Selbstkorrigierende Programme

Defensive Programmierung: Fehlertolerant sein.

Selbstkorrigierendes Programm:

1. Ergebnis und Zertifikat mit einem effizienten, aber komplizierten Programm berechnen.
2. Zertifikat überprüfen.
3. Überprüfung negativ: Ergebnis neu berechnen mit einem langsamen, aber einfachen und sicheren Programm.

```
int gcd3(int a, int b) {
  int x = 1, y = 0, xx = 0, yy = 1, quot, temp;
  int r = a, rr = b;
  while (rr != 0) {
    quot = r/rr;
    temp = rr; rr = r - quot * rr; r = temp;
    temp = xx; xx = x - quot * xx; x = temp;
    temp = yy; yy = y - quot * yy; y = temp;
  }
  if(!(x * a + y * b == r && a%r == 0 && b%r == 0)) {
    r = a;
    while(a%r != 0 || b%r != 0) {
      r = r - 1;
    }
  }
  return r;
}
```

Überblick

3. Fehler finden und vermeiden
   - Contract Programming und der Hoare-Kalkül
   - Debugging
   - Testen

Debugging – Fehler beseitigen

Fehler entstehen weniger, wenn das Programm gut geschrieben wird.

- Gute Dokumentation
- Gute Klassenstruktur
- Aussagekräftige Namen (Klassen, Methoden, Variablen)
- Einheitliche Namensstruktur
- Kurze Methoden
- Code Duplication vermeiden
- . . . (später mehr)

Debugging – Fehler beseitigen

Fehler können leichter entdeckt werden, wenn das Programm gut geschrieben wird.

- Debug-Ausgaben, „verbose mode“ (abschaltbar)
- **assert**s einbauen
- Contract Programming
- Tests parallel entwickeln
- Methoden einzeln testbar auslegen
- Zusätzliche Visualisierung
- ...

Debugging – Fehler beseitigen

Fehler finden:

- Programm genau inspizieren
- Code Walkthrough
- Teile nacheinander auskommentieren
- Debug-Ausgaben einbauen
- **assert**s einbauen
- Breakpoints setzen, Variablen inspizieren
- Einzelschrittmodus

Einschub: Statische Methoden

```
class Kegelbahn {
  ...
  public static double sqdistance(Point p, Point q) {
    double dx = p.x − q.x;
    double dy = p.y − q.y;
    return dx ∗ dx + dy ∗ dy;
  }
  ...
}
```

Eine statische Methode gehört zur Klasse, nicht zum einzelnen Objekt. Sie kann daher nicht auf Instanzvariablen zugreifen.

Ausserhalb der Klasse *Kegelbahn* können wir sie so aufrufen:

```
...
double dist = Kegelbahn.sqdistance(point1, point2);
...
```

Einschub: Mengen in der Java-Bibliothek

```
Set<Person> freunde = new HashSet<Student>();
freunde.add(karl);
freunde.add(eva);
int anzahl = freunde.size();
for(Person p : freunde) {
  System.out.println(p);
}
```

Programmieren mit solchen „Container-Klassen“ ist sehr bequem.

Später mehr dazu.

Vorsicht: Es fehlen noch wichtige Voraussetzungen.