LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Lehrstuhl Bioinformatik • Konstantin Pelz

Erste Java-Programme

(Java Wiederholung & Vererbung)

Tutorium Bioinformatik
(WS 18/19)

Konstantin: Konstantin.pelz@campus.lmu.de
Homepage: https://bioinformatik-muenchen.com/studium/propaedeutikum-programmierung-in-der-bioinformatik/

Organisatorisches

!!! NICHT VERGESSEN !!!

Bedarfsermittlung für das PBL (bis 14.01.19)

- 07.01.19-14.01.19: Bedarfserfassung
- 21.01.19-28.01.19: Verbindliche Anmeldung
- voraus. ab 31.01.19: Mitteilung der Einteilung durch die Veranstalter und ggf. Informationen zum ersten Termin

Link:
https://www.bio.ifi.lmu.de/studium/seminare-praktika/

Wiederholung Java

- Datentypen
- If/Else/If Else
- Arrays
- Exceptions
- Schleifen (for/while)
- Scope
- RegEx

Wiederholung Java

- Datentypen
  - z.B. int, double, String
- If/Else/If Else
  - Fallunterscheidung, Alternative ist Switch-Case
- Arrays
  - Festgelegte Größe, Speicherung mit Indices, Beginnt bei 0!
- Exceptions
  - GidF, meist selbsterklärend, nutzt verschiedenes Debugging
- Schleifen (for/while)
  - Entscheidung welche besser zum Problem passt, Bei while: Vorsicht vor endlosen Schleifen
- Scope
  - Sichtbarkeit, bei vielen ähnlichen Variablen immer prüfen, ob man mit der richtigen arbeitet
- RegEx
  - Reguläre Ausdrücke, Gut um Muster zu beschreiben oder zu finden

Wiederholung Java

- Rekursion vs. Iteration

- Klassen

- Objekte

- Methoden

- Statische Variablen und Methoden

- Überladung von Methoden

Wiederholung Java

- Rekursion vs. Iteration
  - Wiederhol. Aufruf von sich selbst vs. „normal“ programmiert
- Klassen
  - Strukur von Objekten, immer Konstruktor und Methoden, Importierbar von schon bestehenden libraries
- Objekte
  - Abbildungen von Klassen, können viele Eigenschaften haben
- Methoden
  - Sehr unterschiedlich, Rückgabetyp oder void, Eingabeparameter oder keine, statisch oder nicht
- Statische Variablen und Methoden
  - Kein Objekt nötig um sie aufzurufen, nur EINE Instanz pro Klasse
- Überladung von Methoden
  - Gleicher Methodenkopf, unterschiedlicher Methodenrumpf

Vererbung

Motivation: Wiederverwendbarkeit der Klassen

- Klasse A erbt von Klasse B

- Klasse A kann Methoden und Eigenschaften von Klasse B nutzen

- Klasse A kann Methoden von Klasse B überschreiben oder erweitern

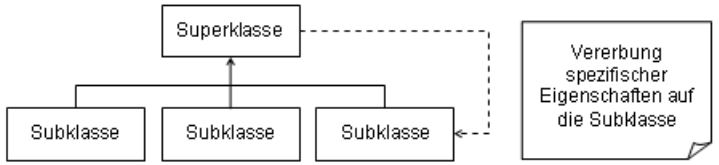
- Klasse A kann weitere Methoden und Eigenschaften definieren

Beispiel:

Student ist eine Person. Professor ist eine Person. Gemeinsame Eigenschaften und Methoden, die sowohl Student wie auch Professor haben, können einmalig in Person geschrieben werden. Student und Professor greifen dann auf Person zu. (Vererbung)

Vererbung

Allgemein



in Java:

- eine Klasse kann nur von einer anderen Klasse erben
- Von einer Klasse, können beliebig viele Klassen erben

Java Code Beispiel:

```
public class SuperClass {
  //CODE
}
```

```
public class SubClass extends SuperClass {
  //CODE
}
```

Vererbung

- **Überladung: Zwei Methoden haben gleichen Namen, aber verschiedene Parameter. Aufruf wird zur Compilezeit unterschieden.**

- **Überschreiben: Zwei Methoden mit gleichem Namen und gleichen Parametern, aber eine davon in Basisklasse und eine in abgeleiteter Klasse. Erst zur Laufzeit wird Typ des Objekts geprüft und die richtige Methode ausgewählt.**

```java
public class SuperClass {
    public void test(int a) {
        System.out.println(a + 1);
    }
}
```

```java
public class SubClass extends SuperClass {
    public void test(int a) {
        System.out.println(a - 1);
    }
}
```

Vererbung

- **Methodendefinition in der abgeleiteten Klasse kann auf Definition in der Basisklasse zugreifen.**

- **Methodenaufruf super.methode(...) geht die Klassenhierarchie durch und sucht zuletzt überschriebene Definition von methode(...).**

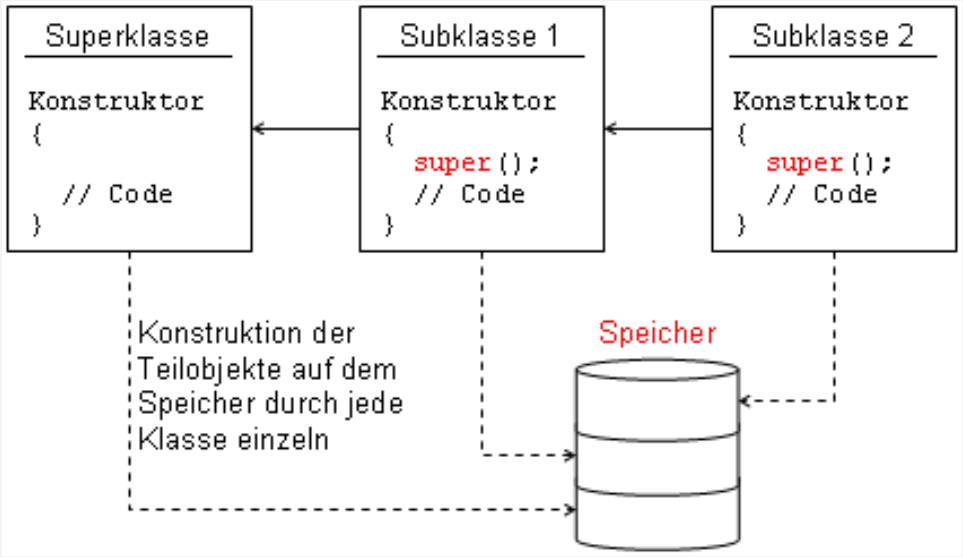
- **super allein ist kein gültiger Ausdruck!**

```
public class SuperClass {
    public void test(int a) {
        System.out.println(a + 1);
    }
    public void test2(int a){
        System.out.println(a+a);
    }
}
```

```
public class SubClass extends SuperClass {
    public void test(int a) {
        test2(a);
        super.test(a);
    }
}
```

Vererbung

- Konstruktoren werden nicht vererbt.
- Konstruktoren der Basisklasse können mit super(...) aufgerufen werden.
- Falls Basisklasse mehrere Konstruktoren hat, ruft super(...) den mit den richtigen Parametertypen auf.



Vererbung

- Wenn bei einer Klassendefinition nicht explizit ”extends ...” dabeisteht, wird sie implizit von Object (in java.lang) abgeleitet.

- Wichtigste Methoden (zum Überschreiben):
  - boolean equals(Object o):
    Bin ich gleich wie Objekt o?

```java
public class SuperClass {
    private int s = 4;

    public int getS() {
        return s;
    }

    public boolean equals(Object obj) {
        //Is it even the same object?
        if (!(obj instanceof SuperClass)) {
            return false;
        }
        //Is it the same object. So are the values the same?
        return this.s == ((SuperClass) obj).getS();
    }
}
```

Vererbung

- Wenn bei einer Klassendefinition nicht explizit ”extends ...” dabeisteht, wird sie implizit von Object (in java.lang) abgeleitet.

- Wichtigste Methoden (zum Überschreiben):
  - String toString():
    erzeuge eine String- Repräsentation von mir. Wird automatisch bei +-Operation für Strings aufgerufen.

```
public class SuperClass {
    private int s = 4;

    public int getS() {
        return s;
    }

    public String toString() {
        // Giving all information in form of String
        return "s value:" + s;
    }
}
```

Vererbung

Beispiel für eine Vererbung
Superklasse Sequence und zwei Subklassen AminoSequence und MrnaSequence

```java
public class Sequence {
    private int id;
    private String seq;

    public Sequence(int id, String seq) {
        this.id = id;
        this.seq = seq;
    }

    public int getId() {
        return id;
    }

    public String getSequence() {
        return seq;
    }

    public int getLength() {
        return seq.length();
    }

    public static String getFasta(Sequence seq) {
        String fasta = "";
        // Constructing Fasta out of ID and Sequence
        return fasta;
    }
}
```

```java
public class AminoSequence extends Sequence {

    public AminoSequence(int id, String seq) {
        super(id, seq);
    }

    public int getMass() {
        int mass = 0;
        // Code for adding up mass of each amino
        return mass;
    }
}
```

```java
public class MrnaSequence extends Sequence {

    public MrnaSequence(int id, String seq) {
        super(id, seq);
    }

    public String translate() {
        String translated = "";
        // Translate MRNA to AminoAcid Sequence
        return translated;
    }
}
```

Vererbung

Beispiel für eine Vererbung

```java
public class Sequence {
    private int id;
    private String seq;

    public Sequence(int id, String seq) {
        this.id = id;
        this.seq = seq;
    }

    public int getId() {
        return id;
    }

    public String getSequence() {
        return seq;
    }

    public int getLength() {
        return seq.length();
    }

    public static String getFasta(Sequence seq) {
        String fasta = "";
        // Constructing Fasta out of ID and Sequence
        return fasta;
    }
}
```

```java
public class AminoSequence extends Sequence {

    public AminoSequence(int id, String seq) {
        super(id, seq);
    }

    public int getMass() {
        int mass = 0;
        // Code for adding up mass of each amino
        return mass;
    }
}
```

```java
public static void main(String[] args) {
    AminoSequence as = new AminoSequence(0, "AMH");
    System.out.println(as.getSequence());
    System.out.println(as.getMass());
    System.out.println(Sequence.getFasta(as));
}
```

Main ruft Methoden aus der direkten Subklasse oder der Superklasse

Vererbung

Beispiel für eine Vererbung

```java
public class Sequence {
    private int id;
    private String seq;

    public Sequence(int id, String seq) {
        this.id = id;
        this.seq = seq;
    }

    public int getId() {
        return id;
    }

    public String getSequence() {
        return seq;
    }

    public int getLength() {
        return seq.length();
    }

    public static String getFasta(Sequence seq) {
        String fasta = "";
        // Constructing Fasta out of ID and Sequence
        return fasta;
    }
}
```

```java
public class AminoSequence extends Sequence {

    public AminoSequence(int id, String seq) {
        super(id, seq);
    }

    public int getMass() {
        int mass = 0;
        // Code for adding up mass of each amino
        return mass;
    }
}
```

```java
public static void main(String[] args) {
    AminoSequence as = new AminoSequence(0, "AMH");
    System.out.println(as.getSequence());
    System.out.println(as.getMass());
    System.out.println(Sequence.getFasta(as));
}
```

Main ruft Methoden aus der direkten Subklasse oder der Superklasse

Vererbung

Beispiel für eine Vererbung

```java
public class Sequence {
    private int id;
    private String seq;

    public Sequence(int id, String seq) {
        this.id = id;
        this.seq = seq;
    }

    public int getId() {
        return id;
    }

    public String getSequence() {
        return seq;
    }

    public int getLength() {
        return seq.length();
    }

    public static String getFasta(Sequence seq) {
        String fasta = "";
        // Constructing Fasta out of ID and Sequence
        return fasta;
    }
}
```

```java
public class AminoSequence extends Sequence {

    public AminoSequence(int id, String seq) {
        super(id, seq);
    }

    public int getMass() {
        int mass = 0;
        // Code for adding up mass of each amino
        return mass;
    }
}
```

```java
public static void main(String[] args) {
    AminoSequence as = new AminoSequence(0, "AMH");
    System.out.println(as.getSequence());
    System.out.println(as.getMass());
    System.out.println(Sequence.getFasta(as));
}
```

Main ruft Methoden aus der direkten Subklasse oder der Superklasse

Vererbung

Beispiel für eine Vererbung

```java
public class Sequence {
    private int id;
    private String seq;

    public Sequence(int id, String seq) {
        this.id = id;
        this.seq = seq;
    }

    public int getId() {
        return id;
    }

    public String getSequence() {
        return seq;
    }

    public int getLength() {
        return seq.length();
    }

    public static String getFasta(Sequence seq) {
        String fasta = "";
        // Constructing Fasta out of ID and Sequence
        return fasta;
    }
}
```

```java
public class AminoSequence extends Sequence {

    public AminoSequence(int id, String seq) {
        super(id, seq);
    }

    public int getMass() {
        int mass = 0;
        // Code for adding up mass of each amino
        return mass;
    }
}
```

```java
public static void main(String[] args) {
    AminoSequence as = new AminoSequence(0, "AMH");
    System.out.println(as.getSequence());
    System.out.println(as.getMass());
    System.out.println(Sequence.getFasta(as));
}
```

Wenn SuperKlasse gefragt ist, wird auch die SubKlasse erlaubt.