# 📓 Java Handout – Prüfungsvorbereitung (GUI & Sprachmittel)

---

## 1️⃣ Anonyme Klassen

**Was sind anonyme Klassen?**

- Klassen ohne Namen
- Implementieren ein Interface oder erweitern eine Klasse direkt im Code
- Besonders nützlich bei kleinen Helferklassen, z. B. Eventhandling

**Beispiel:**

```
Rechner addierer = new Rechner() {
    @Override
    public int berechne(int a, int b) {
        return a + b;
    }
};
```

**Vorteile:**

- Schnell einsetzbar ohne eigene Datei
- Lesbarkeit leidet bei komplexer Logik

---

## 2️⃣ Funktionale Interfaces

**Definition:**

- Interface mit genau **einer abstrakten Methode**
- Grundlage für Lambda-Ausdrücke
- Annotation `@FunctionalInterface` empfohlen

**Beispiel:**

```
@FunctionalInterface
interface Rechner {
    int berechne(int a, int b);
}
```

**Erlaubt zusätzlich:**

- `default`-Methoden (mit Implementierung)
- `static`-Methoden

**Typische Interfaces (java.util.function):**

- `Predicate<T>` → `boolean test(T t)`
- `Function<T,R>` → `R apply(T t)`
- `Consumer<T>` → `void accept(T t)`
- `Supplier<T>` → `T get()`

---

## 3 Lambda-Ausdrücke

**Lambda = Kurzform für anonyme Klassen**

**Struktur:**

```
(parameter) -> { Anweisungen }
```

**Beispiele:**

```
Rechner add = (a, b) -> a + b;

Consumer<String> drucker = s -> System.out.println(s);
```

**Komplexer:**

```
(a, b) -> {
    int sum = a + b;
    return sum;
}
```

**Vorteile:**

- Kürzere, lesbare Syntax
- Ideal für Callbacks, Streams, Eventhandling

---

## 4 Generics

**Definition:**

- Platzhalter für Typen
- Erlauben Typsicherheit und Wiederverwendbarkeit

**Generische Klasse:**

```
class Box<T> {
    private T inhalt;
```

```
    public void setInhalt(T i) { inhalt = i; }
    public T getInhalt() { return inhalt; }
}
```

**Generische Methode:**

```
public <T> void print(T element) {
    System.out.println(element);
}
```

**Typische Platzhalter:**

| Symbol | Bedeutung |
|---|---|
| `T` | Typ |
| `E` | Element |
| `K,V` | Key/Value |

---

## 5 Einführung in Swing

**Was ist Swing?**

- GUI-Toolkit für Java
- Teil von `javax.swing`

**Typische Komponenten:**

- `JFrame`, `JButton`, `JTextField`, `JLabel`, `JTextArea`, `JPanel`

**Beispiel:**

```
JFrame f = new JFrame("Fenster");
f.add(new JButton("Klick mich"));
f.setSize(300,200);
f.setVisible(true);
```

**LayoutManager (wichtige):**

- `FlowLayout`, `BorderLayout`, `GridLayout`, `BoxLayout`, `CardLayout`

---

## 6 GUI & MVC (Model View Controller)

**Trennung von:**

- **Model** – Datenhaltung und Logik

- **View** – GUI
- **Controller** – Eventhandling und Steuerung

**Struktur:**

```
└── src/
    ├── UserModel.java
    ├── UserView.java
    ├── UserController.java
```

**Beispiel:**

```
UserModel model = new UserModel();
UserView view = new UserView();
new UserController(model, view);
```

**Vorteile:**

- Klare Trennung
- Wartbarkeit & Erweiterbarkeit
- Wiederverwendbarkeit

---

## 7 GUI-Ergonomie & Barrierefreiheit

**Ergonomie-Ziele:**

- Konsistenz
- Lesbarkeit
- Fehlervermeidung
- Effizienz

**Barrierefreiheit bedeutet:**

- Bedienbarkeit per Tastatur
- Tooltips & Screenreader-Unterstützung
- Kontraste beachten

**Beispielcode:**

```
JLabel nameLabel = new JLabel("Name:");
nameLabel.setDisplayedMnemonic('N');
JTextField name = new JTextField();
nameLabel.setLabelFor(name);
name.setToolTipText("Geben Sie Ihren Namen ein");
```

**Normen:**

- WCAG 2.1
- DIN EN ISO 9241
- DGUV 215-450

---

☑ **Merksatz:**

> Mit Swing programmierst du bedienbare, strukturierte und wiederverwendbare GUIs mit modernen Sprachmitteln wie Lambda & Generics.