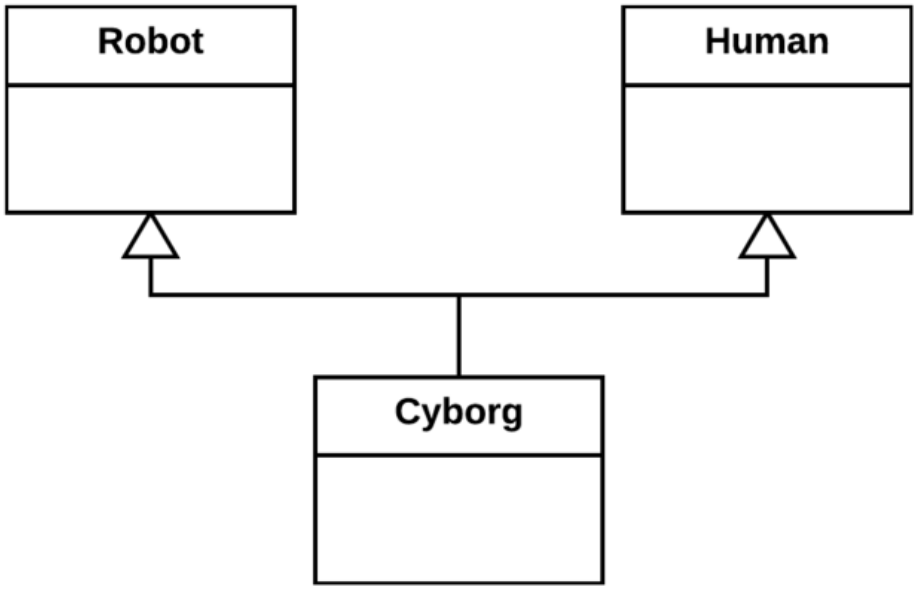
Interfaces

openHPI-Java-Team

Hasso-Plattner-Institut

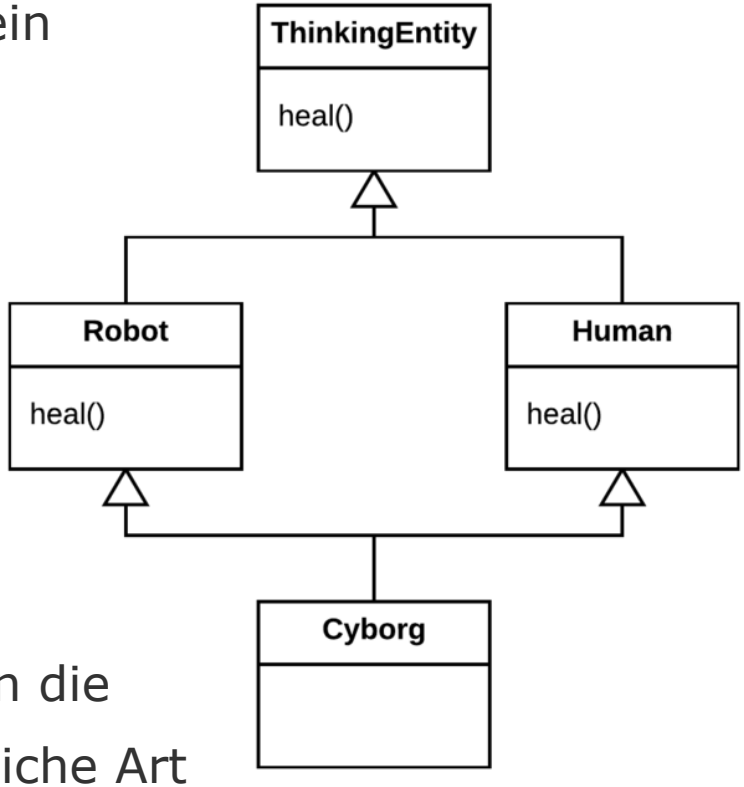
Prinzip der Mehrfachvererbung

- Eine Subklasse erbt aus (potentiell) verschiedenen Vererbungshierarchien
- Erscheint manchmal auf den ersten Blick als vorteilhaft



Das Diamond-Problem

- ABER: Mehrfachvererbung verursacht ein großes Problem:



- Die Klassen `Human` und `Robot` definieren die Methode `heal()` auf unterschiedliche Art
- **Welche der Implementierungen erbt der `Cyborg`, wenn `heal()` nicht neu definiert wird?**

→ **Java unterstützt keine Mehrfachvererbung**

Die Lösung: Interfaces

- Warum wollten wir Mehrfachvererbung haben?

➔ Es sollte sichergestellt werden, dass sowohl die Methoden der Klasse `Human` als auch die Methoden der Klasse `Robot` vorhanden sind.

- Interfaces
  - Stellen einen „Vertrag" zur Verfügung, den die implementierende Klasse einhalten muss.
  - Jeder Entwickler kann sich darauf verlassen, dass jede Klasse die ein Interface implementiert auch deren Methoden implementiert.
  - Sind sozusagen 100% abstrakte Klassen ➔ Sie beinhalten ausschließlich abstrakte Methoden

Die Lösung: Interfaces

- Syntax Definition:

```
interface Healable {  //Konvention: -able
    /* public abstract*/ int heal();
    /*Alle Methoden in Interfaces müssen public und
    abstract sein daher kann man die Modifikatoren
    weglassen.*/
}
```

- Syntax Nutzung:
- `class Cyborg implements Healable { ... }`
- `class <Klassenbezeichner> implements <InterfaceBezeichner>`
- Eine **nicht-abstrakte** Klasse die ein Interface implementiert, muss alle Methoden die im Interface deklariert wurden implementieren.

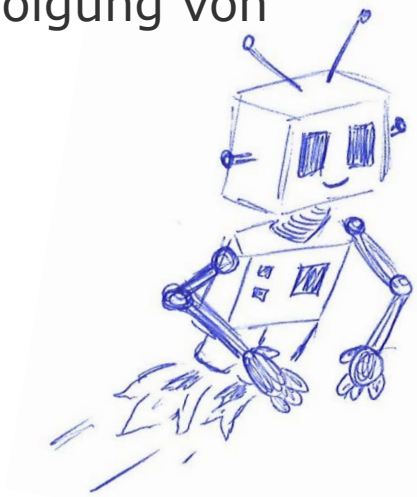
Wozu brauche ich das?
Ein Beispiel (1/4)

- Mal angenommen:
  - Paco schafft es sich aus der Zentrale von Eike Vil zu befreien.
  - Ronja hat kürzlich ein Upgrade erhalten und kann nun auch fliegen.
  - Eike Vil hat einen SuperVillain-Umhang im Schrank und kann damit auch fliegen.

➔ Alle können sich eine Verfolgungsjagd in der Luft liefern

Wozu brauche ich das?
Ein Beispiel (2/4)

- Paco flieht.
- Ronja versucht Paco einzuholen und mit ihm zu Detektiv Duke zu fliegen.
- Eike Vil nimmt die Verfolgung von Paco und Ronja auf.



Wozu brauche ich das?
Ein Beispiel (3/4)

- Um die Verfolgungsjagd zu implementieren,
  - müssen wir sicher sein, dass auch wirklich alle fliegen können,
  - wäre es vorteilhaft alle drei **in eine Collection** stecken zu können, da wir dann bequem über die einzelnen Charaktere iterieren können.
    ➔ Dafür brauchen aber alle Charaktere denselben Typ.
- Leider stecken alle drei Charaktere aber in unterschiedlichen Vererbungshierarchien.

Wozu brauche ich das?
Ein Beispiel (4/4)

- Die Lösung: alle drei implementieren dasselbe Interface **Flyable**
  - `class Parrot implements Flyable { ... }`
  - `class Robot implements Flyable { ... }`
  - `class SuperVillain implements Flyable { ... }`

➔ Alle drei Klassen besitzen nun zusätzlich zu den Typen aus ihrer jeweiligen Vererbungshierarchie auch noch den Typ **Flyable**

➔ Sie können daher alle drei in eine Collection die **Flyable**s aufnimmt eingefügt werden.

  - `LinkedList <Flyable> theChase = new LinkedList <> ();`
  - `Robot ronja =  new Robot();`
  - `Parrot paco = new Parrot(); ...`
  - `theChase.add(paco); ...`
- **Codebeispiel zum Experimentieren in CodeOcean.**

Interfaces

- Jede Klasse kann **beliebig viele** Interfaces implementieren
- Jede **nicht-abstrakte Klasse,** die ein Interface implementiert, muss alle deren Methoden **implementieren**
- Abstrakte Klassen können ebenfalls Interfaces implementieren
  - Die Implementierung der Methoden aus dem Interface kann an die erste nicht-abstrakte Subklasse weitergereicht werden.
- Jede Klasse kann **nur eine Elternklasse** erweitern, daneben aber auch Interfaces implementieren.
  - `class SuperVillain extends Human implements Flyable, Swimmable {...}`
- Interfaces können sich **gegenseitig erweitern**.
  - `interface Flyable extends Moveable {...}`

Anwendungsbeispiele für Interfaces

- Einfordern von Verhaltensweisen
- Konstantenlisten (Mehr Details im Deep Dive)
- Beispiele aus der Java API:
  - Iterable
  - Comparable
  - ...