

# Woche 1 / Modul 404

**Objektbasiert programmieren nach Vorgabe**

# **Agenda**

## **Siehe Screen**

# Modulstruktur

- 319 Applikationen entwerfen und implementieren
- 404 **Objektbasiert programmieren nach Vorgabe**

## Später

- 216A Klassenbasiert (ohne Vererbung) implementieren
- 216B Objektorientiert (mit Vererbung) implementieren

# Modulidentifikation

Modulnummer	404
Kompetenz	Vorgabe interpretieren, objektbasiert mit einer Programmiersprache implementieren, dokumentieren und testen.
Lektionen	36
Beschreibung	<a href="#">404 auf Modulbaukasten.ch</a>

# **Modul Lernziele**

1. Aufgrund einer Vorgabe den Ablauf darstellen. (*UML Klassendiagramm*)
2. Eine Benutzerschnittstelle entwerfen und implementieren. (*Java Swing*)
3. Erforderliche Daten bestimmen und Datentypen festlegen. (*Java Variablen*)
4. Programmvorgabe unter Nutzung vorhandener Komponenten mit deren Eigenschaften und Methoden, sowie Operatoren und Kontrollstrukturen implementieren. (*Java Basics*)

## **Modul Lernziele 2**

5. Beim Programmieren vorgegebene Standards und Richtlinien einhalten, das Programm inline dokumentieren und dabei auf Wartbarkeit und Nachvollziehbarkeit achten. (*Code Style*)
6. Programm auf Einhaltung der Funktionalität testen, Fehler erkennen und beheben. (*Manuelles Testen*)

# Leistungsbeurteilungsvorgabe

## LB1 50%

### Prüfung

- Woche 5 / 90 Minuten
- Schriftlich + Praktisch am PC

 [details...](#)

## LB2 50%

### Projektarbeit

- Woche 5 bis 9
- Erstellen einer Applikation inkl. GUI

 [details...](#)



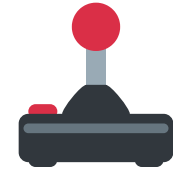
# Regeln

- Pünktlichkeit
- Aufmerksamkeit während des Unterrichts
- Selbständiges Arbeiten
- Internet als Arbeitsmittel





# Regeln / Zocken



**Wie wollen wir vorgehen, wenn ich jemand beim zocken erwische?**

# Heutige Ziele

- **Modulwebseite** kennen und anwenden
- Wissenslücken aus dem Modul 319 schliessen
- Unterschied zwischen **Klassen** und **Objekten** verstehen
- **Eigene Klassen** schreiben können
- **Objekte instanziiieren** können


# Die Modulwebseite





**Lektionen** Wochenfolien und die Quartalsübersicht

**Beurteilungen** Prüfungsrelevante Infos

**Repetition** Repetitionen vom Modul 319 / 403

 **Aufgaben Grundlagen** Alle Grunlegenden Aufgaben

 **Aufgaben Swing** (*GUI*) Alle Aufgaben bezüglich `Swing`

 **Konzepte** Allgemeine Konzepte Isoliert erklärt

 <https://codingluke.github.io/bbzbl-modul-404>



Die Aufgaben unter **Aufgaben Grundlagen / Swing** sind aufeinander aufbauend.

Ihr könnt zu Hause im **Selbststudium** alle Aufgaben lösen.

**Fragen?** 🙅



# Repetition Modul 319 (20 Minuten)



Repetition Lernkarten öffnen



Ihr erhaltet Lernkarten



Zweierteam bilden



Zufällige Karten ziehen und  
versuchen zu lösen



Korrekt gelöste Karten  
auf **einen Stapel**



Schwierige Karten  
auf einen **eigenen Stapel**

# Account Applikation (45 Minuten)



## Account Application öffnen

 Es handelt sich um eine Terminal Applikation

 Lesen und lösen Sie die Aufgabe

 Unterhalb existierte eine Musterlösung\*

 Die machen wir zusammen!

## Tipp

✓ Kennen sie noch den `Scanner` ?

✓ Diese Code-Snippets könnten hilfreich sein\*\*

```
Scanner sc = new Scanner(System.in);  
// liest double von der Commandline  
double amount = sc.nextDouble();  
// liest String von der Commandline  
String command = sc.nextString();
```

\* Musterlösung nur verwenden, wenn Sie nicht mehr weiter kommen! \*\* natürlich nicht genau so 😊



# Analyse Account Applikation (20 Minuten)



Analyse Account Applikation öffnen



Gehen Sie nach der Anleitung vor



Analysieren Sie die dargestellte Klasse **zu zweit**





# Input Fachklassen



- **Fachklassen** beinhalten die **generalisierte Logik** für ein Fachproblem
  - *In unserem Fall wäre das Fachproblem die Kontoverwaltung*
- Mit **Fachklassen** lassen sich **Programme entkoppeln**
  - **einfacherer Wartbarkeit**
  - **besserer Testbarkeit**
  - **besserer Qualität**
- Nennen wir es 🍣 **Sushi-Code**, das Gegenteil von 🍝 *Spaghetti-Code*



# Gemeinsame Analyse AccountApplication

```

public class AccountApplication {
    public static void main(String[] args) {
        System.out.println("Welcome to the account application");
        double kontostand = 0;
        double amount = 0;
        String command = "";
        do {
            Scanner sc = new Scanner(System.in);
            System.out.println("Please enter the amount, 0 (zero) to terminate");
            amount = sc.nextDouble();
            if (amount != 0) {
                System.out.println("To deposit, press +, to withdraw press -");
                command = sc.next();
                if (command.equals("+")) {
                    kontostand = einzahlen(kontostand, amount);
                } else if (command.equals("-")) {
                    kontostand = abheben(kontostand, amount);
                }
            }
        } while (amount != 0);
        System.out.println("Final balance: " + aktuellerKontostand(kontostand));
    }

    public static double einzahlen(double ks, double betrag) {
        return ks + betrag;
    }

    public static double abheben(double ks, double betrag) {
        return ks - betrag;
    }

    // diese Methode macht keinen Sinn
    // es ging bei der Aufgabe aber darum, wieder ein paar Methoden zu schreiben
    public static double aktuellerKontostand(double ks) {
        return ks;
    }
}

```

Diagram illustrating the code structure with numbered annotations (0-9) pointing to specific lines:

- 0: `System.out.println("Welcome to the account application");`
- 1: `Scanner sc = new Scanner(System.in);`
- 2: `System.out.println("Please enter the amount, 0 (zero) to terminate");`
- 3: `amount = sc.nextDouble();`
- 4: `kontostand = einzahlen(kontostand, amount);`
- 5: `return ks + betrag;`
- 6: `kontostand = abheben(kontostand, amount);`
- 7: `return ks - betrag;`
- 8: `System.out.println("Final balance: " + aktuellerKontostand(kontostand));`
- 9: `} while (amount != 0);`