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)

```
// mobileapplication/ui/MobileElement.java
package mobileapplication.ui;

public abstract class MobileElement {
    private final String title;
    private MobileForm form;

    protected MobileElement(String title) { // nur Paket/SUB-Klassen
```

```
        this.title = title;
    }

    public String getTitle() { return title; }

    void register(MobileForm form) { // Paket-sichtbar
        this.form = form;
    }

    public MobileForm getForm() { return form; }

    public abstract void render();
}

// mobileapplication/ui/MobileForm.java
package mobileapplication.ui;

import java.util.ArrayList;
import java.util.List;

public abstract class MobileForm {
    private final List<MobileElement> elements = new ArrayList<>();

    protected abstract MobileElement createElement(String title);

    public final void addElement(String title) {
        MobileElement element = createElement(title);
        element.register(this);
        elements.add(element);
    }

    public void render() {
        elements.forEach(MobileElement::render);
    }
}

// mobileapplication/ui/AndroidElement.java
package mobileapplication.ui;

class AndroidElement extends MobileElement {
    AndroidElement(String title) { super(title); }

    @Override
    public void render() {
        System.out.println("Android-Element: " + getTitle());
    }
}

// mobileapplication/ui/IOSElement.java
package mobileapplication.ui;

class IOSElement extends MobileElement {
    IOSElement(String title) { super(title); }
```

```
    @Override
    public void render() {
        System.out.println("iOS-Element: " + getTitle());
    }
}

// mobileapplication/ui/AndroidForm.java
package mobileapplication.ui;

public class AndroidForm extends MobileForm {
    @Override
    protected MobileElement createElement(String title) {
        return new AndroidElement(title);
    }
}

// mobileapplication/ui/IOSForm.java
package mobileapplication.ui;

public class IOSForm extends MobileForm {
    @Override
    protected MobileElement createElement(String title) {
        return new IOSElement(title);
    }
}

// mobileapplication/App.java
package mobileapplication;

import mobileapplication.ui.AndroidForm;
import mobileapplication.ui.IOSForm;
import mobileapplication.ui.MobileForm;

public class App {
    private MobileForm form;

    private String getOS() {
        return Math.random() > 0.5 ? "Android" : "iOS"; // Platzhalter
    }

    private void initialize() {
        String os = getOS();
        form = "Android".equals(os) ? new AndroidForm() : new IOSForm();
    }

    public void execute() {
        initialize();
        form.addElement("Header");
        form.addElement("Content");
        form.addElement("Footer");
        form.render();
    }
}
```

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`.