15.09.2010

Virtuelle Methoden

Thomas van Bruggen

dkfz.
DEUTSCHES
KREBSFORSCHUNGSZENTRUM
IN DER HELMHOLTZ-GEMEINSCHAFT

Virtuelle Methoden

Objektorientierte Programmierung

- Vererbung definiert eine Typ/Untertyp-beziehung zwischen Klassen.

```
class Human {
    public:
        void getsToEat() const {
            std::cout << “I get to eat pizza!" << std::endl;
        }
        …
};
```

Objektorientierte Programmierung

- Vererbung definiert eine Typ/Untertyp-beziehung zwischen Klassen.

```
class Human {
    public:
        void getsToEat() const {
            std::cout << “I get to eat pizza!" << std::endl;
        }
        …
};

class JustSomeone : public Human { };

int main() {
  JustSomeone someone;
  someone.getsToEat();
}
```

Virtuelle Methoden

Objektorientierte Programmierung

- Vererbung definiert eine Typ/Untertyp-beziehung zwischen Klassen.

```
class Human {
    public:
        void getsToEat() const {
            std::cout << "I get to eat pizza!" << std::endl;
        }
        …
};

class JustSomeone : public Human { };

int main() {
  JustSomeone someone;
  someone.getsToEat();
}
```

```
I get to eat pizza!
```

Virtuelle Methoden

```
class Dutchman : public Human {
  public:
    void getsToEat() const {
      std::cout << "  I get to eat stroopwafels!" << std::endl;
    }
};

class German : public Human {
  public:
    void getsToEat() const {
      std::cout << "  I get to eat Schnitzel!" << std::endl;
    }
};

int main() {
  std::vector<Human*> humans;
  humans.push_back(new Dutchman());
  humans.push_back(new German());
  humans.push_back(new JustSomeone());

  for (std::vector<Human*>::const_iterator it = humans.begin(); it != humans.end(); ++it) {
    (*it)->getsToEat();
    delete *it;
  }
}
```

???

Virtuelle Methoden

```
class Dutchman : public Human {
  public:
    void getsToEat() const {
      std::cout << "  I get to eat stroopwafels!" << std::endl;
    }
};

class German : public Human {
  public:
    void getsToEat() const {
      std::cout << "  I get to eat Schnitzel!" << std::endl;
    }
};

int main() {
  std::vector<Human*> humans;
  humans.push_back(new Dutchman());
  humans.push_back(new German());
  humans.push_back(new JustSomeone());

  for (std::vector<Human*>::const_iterator it = humans.begin(); it != humans.end(); ++it) {
    (*it)->likesToEat();
    delete *it;
  }
}
```

```
I get to eat pizza!
I get to eat pizza!
I get to eat pizza!
```

Virtual methods

Zwei fälle kann man noch separat lösen aber es wird schnell aufwendig

```
std::vector<Dutchman*> dutchmen;
std::vector<German*> germans
dutchman.push_back(new Dutchman());
germans.push_back(new German());
```

…usw

Es kann schöner

- Dynamic binding/dispatch (Dynamische Bindung): Erst während der Laufzeit wird entschieden welches Stück Code ausgeführt wird.
- In C++ non-default (im gegenteil zu z.B. Java) und optional.
- keyword *virtual.*

Virtuelle Methoden

```
class Human {
    public:
        virtual void likesToEat() const {
            std::cout << "I get to eat pizza!" << std::endl;
        }
        …
};

int main() {
  std::vector<Human*> humans;
  humans.push_back(new Dutchman());
  humans.push_back(new German());
  humans.push_back(new JustSomeone());

  for (std::vector<Human*>::const_iterator it = humans.begin(); it != humans.end(); ++it) {
    (*it)->likesToEat();
    delete *it;
  }
}
```

```
I get to eat stroopwafels!
I get to eat Schnitzel!
I get to eat pizza!
```

Virtuelle Methoden

Abstrakte Klassen

- Konzept einer Klasse die selber nicht instanziiert wird.
- Definiert Methoden die von abgeleiteten Klassen implementiert werden müssen.
- Enthält ein oder mehr **Rein-virtuelle Methoden**.
- Rein-abstrakte Klassen haben nur rein-virtuelle Methoden. Kann man vergleichen mit dem interface Konzept aus Java.
- Kontrakt zwischen Hersteller und Benutzer der Basisklasse

```
class AbstractClass {
  public:
    virtual void AbstractMemberFunction() = 0; // pure virtual method
    virtual void NonAbstractMemberFunction1(); //virtual method
void NonAbstractMemberFunction2(); };
```