PROGRAMIRANJE II

Objekti kot elementi razredov

- Podatki (instančne spremenljivke) so lahko poljubnega tipa, torej tudi razreda.
- V tem primeru govorimo o vsebovanju oz. agregaciji/kompoziciji (*aggregation*, *composition*).

Primer 6

// Point.h

```
class Point {
private:
    int x, y;
public:
    Point();  // default constructor
    Point (int x, int y);  // constructor

    int getX() const;
    int getY() const;
    void print() const;
    double distance(const Point& p) const;
};
```

// Point.cpp

```
#include <iostream>
#include <cmath>
#include "Point.h"

Point::Point() : x(0), y(0) {
}

Point::Point (int x, int y) : x(x), y(y) {
}

int Point::getX() const {
    return x;
}

int Point::getY() const {
    return y;
}

void Point::print() const {
    std::cout << "(" << x << ", " << y << ") " << std::endl;
}

double Point::distance(const Point& p) const {
    return std::sqrt((double)(x - p.x)*(x - p.x)+(y - p.y)*(y - p.y));
}
```

Primer 6

// CPoint.h

```
#include "Point.h"

class CPoint {
private:
    Point p;    // composition
    int color;
public:
    CPoint();
    CPoint(int x, int y, int c);
    void print() const;
};
```

// CPoint.cpp

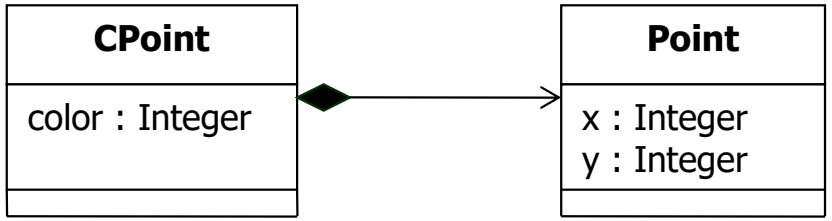
```
#include <iostream>
#include "CPoint.h"

CPoint::CPoint() : p(), color(0) {
}

CPoint::CPoint(int x, int y, int c) : p(x, y), color(c) {
}
void CPoint::print() const {
    std::cout  << "("  << p.getX() << ", " << p.getY() <<
                  ", color= " << color << ")" << std::endl;
}
```

Primer 6

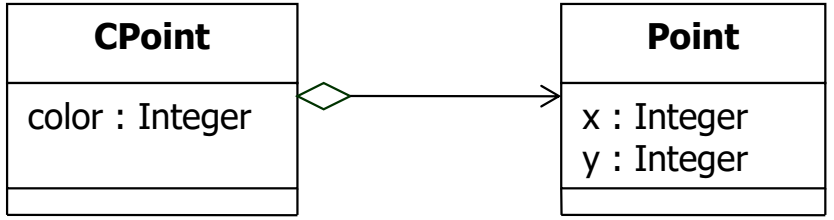
// Example06

```
int main() {
    Point p1(3,4);
    Point p2;
    CPoint c1;
    CPoint c2(1,2,3);
    CPoint c3(c2);

    p1.print();
    c1.print();
    c2.print();
    c3.print();
    std::cout << p1.distance(p2) << std::endl;
    //std::cout << p1.distance(c2) << std::endl;
    //std::cout << c2.distance(p1) << std::endl;
    return 0;
}
```

UML notacija (kompozicija)



UML notacija (agregacija)



Dedovanje

- Najpomembnejši koncept OOP je dedovanje.
- Dedovanje omogoča inkrementalni razvoj programov, kjer pri razvoju programske opreme iz nadrazredov podedujemo lastnosti, tj. strukturo in obnašanje.
- Programer zapiše le specifične lastnosti, ki se razlikujejo od tistih v nadrazredih.
- Dedovanje tako med razrede uvede tranzitivno relacijo in jih uredi v hierarhijo glede na njihove splošne in specifične lastnosti.

Dedovanje

- Dedovanje uvedemo z namenom enostavnejše konceptualne specializacije.
- Konceptualno modeliranje je proces organiziranja znanja o neki aplikacijski domeni v hierarhični red z namenom, da dobimo natančnejšo sliko o problemu oziroma da problem bolje razumemo.
- Dedovanje lahko uporabimo tudi za ponovno uporabo kode*.

Dedovanje - pojmi

- **class** B : **public** A { … };
- Nadrazred (*superclass*) ali bazni razred (*base class*)
- Podrazred (*subclass*) ali izpeljani razred (*derived class*)
- Izpeljava (*derivation*)
- Enkratno (*single*) in večkratno dedovanje (*multiple inheritance*)
- Posplošitev ali generalizacija (*generalization*)
- Specializacija (*specialization*)
- Relacija "is-a" (dedovanje) in "has-a" (vsebovanje: agregacija, kompozicija)

Dedovanje – Osnovno načelo

- Kjerkoli pričakujemo objekt nadrazreda lahko varno uporabimo objekt podrazreda. Saj ima takšen objekt vse lastnosti nadrazreda in dodatne specifične lastnosti.
- Primeri nadrazredov in podrazredov:

Nadrazred	Podrazred
Oseba	Student
Student	PodiplomskiStudent
Lik	Trikotnik
Štirikotnik	Pravokotnik

Primer 7

//Point.h

```
class Point {
private:
    int x, y;
public:
    Point();
    Point (int x1, int y1);

    int getX() const;
    int getY() const;
    void print() const;
    double distance(const Point& p) const;
};
```

// Point.cpp

```
#include <iostream>
#include <cmath>
#include "Point.h"

Point::Point() : x(0), y(0) {
}

Point::Point (int x, int y) : x(x), y(y) {
}

int Point::getX() const {
    return x;
}

int Point::getY() const {
    return y;
}

void Point::print() const {
    std::cout << "(" << x << ", " << y << ") " << std::endl;
}

double Point::distance(const Point& p) const {
    return std::sqrt((double)(x - p.x)*(x - p.x)+(y - p.y)*(y - p.y));
}
```

Primer 7

//CPoint.h

```
#include "Point.h"

class CPoint : public Point {
private:
    int color;
public:
    CPoint();
    CPoint(int x, int y, int c);

    void print() const;
};
```

// CPoint.cpp

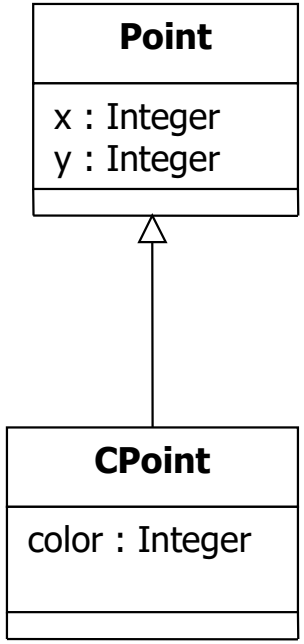
```
#include <iostream>
#include "CPoint.h"

CPoint::CPoint() : Point(), color(0) {
}

CPoint::CPoint(int x, int y, int c) : Point(x, y), color(c) {
}

void CPoint::print() const {
    std::cout << "(" << getX() << ", " << getY() <<
                 ") color=" << color << std::endl;
}
```

Primer 7

// Example07

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

int main() {
    Point p1(3,4);
    Point p2;
    CPoint c1;
    CPoint c2(1,2,3);
    CPoint c3(c2);

    p1.print();
    c1.print();
    c2.print();
    c3.print();

    std::cout << "-------------" << std::endl;
    std::cout << p1.distance(p2) << std::endl;
    std::cout << p1.distance(c2) << std::endl;
    std::cout << c1.distance(p1) << std::endl;
    std::cout << c2.distance(c1) << std::endl;
    return 0;
}
```

UML notacija



Določilo protected

- Ograjevanje oz. skrivanje elementov razreda (instančnih spremenljivk in metod) implementiramo z določili:
  - private (privatni)
  - public (javni)
  - protected (zaščiteni)
- Zaščiteni elementi (protected) posameznega razreda so dosegljivi tudi v metodah izpeljanih razredov tega razreda.

Virtualne metode - Primer 8

//Point.h

```
class Point {
protected:
    int x, y;
public:
    Point();
    Point(int x, int y);
    virtual ~Point();  // virtual destructor

    int getX() const;
    int getY() const;
    virtual void print() const;
    double distance(const Point& p) const;
};
```

//Point.cpp

```
#include <iostream>
#include <cmath>
#include "Point.h"

Point::Point() : x(0), y(0) {
}

Point::Point (int x, int y) : x(x), y(y) {
}

Point::~Point() {
     std::cout << "destructor Point" << std::endl;
}

int Point::getX() const {
    return x;
}

int Point::getY() const {
    return y;
}

void Point::print() const {
    std::cout << "(" << x << ", " << y << ") " << std::endl;
}

double Point::distance(const Point& p) const {
    return std::sqrt((double)(x-p.x)*(x-p.x)+(y-p.y)*(y-p.y));
}
```

Virtualne metode - Primer 8

//CPoint.h

```
#include "Point.h"

class CPoint : public Point {
protected:
    int color;
public:
    CPoint();
    CPoint(int x, int y, int c);
    ~CPoint();
    void print() const;
};
```

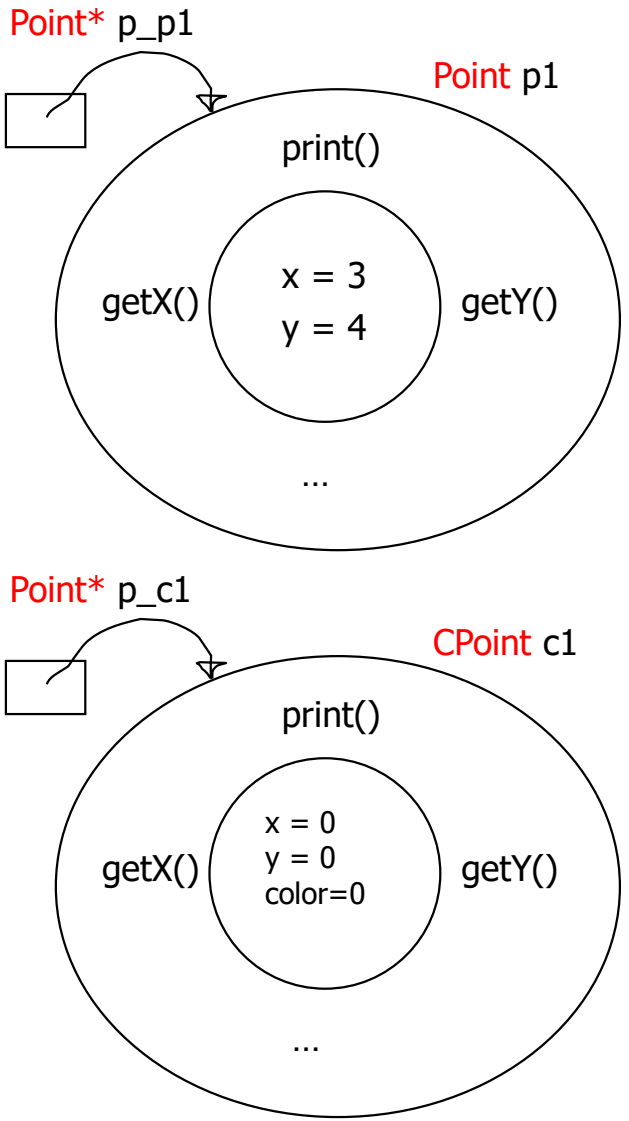
//CPoint.cpp

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

CPoint::CPoint() : Point(), color(0) {
}

CPoint::CPoint(int x, int y, int c) : Point(x, y), color(c) {
}

CPoint::~CPoint() {
    std::cout << "Destructor CPoint" << std::endl;
}

void CPoint::print() const {
    std::cout <<  "(" << x << ", " << y << ") color= " << color << std::endl;

}
```

Virtualne metode

//Example08

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

int main() {
    Point p1(3,4);
    CPoint c1;
    p1.print();
    c1.print();
    Point* p_p1 = &p1;
    Point* p_c1 = &c1;
    p_p1->print();
    p_c1->print();
    delete p_p1;
    delete p_c1;
    return 0;
}
```



Rezultat brez virtualnih metod:
(3, 4)
(0, 0) color= 0
(3, 4)
(0, 0) Zakaj?

Virtualne metode

- Tip p_c1 je Point* (statični tip)
- Tip c1 je CPoint (dinamični tip)
- Pri klicu metode je merodajen statični tip!
- Pri virtualnih metodah (določilo **virtual**) je merodajen dinamični tip!
- Virtualne metode imenujemo tudi polimorfne metode, za katere velja, da se na isto sporočilo objekti odzovejo na njim lasten način.

Virtualne metode

- Kadar je kakšna metoda v hierarhiji definirana kot virtualna, so vse metode, ki so ponovno definirane v podrazredih (polimorfna redefinicija), prav tako virtualne.
- Pri polimorfni redefiniciji se redefinirana metoda mora ujemati z virtualno metodo v signaturi (ime metode, število in tipi argumentov) ter tip rezultata metode*.
- Virtualne metode v podrazredih ni potrebno definirati ponovno. Če podrazred nima definirane virtualne metode, se kliče ustrezna metoda nadrazreda.

Pravila dobrega programiranja

- Vse metode, ki ne spreminjajo stanja objekta, definirajmo kot konstantne metode.
- V razredu, ki ima kako virtualno metodo, definirajmo tudi virtualni destruktor.

Pogoste napake programerja

- Klicanje metode, ki ni konstantna, s konstantnim objektom.
- Spreminjanje podatkov objekta v konstantni metodi.
- Uporaba kazalca **this** in nestatičnih podatkov v razrednih (statičnih) metodah razreda.

Pogoste napake programerja

- Obravnava objektov nadrazreda, kot da so objekti podrazreda.
- Pri redefiniciji metode nadrazreda v podrazredu je v navadi, da v njej pokličemo metodo nadrazreda in nato opravimo še nekaj dodatnega dela. Napačno je, da metoda v podrazredu kliče samo sebe, če tega ne želimo.

Pogoste napake programerja

- Ponovno definirana virtualna metoda v izpeljanem razredu nima istega izhodnega tipa in istih argumentov kot v nadrazredu.

Vprašanja