Studienleistung Programmieren 2 Teil 2

Recherche und Definitionen

Was bedeutet Polymorphie?

Polymorphie ist ein Konzept der Objektorientierten Programmierung, das ermöglicht, dass ein Bezeichner abhängig von seiner Verwendung unterschiedliche Datentypen annimmt

Wie unterscheiden sich statische und dynamische Bindung?

- static: der Typ eines Objekts wird zur Kompilierung festgelegt
- dynamic: der Typ eines Objekts wird zur Laufzeit festgelegt

Was ist das Liskovsche Substitutionsprinzip?

Ein Programm das Objekte einer Basisklasse verwendet, muss auch mit Objekten einer davon abgeleiteten Klasse funktionieren, ohne dass das Programm dafür geändert werden muss.

Wie unterscheiden sich abstrakte Klassen und Interfaces?

- Interfaces können keinen Zustand oder Implementation haben.
- Eine Klasse die ein Interface implementiert, muss alle Methoden des Interfaces implementieren.
- Abstrakte Klassen können Zustände und Methodenimplementationen enthalten.
- Abstrakte Klassen können vererbt werden, ohne dass alle abstrakten Methoden implementiert werden müssen.
- Interfaces können mehrfach vererbt werden, abstrakte Klassen nicht.

Was sind Arrays variabler Länge? Kennen C#/Java solche Arrays? Gibt es alternativen? Finden sie min. 2 Sprachen, die Arrays variabler Länge unterstützen.

Die Länge eines solchen Arrays wird zur Laufzeit festgelegt. In C#/Java gibt es ArrayLists. Typisierung: Welche Art von Typisierung gibt es? Wie sind Java und C# einzuordnen?

- Starke/Schwache Typisierung
- Dynamische/Statische Typisierung
- Explizite/Implizite Typisierung

Was ist das Diamond-Problem? Welche Möglichkeiten gibt es die entstehenden Probleme zu vermeiden? Gehen sie exenplarisch auf C++, C# und Java ein.

Das Diamond-Problem entsteht durch Mehrfachvererbung in der Objektorientierten Programmierung. Es kann auftreten, wenn eine Klasse D auf zwei verschiedenen Vererbungspfaden (B und C) von der selben Basisklasse A abstammt.

Normale Mehrfachvererbung in C++

```
class A {
   int a;
};

class B: A {
   int b;
};

class C: A {
   int c;
};

class D: B, C {
   int d;
};
```

Diamond-Vererbung in C++

```
class A {
   int a;
};

class B: virtual A {
   int b;
};
```

```
class C: virtual A {
   int c;
};

class D: B, C {
   int d;
};
```

Teil A

---

siehe e3

Teil B

---

siehe e2

Diskussion und Bewertung

Wann ist der Einsatz eines Interfaces sinnvoll?

Wenn ich viele verschiedene Klassen implementiere, wie z.B. verschiedene Körper (Würfel, Kugel...), so bietet es sich an gemeinsame Eigenschaften in einem Interface zu definieren. So lassen sich z.B. Volumen getVolume() oder Gewicht getWeight() mit einer einheitlichen Methode ausgeben, auch wenn die Implementierung in den verschiedenen Klassen unterschiedlich ist.

Was sind Mixins, wie funktionieren diese?

Ein Mixin ist ein spezieller Fall mehrfacher Vererbung. Die zwei häufigsten Fälle in denen man Mixins nutzt sind

1. Ich möchte einer Klasse viele optionale Features hinzufügen
2. Ich möchte ein Feature in vielen verschiedenen Klassen nutzen

Ein Persistenzmixin zum speichern/lesen einer Datenbank ist ein gutes Beispiel. Möglicherweise hat man mehrere

Klassen, welche ihren Zustand/Attribute nicht-flüchtig speichern wollen. Anstatt nun für jede Klasse eine speicher/lese-Methode zu implementieren, nutzt man das Persistenzmixin in den einzelnen Klassen.