

Tamagotchi – Tag 4

12. Dezember 2024

Plan für die Woche

Montag

- Wiederholung

Dienstag

- Test 60min
- JavaFX (Abriss)

Mittwoch

- Exceptions in großen Projekten

Donnerstag

- Überladen von Methoden
- Interfaces
- Strings

Freitag

- Speichern und Laden

Plan für heute

- Überladen von Methoden
- Interfaces
- Strings

Überladen von Methoden

(METHOD OVERLOADING)

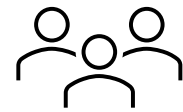
Method Overloading

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

Method Overloading

```
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  
    }  
}
```

```
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.

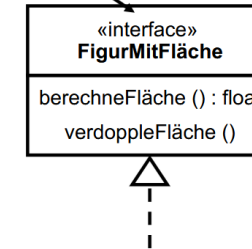
1. Bearbeite die TODO im *TamagotchiPresenter* Line 54.
2. An welchen anderen Stellen müssen Änderungen vorgenommen werden?

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 wird Mehrfachvererbung möglich -> Eine Klasse kann mehrere Interfaces implementieren

Das Namensfeld wird durch «interface» gekennzeichnet ("UML Stereotype")



Interfaces

```
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();  
}
```

```
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 bremsst.");  
    }  
}
```

Default-Methoden in Interfaces

- Ab Java 8
- Implementierte Standardmethoden 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

```
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	<ul style="list-style-type: none">- final bei Variablen: Konstanten- final bei Methoden: Nicht überschreibbar- final bei Klassen: Nicht vererbbar	<code>final int MAX = 100;</code>
static	<ul style="list-style-type: none">- Statische Variablen und Methoden- Statischer Block für Initialisierung	<code>static int count = 0;</code>
Innere Klassen	<ul style="list-style-type: none">- Nicht-statische: Bindung an Instanz der äußeren Klasse- Statische: Bindung an die Klasse- Lokale/Anonyme Klassen für spezifische Aufgaben	<code>class Auto { class Motor { } }</code>

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.

```
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.

```
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

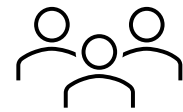
Eine lokale innere Klasse ist innerhalb einer Methode oder eines Blocks definiert und hat nur in diesem Kontext Gültigkeit.

```
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.  
    }  
}
```

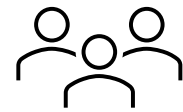


Aufgabe

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?

Strings



Selbststudium

Erarbeite dir in *Deshmukh (2024):Java Foundations Exam 1Z0-811* das Kapitel 13: *Working with the String Class*.

1. Erstelle dir eine A4 Seite mit einer Zusammenfassung und den wichtigsten Informationen zu dem Thema.
2. Löse die Aufgaben in 13.3 Exercise.
3. **Zusatz:** Finde zwei Stellen in unserem Tamagotchi-Code, an denen man die Konzepte anwenden kann.

Quellen

https://unsplash.com/de/grafiken/ein-rosafarbenes-objekt-an-dem-eine-kette-befestigt-ist-OadN6t6_zzs

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

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