

## Aufgabe A1 – Mobile Application

### a) Klassenverband (Factory Method)

- **mobileapplication.ui.MobileElement** (abstract, Produkt): schreibgeschütztes Attribut `title:String`, Referenz auf `MobileForm`, Operationen `register(form)`, `render()` (abstrakt), Getter `getTitle()`.
- **mobileapplication.ui.MobileForm** (abstract, Creator): Aggregiert `List<MobileElement> elements`; Factory Method `createElement(title)` (*protected abstract*), konkrete Methode `addElement(title)` ruft `createElement`, registriert Element und legt es in `elements` ab; `render()` delegiert an alle Elemente.
- **mobileapplication.ui.AndroidForm / IOSForm** (konkrete Creator): überschreiben `createElement`, erzeugen jeweils passendes konkretes Element.
- **mobileapplication.ui.AndroidElement / IOSElement** (konkrete Product): besitzen Paket-Sicht-Konstruktor, damit Erzeugung nur über `MobileForm` möglich ist; implementieren `render()`.
- **mobileapplication.App**: Attribut `form: MobileForm`; `getOS()` liefert „Android“ oder „iOS“; `initialize()` legt passende konkrete Form an; `execute()` fügt drei Elemente hinzu und rendert.

Beziehungen: `MobileForm` ist abstrakter Creator und kennt die Factory Method `createElement`; konkrete Formen liefern passende konkrete Elemente. `addElement` kapselt die Erzeugung (Konstruktoren der konkreten Elemente sind paket-privat), sodass von außen keine widersprüchlichen Kombinationen angelegt werden können. `App` nutzt nur das abstrakte Interface und bleibt vom konkreten OS entkoppelt.

### b) Sequenzdiagramm (Android)

Gefundene Nachricht: `App.execute()`.

1. `execute()` ruft `initialize()`.
2. `initialize()` erzeugt `AndroidForm` und speichert sie in `form`.
3. Dreimal `form.addElement("...")`: Aufruf von `AndroidForm.createElement()`, Konstruktion new `AndroidElement(title)`, `register(form)`, Ablage in `elements`.
4. `form.render()`: iteriert über alle `AndroidElement`-Instanzen, die jeweils `render()` ausführen.

Die Nachrichten laufen zwischen den konkreten Klassen `App` → `AndroidForm` → `AndroidElement`; dank Factory Method entsteht keine Kreuzung Android/iOS.

### 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    void register(MobileForm form) { // Paket-sichtbar
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
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`.