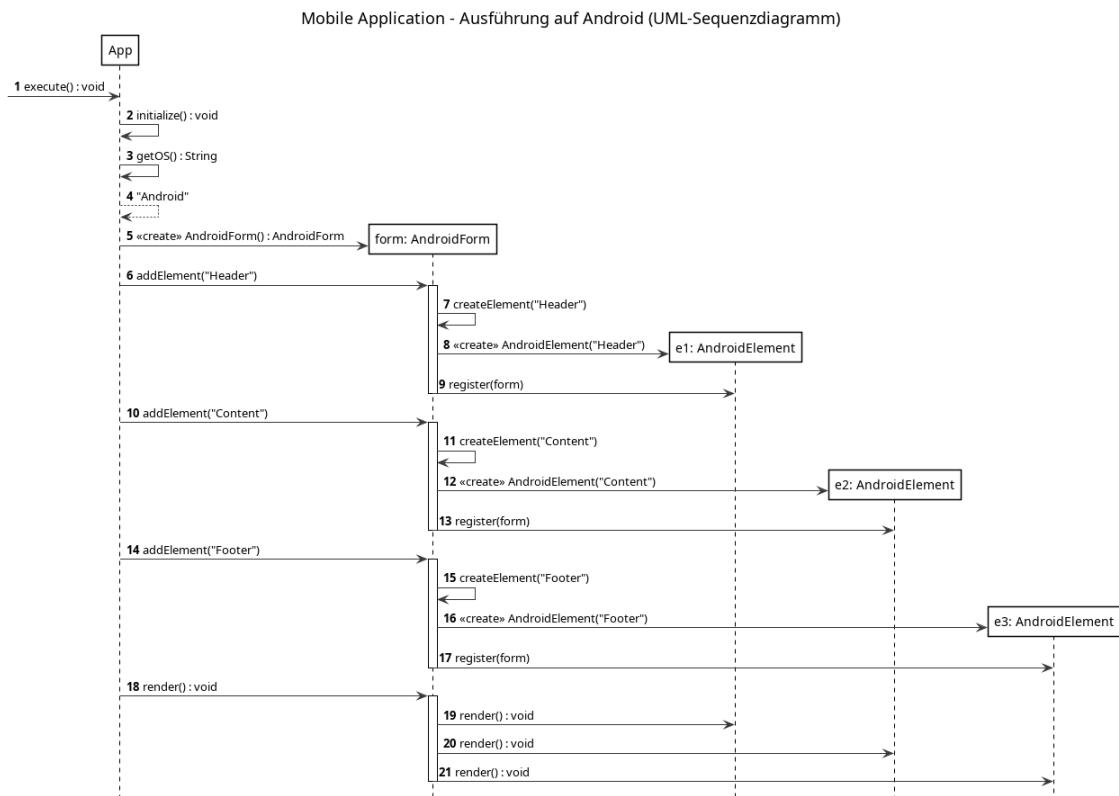


Abbildung 2: UML-Sequenzdiagramm der Ausführung von `App.execute()` auf Android (modelliert mit PlantUML)

c) Java-Rumpf (Paketstruktur angedeutet)

```
1 // mobileapplication/ui/MobileElement.java
2 package mobileapplication.ui;
3
4 public abstract class MobileElement {
5     private final String title;
6     private MobileForm form;
7
8     protected MobileElement(String title) { // nur Paket/SUB-Klassen
9         this.title = title;
10    }
11
12    public String getTitle() { return title; }
13
14    public void register(MobileForm form) {
15        this.form = form;
16    }
17
18    public MobileForm getForm() { return form; }
19
20    public abstract void render();
21 }
22
23 // mobileapplication/ui/MobileForm.java
24 package mobileapplication.ui;
25
26 import java.util.ArrayList;
27 import java.util.List;
28
29 public abstract class MobileForm {
30     private final List<MobileElement> elements = new ArrayList<>();
31
32     protected abstract MobileElement createElement(String title);
33
34     public final void addElement(String title) {
35         MobileElement element = createElement(title);
36         element.register(this);
37         elements.add(element);
38     }
39
40     public void render() {
41         elements.forEach(MobileElement::render);
42     }
43 }
44
45 // mobileapplication/ui/AndroidElement.java
46 package mobileapplication.ui;
47
48 class AndroidElement extends MobileElement {
49     AndroidElement(String title) { super(title); }
50
51     @Override
52     public void render() {
```

```
53         System.out.println("Android-Element: " + getTitle());
54     }
55 }
56
57 // mobileapplication/ui/IOSElement.java
58 package mobileapplication.ui;
59
60 class IOSElement extends MobileElement {
61     IOSElement(String title) { super(title); }
62
63     @Override
64     public void render() {
65         System.out.println("iOS-Element: " + getTitle());
66     }
67 }
68
69 // mobileapplication/ui/AndroidForm.java
70 package mobileapplication.ui;
71
72 public class AndroidForm extends MobileForm {
73     @Override
74     protected MobileElement createElement(String title) {
75         return new AndroidElement(title);
76     }
77 }
78
79 // mobileapplication/ui/IOSForm.java
80 package mobileapplication.ui;
81
82 public class IOSForm extends MobileForm {
83     @Override
84     protected MobileElement createElement(String title) {
85         return new IOSElement(title);
86     }
87 }
88
89 // mobileapplication/App.java
90 package mobileapplication;
91
92 import mobileapplication.ui.AndroidForm;
93 import mobileapplication.ui.IOSForm;
94 import mobileapplication.ui.MobileForm;
95
96 public class App {
97     private MobileForm form;
98
99     private String getOS() {
100         return Math.random() > 0.5 ? "Android" : "iOS"; // Platzhalter
101     }
102
103     private void initialize() {
104         String os = getOS();
105         form = "Android".equals(os) ? new AndroidForm() : new IOSForm();
106     }
107 }
```

```
107  
108     public void execute() {  
109         initialize();  
110         form.addElement("Header");  
111         form.addElement("Content");  
112         form.addElement("Footer");  
113         form.render();  
114     }  
115 }
```

Kernideen: Factory Method (`createElement`) koppelt App von den konkreten OS-Klassen ab und stellt sicher, dass Form und Elemente stets zusammenpassen. Kapselung über paket-private Konstruktoren verhindert die Erzeugung konkreter Elemente außerhalb von `mobileapplication.ui`.