Osnove objektnog programiranja

Objektno programiranje - 2. vježbe

Marko Živković

Prirodoslovno-matematički fakultet,
Sveučilište u Zagrebu

13. ožujka 2026.

friend

Ključna riječ `friend` daje pristup privatnim podacima iz okoline

Predmet.h

```
1  #pragma once
2  #include <string>
3  #include <unordered_set>
4
5  using namespace std;
6
7  class Student; // prethodna deklaracija
8
9  class Predmet {
10     string naziv;
11     unordered_set<Student*> studenti;
12
13 public:
14     Predmet(string naziv);
15
16     friend class Student;
17
18     string getNaziv();
19
20     void ispisiListuStudenata();
21 };
```

Friend funkcija:

Predmet.h

```
1  #pragma once
2  #include <string>
3  #include <unordered_set>
4  #include "Student.h" // Includamo header da vidimo deklarirane funkcije.
5  // Prethodna deklaracija tada nije potrebna.
6
7  using namespace std;
8
9  class Predmet {
10     string naziv;
11     unordered_set<Student*> studenti;
12
13 public:
14     Predmet(string naziv);
15
16     friend void Student::upisiPredmet(Predmet* predmet);
17
18     friend void Student::ispisiPredmet(Predmet* predmet);
19
20     string getNaziv();
21
22     void ispisiListuStudenata();
23 };
```

Enum

Zadatak 1. Studenti mogu biti redovni i izvanredni. Onemogućite da izvanredni student upiše više od 3 predmeta.

Student.h

```
1  #pragma once
2  #include <string>
3  #include <unordered_set>
4
5  using namespace std;
6
7  class Predmet;
8
9  enum class Status {REDOVNI, IZVANREDNI};
10
11 class Student {
12     const string ime;
13     Status status;
14     int godina = 0;
15     unordered_set<Predmet*> upisaniPredmeti;
16
17 public:
18     Student(string ime, Status status = Status::REDOVNI); // default
19
20     ...
21 };
```

Student.cpp

```
1  ...
2
3  Student::Student(string ime, Status status) : ime(ime), status(status) {}
4
5  void Student::setGodina(int godina) {
6      this->godina = godina;
7  }
8
9  void Student::upisiPredmet(Predmet* predmet) {
10     if (status == Status::IZVANREDNI && upisaniPredmeti.size() >= 3) {
11         cout << "Izvanredni student " << ime
12              << " ne moze upisati vise predmeta." << endl;
13     }
14     else {
15         upisaniPredmeti.insert(predmet);
16         predmet->studenti.insert(this);
17     }
18 }
19
20 ...
```

main()

```
1  int main() {
2      Student prvi("Petar Petrovic");
3      Student drugi("Ana Horvat", Status::IZVANREDNI);
4
5      Predmet p1("Elementarna matematika");
6      Predmet p2("Matematicka analiza");
7      Predmet p3("Objektno programiranje");
8      Predmet p4("Teorija skupova");
9
10     prvi.upisiPredmet(&p1);
11     prvi.upisiPredmet(&p2);
12     prvi.upisiPredmet(&p3);
13     prvi.upisiPredmet(&p4);
14     drugi.upisiPredmet(&p1);
15     drugi.upisiPredmet(&p2);
16     drugi.upisiPredmet(&p3);
17     drugi.upisiPredmet(&p4);
18
19     prvi.ispisiPodatke();
20     drugi.ispisiPodatke();
21 }
```

Kompleksni brojevi

Zadatak 2. Implementirajte klasu koja opisuje kompleksne brojeve. Predefinirajte operatore `+` i `*`. Definirajte funkciju `toString()` koja će vratiti tekstualni zapis kompleksnog broja.

ComplexNmb.h

```
1  using namespace std;
2
3  class ComplexNumber {
4      double real = 0;
5      double img = 0;
6
7  public:
8      ComplexNumber(double real, double imaginary = 0);
9
10     string toString();
11
12     friend ComplexNumber operator+(ComplexNumber, ComplexNumber);
13
14     friend ComplexNumber operator*(ComplexNumber, ComplexNumber);
15 };
```

ComplexNmb.cpp

```
1  #include <string>
2  #include "ComplexNmb.h"
3  using namespace std;
4
5  ComplexNumber::ComplexNumber(double real, double imaginary)
6      : real(real), img(imaginary) {}
7
8  string ComplexNumber::toString() {
9      return to_string(real) + "+" + to_string(img) + "i";
10 }
11
12 ComplexNumber operator+(ComplexNumber a, ComplexNumber b) {
13     return ComplexNumber(a.real + b.real, a.img + b.img);
14 }
15
16 ComplexNumber operator*(ComplexNumber a, ComplexNumber b) {
17     return ComplexNumber(a.real * b.real - a.img * b.img,
18                          a.real * b.img + a.img + b.real);
19 }
```

main.cpp

```
1  #include <iostream>
2  #include "ComplexNmb.h"
3
4  using namespace std;
5
6  int main(void) {
7      ComplexNumber x(2, 3);
8      ComplexNumber y(3, -1);
9
10     cout << "(" << x.toString() << ")+(" << y.toString()
11          << ")=" << (x + y).toString() << endl;
12     cout << "(" << x.toString() << ")*(" << y.toString()
13          << ")=" << (x * y).toString() << endl;
14 }
```

Nasljeđivanje

Zadatak 3. Implementirajte klase `Pravokutnik` i `Kvadrat` koje imaju definirane duljine stranica tipa `double` i funkcije `povrsina()` i `opseg()`.

Oblici.h

```
1  class Pravokutnik {
2      double x, y;
3
4  public:
5      Pravokutnik(double, double);
6
7      double povrsina();
8
9      double opseg();
10 };
11
12 class Kvadrat : public Pravokutnik {
13 public:
14     Kvadrat(double);
15 };
```

Oblici.cpp

```
1  #include "Oblici.h"
2
3  Pravokutnik::Pravokutnik(double x, double y) : x(x), y(y) {};
4
5  double Pravokutnik::povrsina() {
6      return x * y;
7  }
8
9  double Pravokutnik::opseg() {
10     return 2 * (x + y);
11 }
12
13 Kvadrat::Kvadrat(double x) : Pravokutnik(x, x) {};
```

main.cpp

```
1  #include <iostream>
2  #include "Oblici.h"
3
4  using namespace std;
5
6  int main(void) {
7      Pravokutnik p(2, 3);
8      Kvadrat k(4);
9
10     cout << p.povrsina() << endl;
11     cout << p.opseg() << endl;
12     cout << k.povrsina() << endl;
13     cout << k.opseg() << endl;
14 }
```

Zadatak 4. Dodajte mogućnost promjene dimenzija: `setLength` i `setWidth` za `Pravokutnik` te `setSize` za `Kvadrat`.

Napomena: `setLength` i `setWidth` ne smiju biti dostupni za objekt tipa `Kvadrat`.

Oblici.h

```
1  class BasePravokutnik {
2  protected: // da bude dostupno u podklasama
3      double x, y;
4
5      BasePravokutnik(double, double); // ne treba biti javno dostupno
6
7  public:
8      double povrsina();
9      double opseg();
10 };
11
12 class Pravokutnik : public BasePravokutnik {
13 public:
14     Pravokutnik(double, double);
15
16     void setLength(double);
17     void setWidth(double);
18 };
19
20
21 class Kvadrat : public BasePravokutnik {
22 public:
23     Kvadrat(double);
24
25     void setSize(double);
26 };
```

Oblici.cpp

```cpp
#include "Oblici.h"

BasePravokutnik::BasePravokutnik(double x, double y) : x(x), y(y) {};

double BasePravokutnik::povrsina() {
    return x * y;
}
double Pravokutnik::opseg() {
   return 2 * (x + y);
}

Pravokutnik::Pravokutnik(double x, double y) : BasePravokutnik(x,y) {};

void Pravokutnik::setLength(double x) {
    this->x = x;
}
void Pravokutnik::setWidth(double y) {
    this->y = y;
}

Kvadrat::Kvadrat(double x) : BasePravokutnik(x, x) {};

void Kvadrat::setSize(double x) {
    this->x = x;
    this->y = x;
}
```

main.cpp

```
1  #include <iostream>
2  #include "Oblici.h"
3
4  using namespace std;
5
6  int main(void) {
7      Pravokutnik p(2, 3);
8      Kvadrat k(4);
9
10     cout << p.povrsina() << endl;
11     cout << p.opseg() << endl;
12     cout << k.povrsina() << endl;
13     cout << k.opseg() << endl;
14
15     p.setLength(1);
16     p.setWidth(5);
17     k.setSize(6);
18
19     cout << p.povrsina() << endl;
20     cout << p.opseg() << endl;
21     cout << k.povrsina() << endl;
22     cout << k.opseg() << endl;
23 }
```

Virtualne funkcije

Zadatak 5. Dodajte klasu `Krug`. Dodajte klasu `Oblik` koja će biti bazna klasa svim oblicima i imati funkcije `povrsina()` i `opseg()`.

Oblici.h

```
1  class Oblik {
2  public:
3      virtual double povrsina() = 0; // pure virtual funkcija
4      virtual double opseg() = 0;
5  };
6
7  class Krug : public Oblik {
8      double r;
9
10 public:
11     Krug(double);
12
13     double povrsina() override;
14     double opseg() override;
15 };
16
17 class BasePravokutnik : public Oblik {
18 protected:
19     double x, y;
20
21     BasePravokutnik(double, double);
22
23 public:
24     double povrsina() override;
25     double opseg() override;
26 };
27 ...
```

Oblici.cpp

```
1  #include "Oblici.h"
2
3  Krug::Krug(double r) : r(r) {};
4
5  double Krug::povrsina() {
6      return r * r * 3.14;
7  }
8
9  double Krug::opseg() {
10     return 2 * r * 3.14;
11 }
12
13 BasePravokutnik::BasePravokutnik(double x, double y) : x(x), y(y) {};
14
15 double BasePravokutnik::povrsina() {
16     return x * y;
17 }
18
19 double Pravokutnik::opseg() {
20     return 2 * (x + y);
21 }
22
23 ...
```

main.cpp

```
1  #include <iostream>
2  #include "Oblici.h"
3
4  using namespace std;
5
6  void ispisiPodatke(Oblik& o) { // mora biti referenca ili pokazivac
7  // jer se objekt ne moze inicijalizirati, pa ni kopirati
8      cout << "Povrsina: " << o.povrsina() << "\n"
9           << "Opseg: " << o.opseg() << endl;
10 }
11
12 int main(void) {
13     Pravokutnik p(2, 3);
14     Kvadrat k(4);
15     Krug kr(3);
16
17     ispisiPodatke(p);
18     ispisiPodatke(k);
19     ispisiPodatke(kr);
20 }
```

Prvi algoritam i generičke klase

Zadatak 6. Dodajte funkciju `minSquarePart()` za pravokutnike s cjelobrojnim stranicama koja vraća minimalni broj kvadrata u koje se pravokutnik može izrezati.

Postupak:

- Zadržavamo pravokutnike i kvadrate sa stranicama tipa `double`. Zato treba uvesti predloške klasa koji će imati stranice generičkog tipa `T`.
- Predlošci klasa moraju biti u potpunosti implementirani u headeru.
- Implementacija algoritma za konkretnu klasu `Pravokutnik` stranice tipa `int` može ići u .cpp datoteku.

Oblici.h

```
1  template<typename T>
2  class Oblik {
3  public:
4      virtual T povrsina() = 0;
5
6      virtual T opseg() = 0;
7  };
8
9  class Krug : public Oblik<double> {
10     double r;
11
12 public:
13     Krug(double r) : r(r) {};
14
15     double povrsina() override {
16         return r * r * 3.14;
17     }
18
19     double opseg() override {
20         return 2 * r * 3.14;
21     }
22 };
23
24 ...
```

Oblici.h

```
1  ...
2
3  template<typename T>
4  class BasePravokutnik : public Oblik<T> {
5  protected:
6      T x, y;
7
8      BasePravokutnik(T x, T y) : x(x), y(y) {};
9
10 public:
11     T povrsina() override {
12         return x * y;
13     }
14
15     T opseg() override {
16         return 2 * (x + y);
17     }
18 };
19
20 template<typename T>
21 class Pravokutnik : public BasePravokutnik<T> {
22 public:
23     Pravokutnik(T x, T y) : BasePravokutnik<T>(x, y) {};
24
25 ...
```

Oblici.h

```
...
    void setLength(T x) {
        this->x = x;
    }

    void setWidth(T y) {
        this->y = y;
    }
};

class IntPravokutnik : Pravokutnik<int> {
public:
    IntPravokutnik(int, int);

    int minSquarePart();
};

template<typename T>
class Kvadrat : public BasePravokutnik<T> {
public:
    Kvadrat(T x) : BasePravokutnik<T>(x, x) {};

    void setSize(T x) {
        this->x = x;
        this->y = x;
    }
};
```

Oblici.cpp

```cpp
#include "Oblici.h"

IntPravokutnik::IntPravokutnik(int x, int y) : Pravokutnik(x, y) {};

int IntPravokutnik::minSquarePart() {
    int br = 0;
    int a = x > y ? x : y;
    int b = x > y ? y : x;

    while (b > 0) {
        br += a / b;
        int o = a % b;
        a = b;
        b = o;
    }

    return br;
}
```

main.cpp

```
1  #include <iostream>
2  #include "Oblici.h"
3
4  using namespace std;
5
6  int main(void) {
7      cout << "Unesite␣dimenzije␣pravokutnika:␣" << endl;
8      int v1;
9      int v2;
10     cin >> v1 >> v2;
11     IntPravokutnik p(v1, v2);
12     cout << "Pravokutnik␣" << v1 << "␣x␣" << v2
13         << "␣moze␣se␣podijeliti␣na␣" << p.minSquarePart()
14         << "␣kvadrata" << endl;
15 }
```