

PROGRAMIRANJE II

Primer 3 – Kompleksna števila

- Vmesnik razreda (*class interface*) predstavljajo vse javne komponente razreda.
- Uporabnik nekega razreda mora poznati le njegov vmesnik, podrobnosti o implementaciji pa zanj niso pomembne.
- Konstruktorji lahko imajo tudi privzete vrednosti.

// Complex.h

```
class Complex {  
private:  
    double real, imag;  
public:  
    Complex(); // default constructor  
    // conversion constructor with default arguments  
    Complex(double r, double i = 0);  
  
    void print();  
    Complex plus(Complex& c);  
};
```

Primer 3 – Kompleksna števila

```
// Complex.cpp
```

```
#include <iostream>
#include "Complex.h"

Complex::Complex() : real(0), imag(0) {
}

Complex::Complex(double r, double i): real(r), imag(i) {
}

void Complex::print() {
    std::cout << "(" << real << ", " << imag << "i)" << std::endl;
}

Complex Complex::plus(Complex& c) {
    Complex temp(real+c.real, imag+c.imag);
    return temp;
}
```

Primer 3 – Kompleksna števila

// Example03

```
#include <iostream>
#include "Complex.h"

int main() {
    Complex c1(1,1), c2(1), i(0,1);
    c1.print();
    c2.print();
    i.print();
    //c1.plus(5).print();
    c1.plus(i).print();

    std::cout << "Dynamic allocation" << std::endl;
    Complex* p_c = new Complex(4,2);
    p_c->print();
    c1.plus(*p_c).print();
    Complex c3(c1.plus(*p_c));
    c3.print();
    Complex* p_c1 = new Complex(c1.plus(i));
    p_c1->print();
    delete p_c;
    delete p_c1;

    return 0;
}
```

Konstantni objekti

- Konstanta spremenljivka (konstanta) `const int i=1;`
- Konstantni objekt – njegovega stanja ne moremo spreminjati!
- Konstantni objekt definiramo z določilom **const** pred definicijo objektne spremenljivke.
- Konstantni objekt lahko kliče samo konstantne metode. Takšne metode ne spreminjajo stanja objekta.
- Konstantne metode definiramo, tako da pri definiciji metode za argumenti navedemo določilo **const**.

Konstantni objekti – primer 4

```
// Complex.h
```

```
class Complex {  
private:  
    double real, imag;  
public:  
    Complex();  
    Complex(double r, double i = 0);  
  
    void print() const;           // constant method  
    // argument is a constant object and method is constant  
    Complex plus(const Complex& c) const;  
    void add(double d);          // non-constant method  
};
```

Konstantni objekti – primer 4

```
// Complex.cpp
```

```
#include <iostream>
#include "Complex.h"

Complex::Complex() : real(0), imag(0) {
}

Complex::Complex(double r, double i): real(r), imag(i) {
}

void Complex::print() const {
    std::cout << "(" << real << ", " << imag << "i)" << std::endl;
}

Complex Complex::plus(const Complex& c) const {
    Complex temp(real+c.real, imag+c.imag);
    return temp;
}

void Complex::add(double d) {
    real+=d;
}
```

Konstantni objekti – primer 4

// Example04

```
#include <iostream>
#include "Complex.h"

int main() {
    Complex c1(1,1);
    const Complex i(0,1);
    c1.add(10);
    //i.add(1);
    i.print();
    c1.print();
    std::cout << "-----" << std::endl;
    i.plus(c1).print();
    c1.plus(i).print();
    return 0;
}
```


Razredne spremenljivke in metode

- Vsak objekt ima shranjene svojstvene, njemu lastne, podatke v instančnih spremenljivkah.
- Metode se vedno izvajajo nad podatki objekta, kateremu je bilo poslano sporočilo.
- Občasno je zaželeno, da imamo podatek, ki je skupen vsem objektom nekega razreda.
- Imeti kopijo takšnega podatka v vsakem objektu ni dobra rešitev. Zakaj?

Razredne spremenljivke in metode

- Podatek, ki je skupen vsem objektom imenujemo statični podatek ali **razredna spremenljivka**.
- Metode, ki operirajo nad statičnimi podatki imenujemo statične oz. **razredne metode**.
- Razredne spremenljivke (statični podatki) so znani še preden ustvarimo objekte tega razreda.
- Razredne spremenljivke in metode določimo z določilom **static**.

Primer 5

// Complex.h

```
class Complex {
private:
    double real, imag;
    static int counter; // class variable
public:
    Complex(); // default constructor
    Complex(double r, double i = 0); // conversion constructor
    Complex(const Complex& c); // copy constructor
    ~Complex(); // destructor
    void print() const;
    Complex plus(const Complex& c) const;
    static int getCounter() { // class method
        return counter;
    }
};
```

Primer 5

```
// Complex.cpp
```

```
#include <iostream>
#include "Complex.h"

int Complex::counter=0; // class variables can't be initialized in class definition

Complex::Complex(): real(0), imag(0) {
    counter++;
}

Complex::Complex(double r, double i): real(r), imag(i) {
    counter++;
}

Complex::Complex(const Complex& c): real(c.real), imag(c.imag) {
    counter++;
}

Complex::~~Complex() {
    counter--;
}

void Complex::print() const {
    std::cout << "(" << real << ", " << imag << "i)" << std::endl;
}

Complex Complex::plus(const Complex& c) const {
    Complex temp(real+c.real, imag+c.imag);
    return temp;
}
```

Primer 5

// Example05

```
#include <iostream>
#include "Complex.h"

int main() {
    std::cout << "Current number of objects: " << Complex::getCounter() << std::endl;
    Complex c1(1,1);
    const Complex i(0,1);    // constant object
    c1.print();
    i.print();
    std::cout << "Current number of objects: " << c1.getCounter() << std::endl;
    std::cout << "Current number of objects: " << i.getCounter() << std::endl;

    Complex* p_c1 = new Complex(c1.plus(i));
    Complex* p_c2 = new Complex();
    p_c1->print();
    p_c2->print();
    std::cout << "Current number of objects: " << p_c1->getCounter() << std::endl;
    delete p_c1;
    delete p_c2;
    std::cout << "Current number of objects: " << c1.getCounter() << std::endl;

    return 0;
}
```

Kazalec **this**

- Razlika med funkcijo in metodo
 - `plus(c1, i)` vs. `c1.plus(i)`
- Metoda ima implicitni parameter, to je kazalec na objekt, ki je prejel sporočilo.
- Ta kazalec v programu dosežemo s rezervirano besedo **this**, ki je konstantni kazalec.

Kazalec this

- Primer

```
void Complex::print() const {  
    std::cout << "Object address: " << this << "=( " << this->real << ", " << this->imag << "i)" <<  
    this->counter << std::endl;  
}
```

Razlika med konstantnim kazalcem (int* const) in kazalcem na konstanto (const int*).

Kazalec **this**

- Kazalec **this** je konstantni kazalec in ga lahko uporabljamo samo znotraj nerazrednih (nestatičnih) metod razreda.
- Zunaj metod in v razrednih (statičnih) metodah kazalec **this** ne obstaja.

Vprašanja

