Programmierung

Organisatorisches:

`http://programmierung.informatik.rwth-aachen.de`

Algorithmus

$\approx$ Eindeutige, genaue Anleitung für die Lösung eines Problems.

Programm

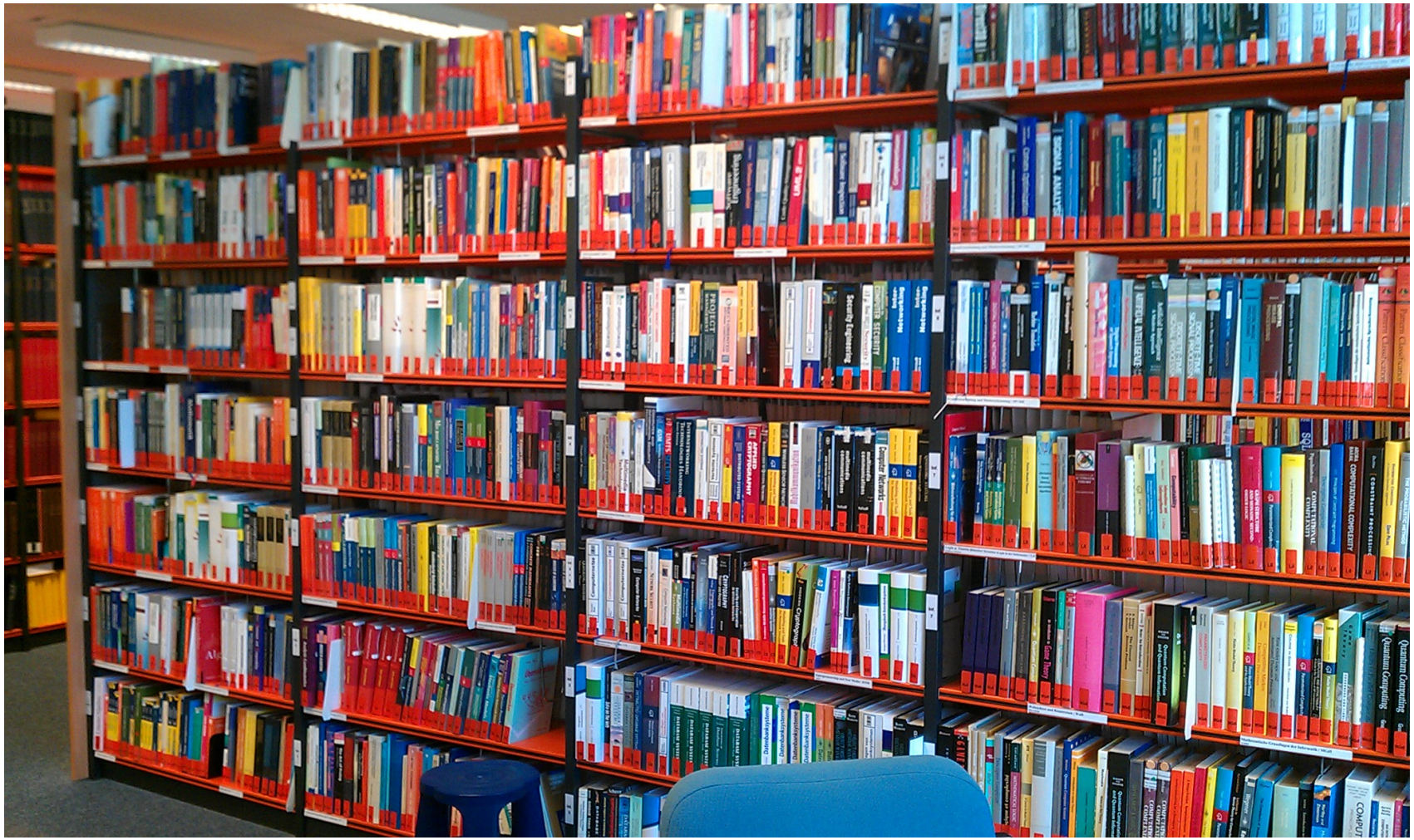
$\approx$ Eindeutige, genaue Anleitung für die Lösung eines Problems in einer *Programmiersprache* formuliert.

Programme und Algorithmen

Es gibt sehr alte Algorithmen:

- Euklidischer Algorithmus
  $\longrightarrow$ größter gemeinsamer Teiler
- Sieb des Erathostenes
  $\longrightarrow$ Berechnung von Primzahlen
- Babylonisches Wurzelziehen
  $\longrightarrow$ Näherung der Quadratwurzel

Literatur zur Vorlesung



Handapparat der Informatik-Bibliothek.

Literatur zur Vorlesung

Ausschnitt zur Programmierung.

Tabelliermaschinen (1890)

Bild von Arnold Reinhold

Neues Kapitel

Überblick

1. Einführung in die objektorientierte Programmierung
   - Klassen und Objekte
   - Vererbung, Setter, Getter
   - Primitive Datentypen
   - Kontrollstrukturen
   - Arrays

Erstes Beispiel einer Klasse

```
class Person {
    String nachname;
    String vorname;
}
```

Vorname und *Nachname* können gelesen und geschrieben werden.

```
Person fahrer = new Person();
fahrer.vorname = "Karl";
fahrer.nachname = "Ranseier";
String x = fahrer.nachname;
```

Private Attribute

```
class Person {
  private String nachname;
  private String vorname;
  String getNachname() {
    return nachname;
  }
  String getVorname() {
    return vorname;
  }
  void setNachname(String name) {
    nachname = name;
  }
  void setVorname(String name) {
    vorname = name;
  }
}
```

Private Attribute

Auf *vorname* und *nachname* kann nicht direkt zugegriffen werden.

Dies ist nur über dafür vorgesehene Methoden möglich.

```
Person fahrer = new Person();
fahrer.setVorname("Karl");
fahrer.setNachname("Ranseier");
String x = fahrer.getNachname();
```

Konstruktoren

```
class Person {
  private String nachname;
  private String vorname;

  public Person(String vorname, String nachname) {
    this.nachname = nachname;
    this.vorname = vorname;
  }

  String getNachname() {
    return nachname;
  }

  ...
```

Konstruktoren

Nicht mehr möglich:

```
Person fahrer = new Person();
```

Wir müssen unseren Konstruktor verwenden:

```
Person fahrer = new Person("Karl","Ranseier");
```

Überblick

Vererbung

Eine Oberklasse kann Unterklassen haben:

```
class Person {
    String nachname;
    String vorname;
    ...
}
```

Unterklasse:

```
class Student extends Person {
    String matrikelnummer;
    public Student(String nachname, String vorname, String nummer) {
        this.nachname = nachname;
        this.vorname = vorname;
        matrikelnummer = nummer;
    }
}
```

Vererbung

```
Student karl = new Student("Karl","Ranseier","123456");
String x = karl.matrikelnummer;
```

Das geht aber nicht:

```
Person karl = new Student("Karl","Ranseier","123456");
String x = karl.matrikelnummer;
```

Stattdessen:

```
Person karl = new Student("Karl","Ranseier","123456");
String x = ((Student)karl).matrikelnummer;
```

Dies nennen wir einen *cast*.