Übergang blockbasierte zu textbasierte Programmierung

Mike Barkmin

Materialien

- `https://informatik.openpatch.org/`
- `https://scratch4j.openpatch.org/`
- Material-Paket (BlueJ-Projekten und Klausuren + Lernziele)

Überblick

- Kleine Umfrage
- Didaktische Bibliotheken für den Einstieg in die Programmierung mit Java
- Scratch for Java
  - Von Scratch zu Java — Ein Beispiel
  - Den Übergang gestalten
  - Beispiele aus der Unterrichtspraxis
  - Selbst ausprobieren
- Abschluss

Kleine Umfrage

Kleine Umfrage

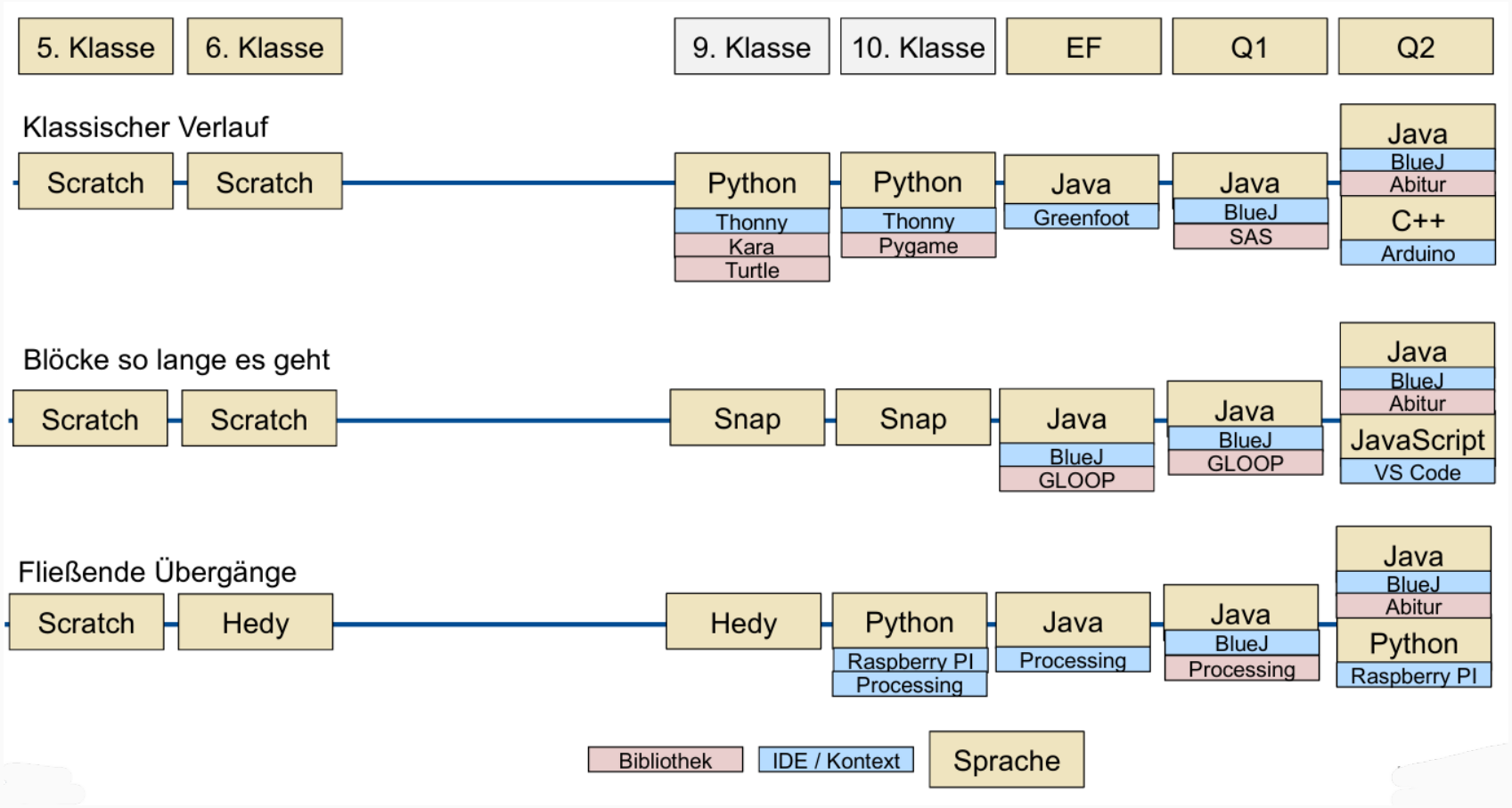
www.menti.com

Didaktische Bibliotheken für den Einstieg in die Programmierung mit Java

Didaktische Bibliotheken

- GLOOP: `https://www.schulentwicklung.nrw.de/cms/programmierung-mit-gloop/installation/index.html`
- Stifte und Mäuse: `https://www.mg-werl.de/sum/`
- Processing: `https://processing.org/`
- Greentfoot: `https://www.greenfoot.org/door`
- Shapes and Sprites: `http://www.dingemann.de/sas/`

Wege durch die Programmierung



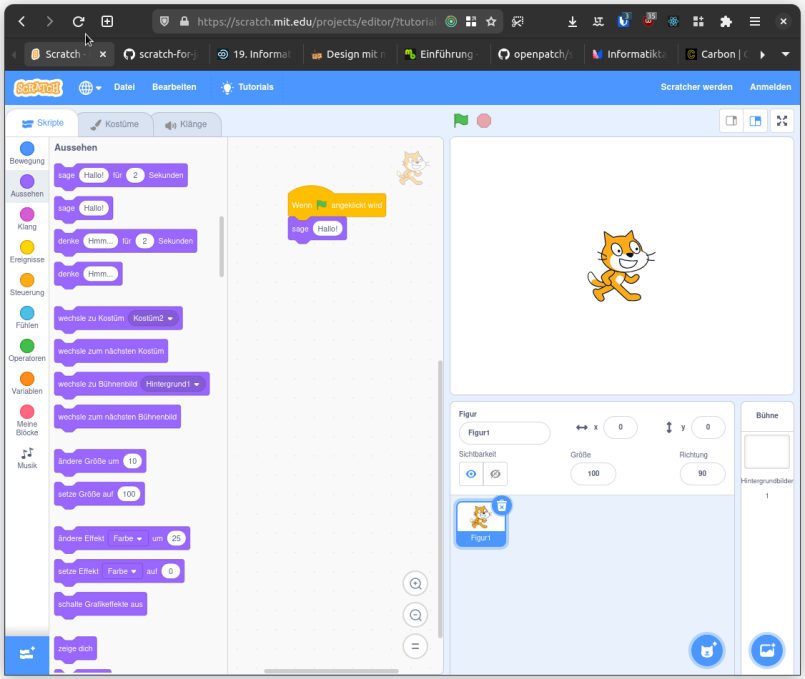
Scratch for Java

Links

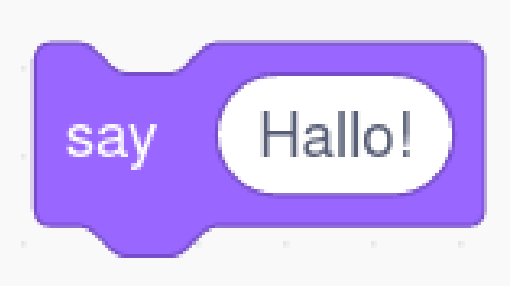
- Quelltext:
  `https://github.com/openpatch/scratch-for-java`
- Dokumentation:
  `http://scratch4j.openpatch.org/de`

Das Scratch-Modell

- Es gibt **eine** Bühne
  - Hintergründe
  - Klänge
  - Skripte
- Es gibt mehrere Figuren
  - Kostüme
  - Klänge
  - Skripte



Ziel von Scratch for Java




```
meineFigur.say("Hallo!");
```

- Das Scratch-Modell nachstellen
- Jeden Block in eine Methode überführen
- An passenden Stellen das Scratch-Modell erweitern

An Bekanntes anknüpfen!

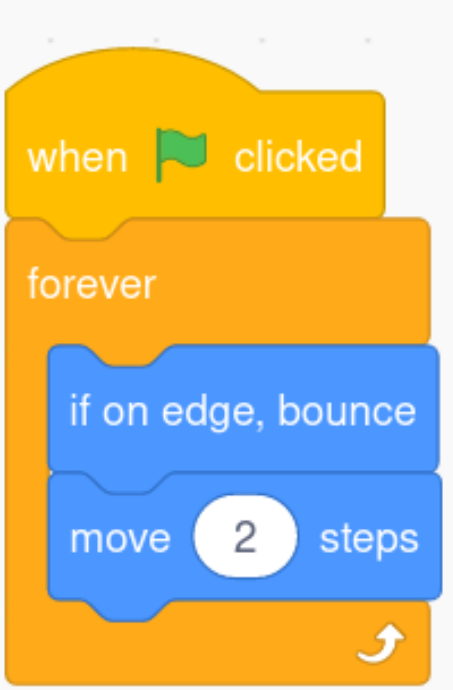
Einfaches Beispiel

Ziel: Eine Katze soll sich von links nach rechts bewegen.

LIVE DEMO

Scratch for Java — Scratch

```java
import org.openpatch.scratch.*;

class CatSprite extends Sprite {
    CatSprite() {
        this.addCostume("cat", "sprites/cat.png");
        this.setOnEdgeBounce(true);
    }

    public void run() {
        this.move(2);
    }
}

public class CatStage extends Stage {

    public CatStage() {
        this.add(new CatSprite());
    }
}
```



Design mit mehreren Ansätzen

- Imperativer Ansatz (Klassen-später)
- Objektorientierte Ansatz (Klassen-zuerst)
- Erweiterte Scratch Ansatz

Imperativer Ansatz (Klassen-später)

```
public class MyProgram {
    public MyProgram() {
        Stage myStage = new Stage();
        Sprite zebra = new Sprite();
        zebra.addCostume("walk_1", "assets/walk_1.png");
        zebra.setOnEdgeBounce(true);

        while(true) {
            zebra.move(1);
        }
    }
}
```

- \+ Nur die Klassen Sprite und Stage werden verwendet
- \+ Methoden sind den Schüler:innen durch Scratch bekannt
- \- Verhalten in einer globalen Klasse definiert

Objektorientierter Ansatz (Klassen-zuerst)

```
import org.openpatch.scratch.*;

class CatSprite extends Sprite {
    CatSprite() {
        this.addCostume("cat", "sprites/cat.png");
        this.setOnEdgeBounce(true);
    }

    public void run() {
        this.move(2);
    }
}

public class CatStage extends Stage {

    public CatStage() {
        this.add(new CatSprite());
    }
}
```

- \+ Näher am Scratch-Modell
- \+ Flexibler in der Implementierung
- \- Vererbung von Day 1

Vergleiche ziehen i

In Scratch haben wir unser Projekt vorbereitet, indem wir viel geklickt haben. Zum Beispiel haben wir für das Hinzufügen einer Figur das Katzen-Icon angeklickt oder wir haben ein neues Kostüm hinzugefügt, indem wir den Kostüm-Reiter ausgewählt haben und ein neues Bild hochgeladen haben. In Java bereiten wir unser Projekt vor, indem wir einen Konstruktor schreiben. Wir schreiben Text (Quelltext), welcher unser Geklicke ersetzt, da wir keine Benutzeroberfläche mehr haben.

Vergleiche ziehen ii

In Scratch hat jede Figur und die Bühne eine Menge von vordefinierte n Blöcken, welche wir benutzen konnten. In Java haben wir normalerweise nichts vordefiniertes und müssen alles selbst implementieren. Aber mit der Vererbung können wir auch vordefinierte Methoden (Blöcke) benutzen.

Vergleiche ziehen iii

scratch4j.openpatch.org/de/multiple-approach-design

Objektorientierte Ansatz (aka. Klassen-zuerst)

Dieser Ansatz ist dem Scratch-Modell am ähnlichsten. Um dies zu erreichen, muss direkt mit Vererbung gearbeitet werden. Auch wenn man jetzt vielleicht denkt, dass dieser Ansatz am Anfang zu komplex für Schüler sein könnte, kann man durch den direkten Vergleich von Scratch-Programmen und den Java-Programmen den Übergang gut moderieren.

Hier sind ein paar Vergleiche, die man verwenden könnte:

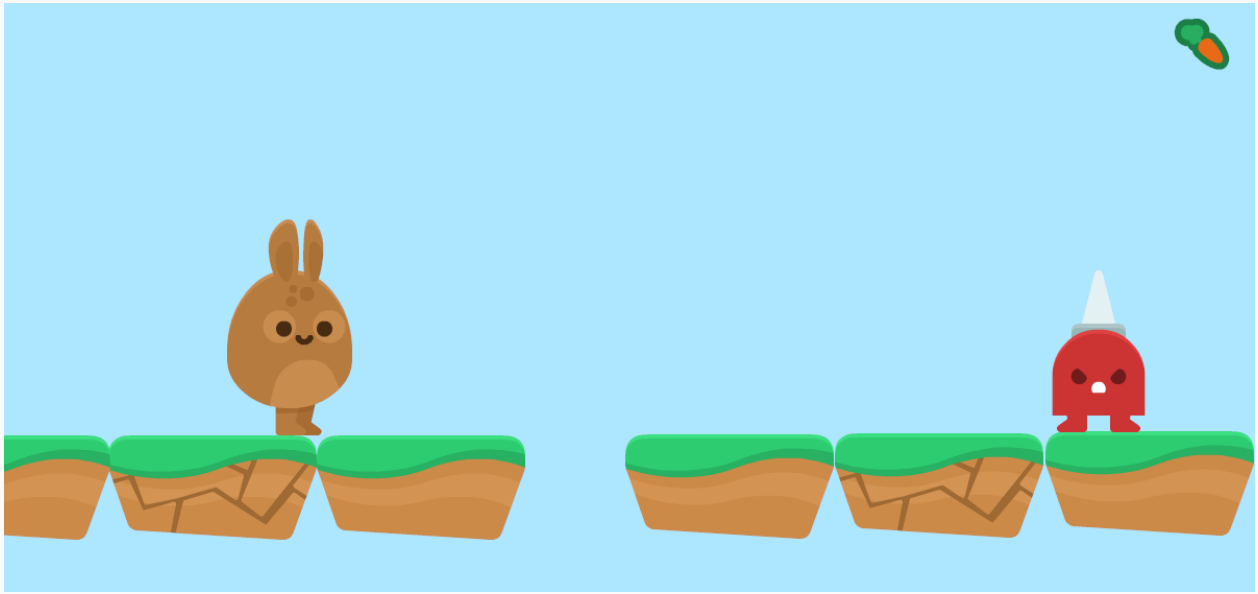
- In Scratch haben wir und Projekt vorbereitet indem wir viel geklickt haben. Zum Beispiel haben wir für das Hinzufügen einer Figur das Katzen-Icon angeklickt oder wir haben ein neues Kostüm hinzugefügt, indem wir den Kostüm-Reiter ausgewählt haben und ein neues Bild hochgeladen haben. In Java bereiten wir unser Projekt vor, indem wir einen Konstruktor schreiben. Wir schreiben Text (Quelltext), welcher unser Geklicke ersetzt, da wir keine Benutzeroberfläche mehr haben.
- In Scratch hat jede Figur und die Bühne eine Menge von vordefinierte n Blöcken, welche wir benutzen konnten. In Java haben wir normalerweise nichts vordefiniertes und müssen alles selbst implementieren. Aber mit der Vererbung können wir auch vordefinierte Methoden (Blöcke) benutzen.
- In Scratch haben wir unseren Figuren Namen gegeben, aber wir haben die Namen gar nicht so viel benutzt. In Java sind Namen viel wichtiger, da wir nicht mit einer Figur oder der Bühne weiterarbeiten können, wenn wir keinen Namen vergeben hanbe. `Sprite cat = new Sprite();`
- In Scratch haben wir eine Figur der Bühne hinzugefügt, indem wir eine erstellt haben. In Java müssen wir explizit sein und schreiben, dass wir eine bestimmte Figur der Bühne hinzufügen wollen.
  `myStage.add(mySprite);`
- In Scratch haben wir oft den warte-Block zum Pausieren der Ausführung eines Skripts von einer Figur benutzt. In Java können wir nicht mehr so einfach ein Skript pausieren. Außer wir wollen das gesamte Programm pausieren (`myStage.wait(100)`). Wenn wir so etwas ähnlich mit Scratch for Java erreichen wollen, dann müssen wir die Timer-Klasse

Erweiterte Scratch Ansatz

```
public class MyProject extends Window {
    public MyProject() {
        super(800, 600);
        Stage stage1 = new Stage();
        stage1.add(new Zebra());
        Stage stage2 = new Stage();

        this.addStage("first", stage1);
        this.addStage("second", stage2);

        this.switchStage("second");
    }
}

public class Zebra extends AnimatedSprite {
    public Zebra() {
        this.addAnimation("walk", "walk_%d.png", 4);
    }

    public void run() {
        this.playAnimation("walk");
    }
}
```

- Erweiterung des Objektorientierten Ansatzes
- Weitere Klasssen (Window, AnimatedSprite, Text, ...)
- Weicht vom Scratch-Modell ab

EF 2. Halbjahr: Bunny Hop



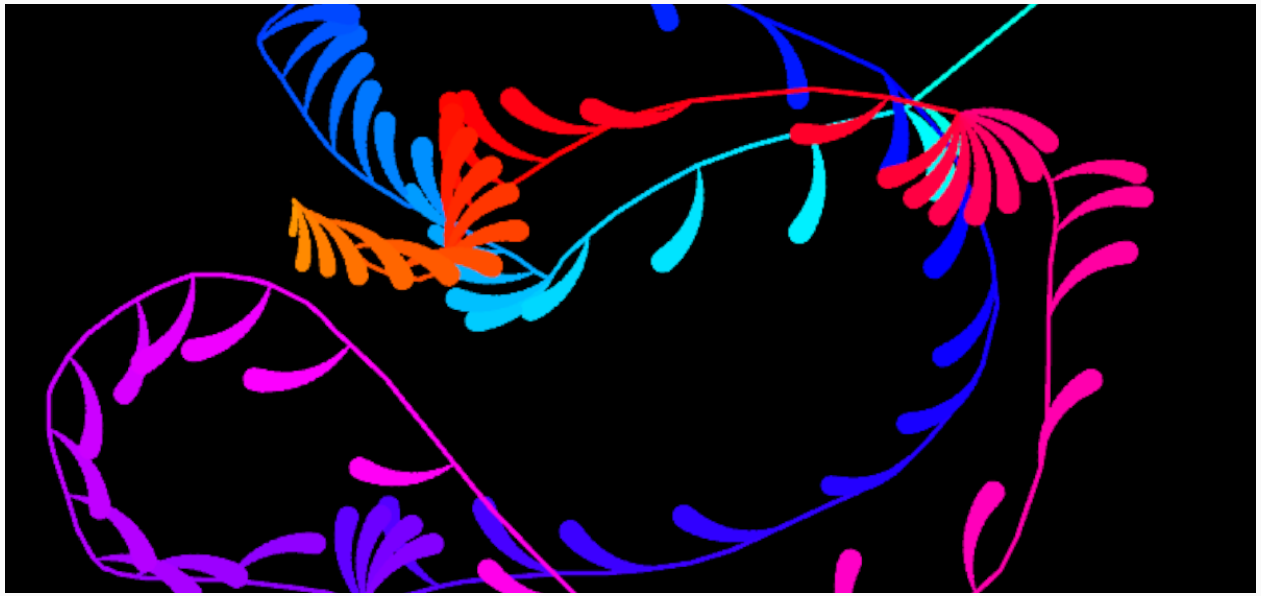
- Spieleentwicklung mit Scrum in Tandems
- `https://informatik.openpatch.org/projekte/bunny-hop`

EF 1. Halbjahr: RPG

- Videotutorials zur Programmierung eines RPGs
- Challenges und fließender Übergang in eigene Erweiterungen
- `https://informatik.openpatch.org/projekte/rpg`

- Analytischer Ansatz
- Einführung von Arrays
- `https://informatik.openpatch.org/projekte/rpg`

EF 1. Halbjahr: Generative Kunst (Planung)



- Kleinere Kunstwerke

Beispiele zum Ausprobieren

Beispiele zum Ausprobieren

https://kurzelinks.de/ap1k

- Beispiel auswählen
- Beispiel ausführen
- Beispiel verändern

Abschluss

Abschluss

- Fragen
- Verwendbarkeit in eurem Unterricht
- Anregungen zur Weiterentwicklung
- Blockbasiert zu textbasiert: Weicher oder harter Übergang?