Einführung in die Programmierung

$$\frac{2}{-5}$$

Tutorium 10

Wintersemester 2025/2026

Arbeitsgruppe Systemsoftware
Angewandte Informatik 12

technische universität dortmund

Arbeitsgruppe Systemsoftware

Was sind Klassen und Objekte, und welchen Zusammenhang haben diese?

- Klassen sind Baupläne für Datentypen, die Eigenschaften und Verhalten definieren.
- Objekte sind die konkreten Instanzen dieser Klassen, die zur Laufzeit Speicher belegen.

Welchen wesentlichen Unterschied haben struct- und class-Definitionen?

Der Unterschied liegt in der Standardsichtbarkeit der Member.

- Bei einem struct ist diese standardmäßig public.
- Bei einer class ist diese standardmäßig private.

Erklären Sie den Begriff des Überladens von Methoden und zeigen Sie ein Beispiel.

Überladen bedeutet, mehrere Methoden mit dem gleichen Namen, aber unterschiedlichen Parameterlisten im selben Gültigkeitsbereich zu definieren.

```
1  void print(int i);
2  void print(double d);
```

Aufgabe 2

Implementierung einer Fraction-Klasse, die Brüche automatisch kürzt und normalisiert, Kapselung sicherstellt und Operatoren für Arithmetik und Ausgabe bereitstellt.

$\frac{6}{-15}$ numerator / denominator

gcd(6, -15) = 3

$\frac{6/3}{-15/3}$ numerator / denominator

$\frac{2}{-5}$ numerator / denominator

Vorzeichen anpassen

$\frac{-2}{5}$ numerator / denominator

```
70  // TODO1:
71  int64_t common = gcd(numerator, denominator);
72
73          numerator /= common;
74          denominator /= common;
75
76          if (denominator < 0) {
77              if (numerator == INT64_MIN) {
78                  numerator = 0;
79                  denominator = 1;
80                  return;
81              } else {
82                  numerator = -numerator;
83              }
84              denominator = -denominator;
85          }
```

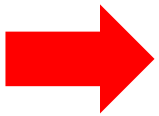
```
95  // TODO2:
96  int64_t Fraction::getNumerator() const {
97      return numerator;
98  }
99
100 void Fraction::setNumerator(int64_t num) {
101     numerator = num;
102     simplify();
103 }
104
105 int64_t Fraction::getDenominator() const {
106     return denominator;
107 }
108
109 void Fraction::setDenominator(int64_t den) {
110     denominator = den;
111     simplify();
112 }
```

```
123 //TODO3:
124 Fraction Fraction::operator+(const Fraction& other) const {
125     int64_t num1 = numerator * other.getDenominator();
126     int64_t num2 = other.getNumerator() * denominator;
127     int64_t den = denominator * other.getDenominator();
128     return Fraction(num1 + num2, den);
129 }
130
131 Fraction Fraction::operator-(const Fraction& other) const {
132     int64_t num1 = numerator * other.getDenominator();
133     int64_t num2 = other.getNumerator() * denominator;
134     int64_t den = denominator * other.getDenominator();
135     return Fraction(num1 - num2, den);
136 }
```

```
...
138 Fraction Fraction::operator*(const Fraction& other) const {
139     int64_t num = numerator * other.getNumerator();
140     int64_t den =  denominator * other.getDenominator();
141     return Fraction(num, den);
142 }
143
144 Fraction Fraction::operator/(const Fraction& other) const {
145     int64_t num = numerator * other.getDenominator();
146     int64_t den =  denominator * other.getNumerator();
147     return Fraction(num, den);
148 }
```

Welche Probleme könnten dadurch entstehen?

```
15 class Fraction {
16 private:
17     int64_t numerator;
18     int64_t denominator;
19     void simplify();
20
21 public:
22     Fraction(int64_t num, int64_t den);
23     int64_t getNumerator() const;
24     void setNumerator(int64_t num);
25 …
```



```
15 class Fraction {
16 public:
17     int64_t numerator;
18     int64_t denominator;
19     void simplify();
20
21     Fraction(int64_t num, int64_t den);
22     int64_t getNumerator() const;
23     void setNumerator(int64_t num);
24 …
```

Welche Probleme könnten dadurch entstehen?

```
1  Fraction f(2, 4); // Konstruktor kürzt automatisch auf 1/2
2
3  // ABER: Die Attribute sind öffentlich, sodass wir sie verändern können!
4  f.numerator = 100; // Ergebnis: Der Bruch ist jetzt 100/2
5  // Nun ist der Bruch wieder ungekürzt und die Methode simplify() umgangen
6  …
```

Verschiedene Operatoren können innerhalb von Klassen überschrieben werden

Arithmetische Operatoren
+, -, *, /, %, ++, --

Zuweisungsoperatoren
=, +=, -=, *=, /=, %=, &=, |=, ^=, <<=

Vergleichsoperatoren
==, !=, <, >, <=, >=

Bitweise Operatoren
~, &, |, ^, <<, >>

Logische Operatoren
!, &&, ||

```
template <typename U> bool operator==(const Fraction<U>& other) const {
    return ???
}

template <typename U> bool operator<(const Fraction<U>& other) const {
    return ???
}
```

```
template <typename U> bool operator==(const Fraction<U>& other) const {
    return numerator * other.getDenominator() == other.getNumerator() * denominator;
}

template <typename U> bool operator<(const Fraction<U>& other) const {
    return numerator * other.getDenominator() < other.getNumerator() * denominator;
}
```

Vergleichsoperatoren

```
template <typename U> bool operator<=(const Fraction<U>& other) const {
    return ???
}

template <typename U> bool operator>(const Fraction<U>& other) const {
    return ???
}

template <typename U> bool operator>=(const Fraction<U>& other) const {
    return ???
}

template <typename U> bool operator!=(const Fraction<U>& other) const {
    return ???
}
```

Vergleichsoperatoren

```
template <typename U> bool operator<=(const Fraction<U>& other) const {
    return (*this < other) | (*this == other);
}

template <typename U> bool operator>(const Fraction<U>& other) const {
    return !(*this <= other);
}

template <typename U> bool operator>=(const Fraction<U>& other) const {
    return !(*this < other);
}

template <typename U> bool operator!=(const Fraction<U>& other) const {
    return !(*this == other);
}
```