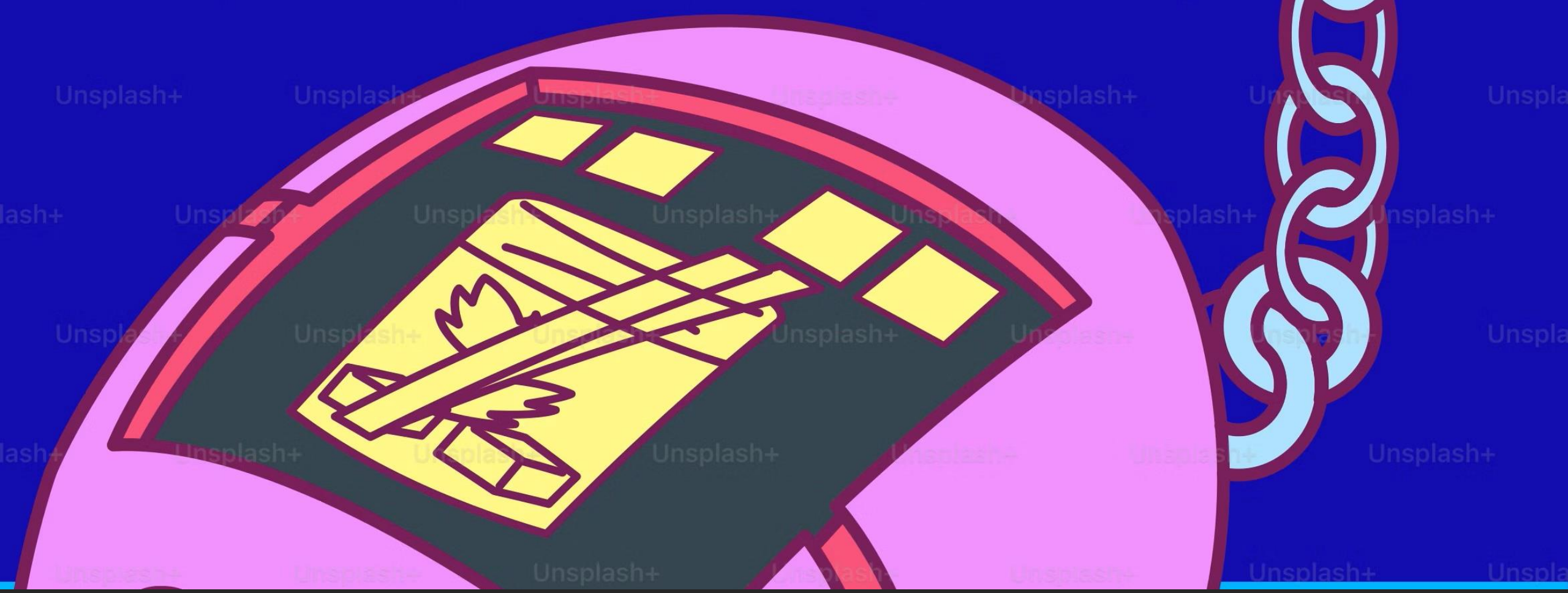


Tamagotchi – Tag 1

10. Dezember 2024

Plan für die Woche

Montag	Dienstag	Mittwoch	Donnerstag	Freitag
• Wiederholung	• Test 60min • JavaFX (Abriss)	• Exceptions in großen Projekten • Überladen von Methoden • Interfaces	• Strings • (Kalender API)	• Speichern und Laden

Plan für heute

- Exceptions in großen Projekten
- Überladen von Methoden
- Interfaces

Exceptions in großen Projekten

Aufgabe

Stelle dir vor du fängst bei einem Spielhersteller als Entwickler an. Du kennst das Projekt noch nicht, erhält aber direkt die Aufgabe fehlende Exceptions zu ergänzen. Dein Arbeitgeber weiß, dass du einiges über Exceptions gelernt hast und vertraut darauf, dass du dein Wissen umsetzen kannst.

1. Schaue dir in Ruhe die verschiedenen Klassen und den Projektaufbau an.
2. Finde alle TODOs.
3. Bearbeite die TODOs zum Exception-Handling. Beginne mit dem TODO in Zeile 140 *TamagotchiView.* (**Hinweis**: Insgesamt sind es vier TODOs.)
4. Findest du andere Stellen, wo eine Exception sinnvoll wäre?

Überladen von Methoden

(METHOD OVERLOADING)

Method Overloading

- innerhalb einer Klasse mehrere Methoden
  - mit dem gleichen Namen
  - Unterschiedlichen Parameterlisten
  - Gleichen Rückgabetypen
- Compiler unterschiede anhand der verschiedenen Parameterlisten
- Inhalt der Methoden kann/soll abweichen

Method Overloading

```java
class Beispiel {
    // Methode mit einem int-Parameter
    void drucke(int a) {
        System.out.println("Int: " + a);
    }

    // Überladene Methode mit einem String-Parameter
    void drucke(String s) {
        System.out.println("String: " + s);
    }

    public static void main(String[] args) {
        Beispiel obj = new Beispiel();
        obj.drucke(10);        // Ruft die Methode mit int-Parameter auf
        obj.drucke("Hallo");   // Ruft die Methode mit String-Parameter auf
    }
}
```

```java
class Beispiel {
    // Methode mit einem Parameter
    void drucke(int a) {
        System.out.println("Die Zahl ist: " + a);
    }

    // Überladene Methode mit zwei Parametern
    void drucke(int a, int b) {
        System.out.println("Die Zahlen sind: " + a + " und " + b);
    }

    public static void main(String[] args) {
        Beispiel obj = new Beispiel();
        obj.drucke(5);        // Ruft die Methode mit einem Parameter auf
        obj.drucke(5, 10);    // Ruft die Methode mit zwei Parametern auf
    }
}
```

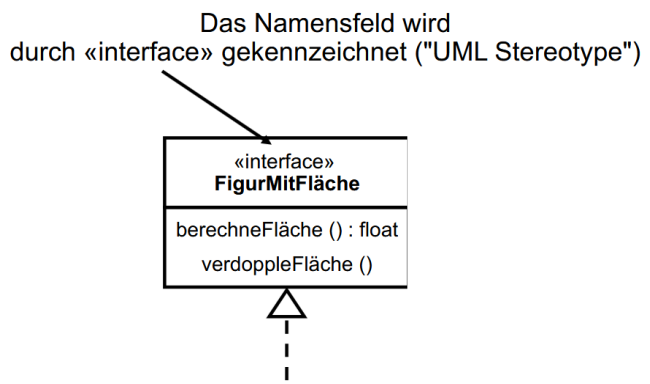
Aufgabe

Da du die erste Aufgabe so gut gemeistert hast bekommst du eine weitere. Jetzt sollst du die Methoden erweitern.

1. Bearbeite die TODO im *TamagotchiPresenter* Line 54.
2. An welchen anderen Stellen müssen Änderungen vorgenommen werden?

Interfaces

Interfaces



- Besondere Art von Klasse -> Schnittstellen
- Definieren Typen
- Beinhaltet nur Signaturen von Methoden und Konstanten
- Schafft grundlegende verpflichtende Strukturen in Subklassen
- Subklassen müssen alle Methoden implementieren (*implements*)
- Hier wir Mehrfachvererbung möglich -> Eine Klasse kann mehrere Interfaces implementieren

https://services.informatik.hs-mannheim.de/~schramm/tpe/files/Kapitel03.pdf

Interfaces

```java
interface Fahrzeug {
    // Konstante (wird automatisch als public static final behandelt)
    int MAX_GESCHWINDIGKEIT = 200;

    // Abstrakte Methode (wird automatisch als public abstract behandelt)
    void fahren();

    // Weitere abstrakte Methode
    void bremsen();
}
```

```java
class Auto implements Fahrzeug {
    // Implementierung der Methode "fahren"
    public void fahren() {
        System.out.println("Das Auto fährt.");
    }

    // Implementierung der Methode "bremsen"
    public void bremsen() {
        System.out.println("Das Auto bremst.");
    }
}
```

Default-Methoden in Interfaces

- Ab Java 8
- Implementierte Standartmethoden in Interfaces
- Müssen nicht zwingend Überschrieben werden

```
interface Fahrzeug {
    default void starten() {
        System.out.println("Das Fahrzeug startet.");
    }

    void fahren();  // Abstrakte Methode, muss von der Klasse implementiert werden
}
```

Static-Methoden in Interfaces

- Ab Java 8
- Interfaces können statische Methoden enthalten
- Methoden gehören nicht zur Instanz, sondern zur Klasse

```java
interface Fahrzeug {
    static void bremseTest() {
        System.out.println("Statische Methode im Fahrzeug-Interface.");
    }
}

public class Main {
    public static void main(String[] args) {
        Fahrzeug.bremseTest(); // Aufruf der statischen Methode des Interfaces
    }
}
```

Unterschied zu anderen Klassen

	Klassen	Abstrakte Klassen	Interfaces
Vererbungstiefe	Einzelvererbung (eine Klasse)	Einzelvererbung (eine Klasse)	Mehrfachvererbung (mehrere Interfaces)
Abstrakte Methoden	Nein	Ja	Ja (aber nur abstrakte Methoden, wenn keine Standardmethoden)
Konkrete Methoden	Ja	Ja	Ja (ab Java 8 Standardmethoden)
Konstruktoren	Ja	Ja	Nein
Instanzvariablen	Ja	Ja	Nein (nur Konstanten)
Ziel	Verwenden von geerbten Methoden und Eigenschaften	Gemeinsame Basis mit teilweiser Implementierung	Verhalten definieren, ohne Implementierung vorzuschreiben
Mehrfachvererbung	Nein	Nein	Ja

Weitere Klassenkonzepte

Konstrukt	Verwendung	Beispiel
final	- final bei Variablen: Konstanten - final bei Methoden: Nicht überschreibbar - final bei Klassen: Nicht vererbbar	final int MAX = 100;
static	- Statische Variablen und Methoden - Statischer Block für Initialisierung	static int count = 0;
Innere Klassen	- Nicht-statische: Bindung an Instanz der äußeren Klasse - Statische: Bindung an die Klasse - Lokale/Anonyme Klassen für spezifische Aufgaben	class Auto { class Motor { } }

Member Inner Class

Eine nicht-statische innere Klasse ist eine Klasse, die an eine Instanz der äußeren Klasse gebunden ist. Sie kann auf die Instanzvariablen und -methoden der äußeren Klasse zugreifen.

```java
class Auto {
    private String marke = "BMW";

    class Motor {
        void starten() {
            System.out.println("Der Motor der Marke " + marke + " startet.");
        }
    }
}

public class Main {
    public static void main(String[] args) {
        Auto auto = new Auto();
        Auto.Motor motor = auto.new Motor();  // Instanz der inneren Klasse
        motor.starten();  // Ausgabe: Der Motor der Marke BMW startet.
    }
}
```

Static Nested Class

Eine statische innere Klasse ist an die äußere Klasse gebunden, jedoch **nicht an eine Instanz** der äußeren Klasse. Sie kann nur auf statische Mitglieder der äußeren Klasse zugreifen.

```java
class Auto {
    private static String marke = "BMW";

    static class Motor {
        void starten() {
            System.out.println("Der Motor der Marke " + marke + " startet.");
        }
    }
}

public class Main {
    public static void main(String[] args) {
        Auto.Motor motor = new Auto.Motor();  // Keine Instanz von Auto erforderlich
        motor.starten();  // Ausgabe: Der Motor der Marke BMW startet.
    }
}
```

Lokale Innere Klassen

Eine lokale innere Klasse ist innerhalb einer Methode oder eines Blocks definiert und hat nur in diesem Kontext Gültigkeit.

```java
class Auto {
    void fahre() {
        class Motor {
            void starten() {
                System.out.println("Der Motor startet.");
            }
        }

        Motor motor = new Motor();  // Nur innerhalb der Methode sichtbar
        motor.starten();
    }
}

public class Main {
    public static void main(String[] args) {
        Auto auto = new Auto();
        auto.fahre();  // Ausgabe: Der Motor startet.
    }
}
```

Anonyme Innere Klassen

Eine anonyme innere Klasse ist eine Klasse ohne Namen, die in einer einzigen Zeile definiert wird. Sie wird häufig verwendet, wenn eine Instanz einer Schnittstelle oder abstrakten Klasse erforderlich ist.

```
interface Fahrer {
    void fahren();
}

public class Main {
    public static void main(String[] args) {
        Fahrer fahrer = new Fahrer() {
            public void fahren() {
                System.out.println("Der Fahrer fährt.");
            }
        };
        fahrer.fahren();  // Ausgabe: Der Fahrer fährt.
    }
}
```

Aufgabe

Das Tamagotchi soll weitere Tiere und verschiedenes Essen zur Verfügung stellen. Überlege dir ein Konzept zur Umsetzung und begründe kurz, wieso du dich dafür entschieden hast.

1. Bearbeite die letzten TODOs.
2. An welchen anderen Stellen müssen Änderungen vorgenommen werden? Was muss ergänzt werden?

Quellen

https://unsplash.com/de/grafiken/ein-rosafarbenes-objekt-an-dem-eine-kette-befestigt-ist-0adN6t6_zzs

https://services.informatik.hs-mannheim.de/~schramm/tpe/files/Kapitel03.pdf

https://oer-informatik.de/uml-klassendiagramm-interface-abstraktion