Praxis der Programmierung

Klassenvariablen und -methoden, Vererbung

Institut für Informatik und Computational Science
Universität Potsdam

Henning Bordihn

Klassenvariablen und -methoden

Klassenvariablen und -methoden

- werden mit dem Schlüsselwort `static` vereinbart
- sind der Klasse, nicht den Objekten zugeordnet; d.h.
  - **Klassenmethoden** können auch aufgrufen werden, ohne dass ein Objekt der Klasse erzeugt wurde
  - **Klassenvariablen** sind (mit ihren Werten) allen Exemplaren der Klasse gemeinsam, sind also *klassenglobal*
  - Klassenvariablen werden vor dem ersten Exemplar der Klasse angelegt, müssen also wie globale Variablen definiert werden
- werden mit dem Klassennamen (statt mit dem Objektnamen) qualifiziert
- Klassenmethoden dürfen direkt nur auf Datenelemente und Methoden zugreifen, die ebenfalls `static` sind!

Aufgaben 1 bis 3

Vererbung

Aufgabe 4

WRITE ONCE!

Prinzipien bei der Vererbung

- Definition von Klassen (*Unterklassen*), die von einer anderen Klasse (*Oberklasse*) abgeleitet sind
- Vererbung aller Datenelemente und Methoden von der Oberklasse an jede ihrer Unterklassen
- Erweiterung der Oberklasse durch neue Datenelemente und Methoden möglich
- (geerbte) Methoden mit derselben Signatur können in verschiedenen Unterklassen verschieden implementiert sein (*Polymorphie*)!
  $\longrightarrow$ realisiert durch *Überschreiben* von (*virtuellen*) Methoden der Oberklasse
- hierarchische Vererbung (Kinder, Enkel, Urenkel, ...)
- Mehrfachvererbung (Erben von mehreren Oberklassen gleichzeitig)

Definition einer Unterklasse

```
class Unterklasse : public Oberklasse {

    // neue Datenelemente

    // Konstruktoren
    Unterklasse(...) {   // ruft Oberklasse() auf
        ...
    }
    // oder
    Unterklasse(...) : Oberklasse( /* aktuelle Parameter */ ) {
        // neue Anweisungen
    }

    // neue Methoden
};
```

Konstruktoren bei der Vererbung

- Konstruktor der Unterklasse ruft zuerst einen Konstruktor der Oberklasse auf
  - den Standardkonstruktor oder
  - den explizit angegebenen Konstruktor

    `Unterklasse(...) : Oberklasse(***)`

- sorgt für Kompatibilität zur Oberklasse

  $\rightsquigarrow$ *Exemplare der Unterklasse sind spezialisierte Exemplare der Oberklasse.*

```
Oberklasse allgemein;
Unterklasse spezialisiert;

allgemein = spezialisiert;   // ok
spezialisiert = allgemein;   // geht nicht
```

Zugriffsattribute bei der Vererbung

- `class Unterklasse : public Oberklasse`
  $\rightsquigarrow$ alle Elemente (Datenelemente und Methoden) werden vererbt
  (*auf private Elemente kann aber nicht direkt zugegriffen werden*)

- `class Unterklasse : Oberklasse` oder
  `class Unterklasse : private Oberklasse`
  $\rightsquigarrow$ alle geerbten Elemente werden `private`

- Definition von Elementen mit `protected`:
  $\rightsquigarrow$ Zugriff nur für die Klasse und ihre Unterklassen

Zugriffsattribute – Beispiel

```
class Oberklasse {
private:
    int privat;
protected:
    int protect;
public:
    int publik;
};
```

```
int main() {
    Oberklasse o;
    Unterklasse u;

    int i = u.publik;  // Fehler
    i = o.publik;      // ok
    i = o.protect;     // Fehler
}
```

```
class Unterklasse : Oberklasse {  // privat!
public:
    int f1() { return privat; }   // geht nicht
    int f2() { return protect; }  // ok
    int f3() { return publik; }   // ok
```

Aufgabe 5

Definition einer Unterklasse

```
class Unterklasse : public Oberklasse {

    // neue Datenelemente

    // Konstruktoren
    Unterklasse(...) {   // ruft Oberklasse() auf
        ...
    }
    // oder
    Unterklasse(...) : Oberklasse( /* aktuelle Parameter */ ) {
        // neue Anweisungen
    }

    // neue Methoden
};
```

Mehrfachvererbung

- Aufzählen der Oberklassen, durch Komma getrennt:

```
class Unterklasse : public Elternklasse, public Superklasse
{ ... }
```

- Vorsicht vor Namenskonflikten
  - z.B. bei Datenelement `var` in beiden Oberklassen:

```
a = Elternklasse::var;
b = Superklasse::var;
```

  - besser Mehrfachvererbung vermeiden!

- *übrigens:*
  neue Elemente vedecken gleichnamige Elemente der Oberklasse(n)

Polymorphie durch Überschreiben von Methoden

- Redifinieren ("Verdecken") der Methode der Oberklasse
- bei Aufrufen wird die Imlementierung der Klasse abgearbeitet, die dem Typ entspricht, mit dem das Objekt definiert wurde
  $\rightsquigarrow$ Entscheidung durch den Compiler (statische/frühe Bindung)
- werden *virtuelle Methoden* (Schlüsselwort `virtual`) überschrieben, entscheidet der Typ des *Objekts*, an das der Aufruf gerichtet ist (dynamische/späte Bindung)
- Aber: bei Zuweisungen von Unterklassenobjekten an Oberklassenvariablen erfolgt automaisch eine Typkonversion
  $\rightsquigarrow$ Besonderheiten der Unterklasse gehen verloren!

Aufgaben 6 und 7