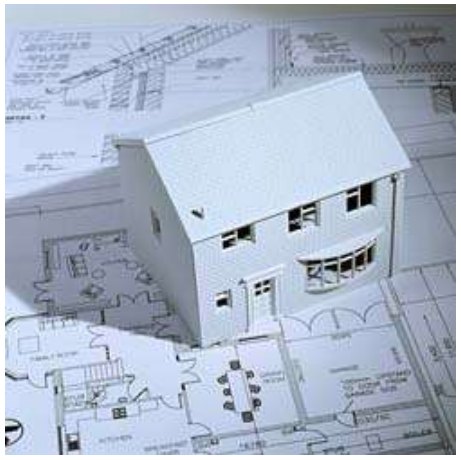
PROGRAMIRANJE II

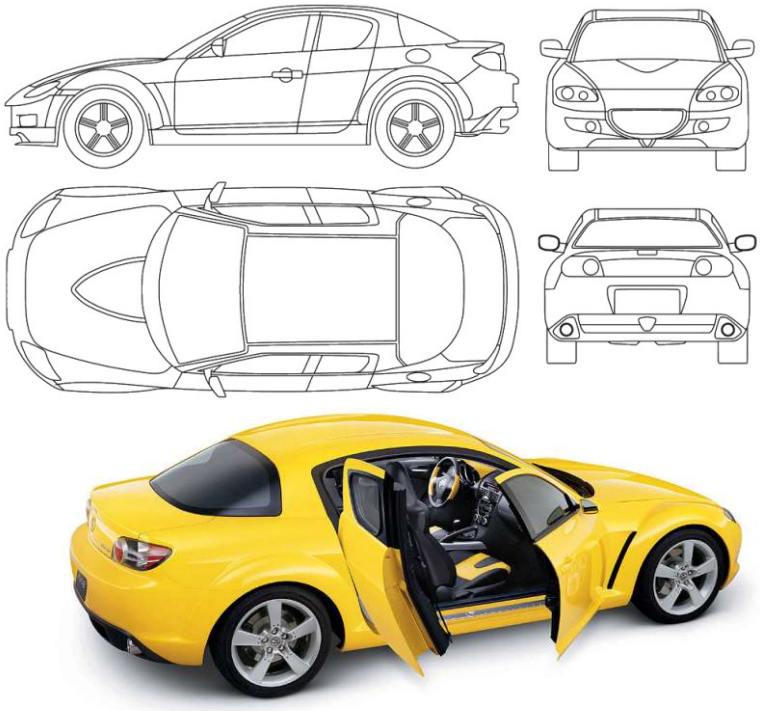
Programiranje: znanost ali umetnost?

- "If computer programming is to become an important part of computer research and development, a transition of programming from an art to a disciplined science must be effected." (CACM 1959)

Objekti in razredi

- Objekt – ograjuje stanje in obnašanje
- Razred – šablona za ustvarjanje objektov





Objekti in razredi

- Stanje oz. strukturo objektov opišemo s spremenljivkami, ki jim pravimo spremenljivke objekta ali tudi instančne spremenljivke (*instance variables*), obnašanje pa s sporočili oz. z metodami (*methods*).
- Pravimo tudi, da je objekt množica metod, ki si delijo stanje.

Definicija razreda

```
class X {
  private:
    // podatki (instančne spremenljivke)
  public:
    // metode
};
```

Definicija razreda

- Definicija razreda – vključitvena datoteka
- Implementacija razreda – glavna datoteka
- Z razredom definiramo nov tip (ADT), ki se obnaša podobno kot vgrajeni tip.
- Kapsuliranje oz. ograjevanje

Ustvarjanje objekta

- Kreiramo lahko spremenljivke takšnega razreda (tipa)
- Spremenljivko imenujemo objektna spremenljivka ali objekt

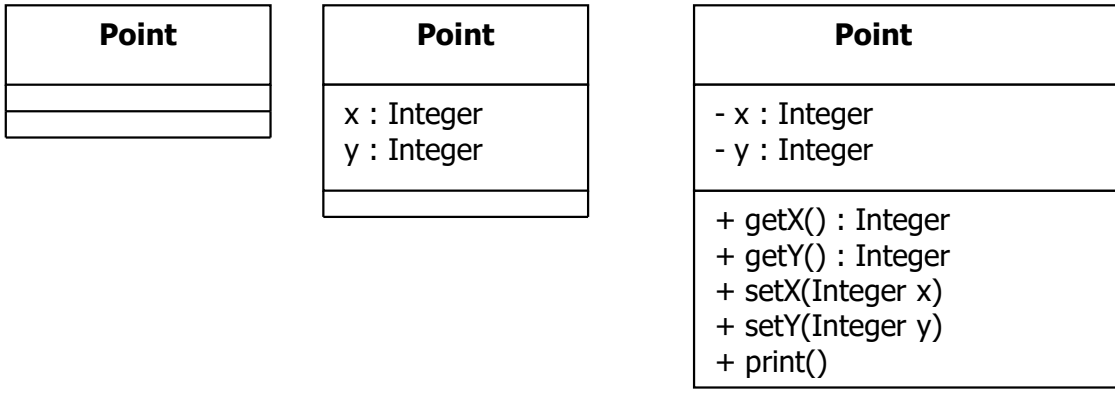
```
// example.cpp

int a;
Stack my_stack1;
X o;
```

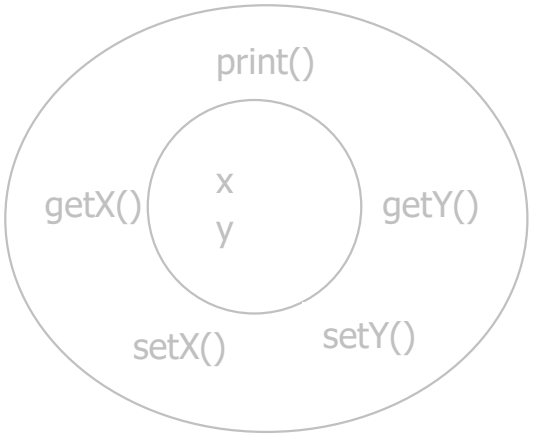
Primer 1 – definicija razreda

// Point.h

```
class Point {
private:
    int x, y;
public:
    int getX();
    int getY();
    void setX(int x);
    void setY(int y);
    void print();
};
```

UML notacija





Primer 1 – implementacija razreda

// Point.h

```
class Point {
private:
    int x, y;
public:
    int getX();
    int getY();
    void setX(int x);
    void setY(int y);
    void print();
};
```

// Point.cpp

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

int Point::getX() {
    return x;
}
int Point::getY() {
    return y;
}
void Point::setX(int x) {
    this->x=x;
}
void Point::setY(int y) {
    this->y=y;
}
void Point::print() {
    std::cout << "(" << x << "," << y << ")" << std::endl;
}
```

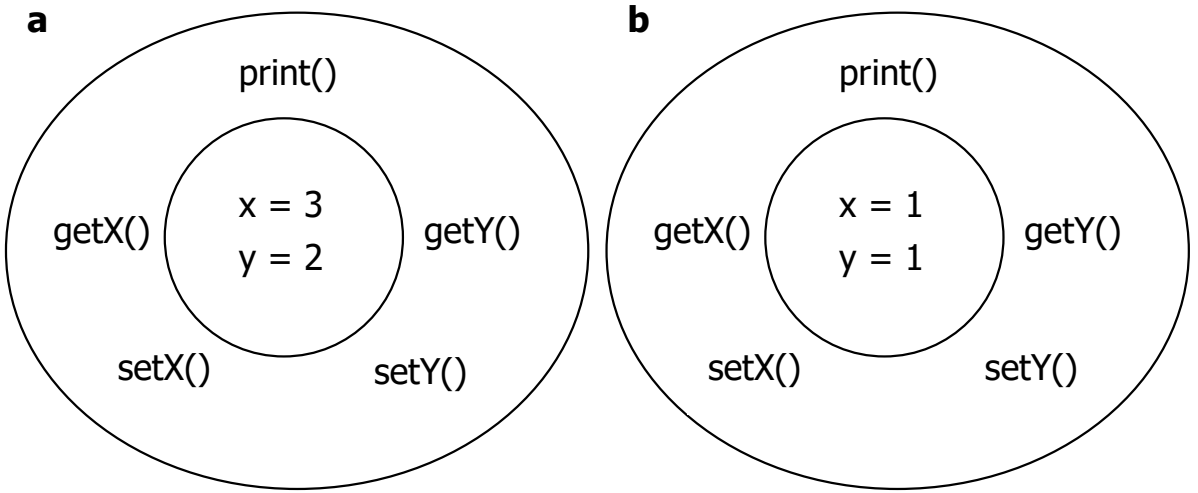
Primer 1 – ustvarjanje objektov

// Example01.cpp

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

int main() {
    Point a;
    Point b;
    a.setX(3);
    a.setY(2);
    b.setX(1);
    b.setY(1);
    std::cout << a.getX() << std::endl;
    std::cout << a.getY() << std::endl;
    //b.x=10;
    a.print();
    b.print();
    return 0;
}
```

UML notacija

a:Point	b:Point
x : Integer = 3 y : Integer = 2	x : Integer = 1 y : Integer = 1



Dobro je vedeti

- Skrivanje komponente (data hiding) razreda dosežemo z določilom **private**.
- Komponento izvozimo z določilom **public**.
- Vse komponente, ki so skrite lahko načrtovalec razreda poljubno spreminja ne da bi to vplivalo na programsko kodo, ki uporablja ta razred.
- Ograjevanje (kapsuliranje) je eno izmed temeljev OOP, saj omogoča enostavnejšo spreminjanje in vzdrževanje programov.

Dobro je vedeti

- Kratke metode lahko zapišemo kar v definicijo razreda (vključitvena datoteka)
- Prevajalnik jih bo obravnaval kot vrinjene (*inline*) metode.

// Point.h

```
class Point {
private:
    int x, y;
public:
    int getX() {return x;}
    int getY() {return y;}
    void setX(int x) {this->x=x;}
    void setY(int y) {this->y=y;}
    void print();
};
```

Dobro je vedeti

- Razred lahko definiramo tudi s ključno besedo **struct**.
- Če razred definiramo s **struct** in ne podamo določil (**private**) so vse komponente javne.
- Če razred definiramo s **class** in ne podamo določil (**public**) so vse komponente skrite.

// Point.h

```
struct Point {
    int getX();
    int getY();
    void setX(int x);
    void setY(int y);
    void print();
private:
    int x, y;
};
```

Konstruktorji in destruktorji

- Potrebujemo mehanizem za inicializacijo in brisanje objekta.
- Konstruktor je metoda, ki se pokliče ob kreiranju objekta in rezervira pomnilniški prostor ter inicializira podatke.
- Destruktor je metoda, ki se pokliče ob brisanju objekta in sprosti pomnilniški prostor.
- Življenjska doba objekta!

// Point.h

```
class Point {
private:
    int x, y;
public:
    Point();                  // default constructor
    Point(const Point& t);    // copy constructor
    Point(int xy);            // conversion constructor
    Point(int x, int y);      // other constructor
    ~Point();                 // destructor
    // methods
    int getX();
    int getY();
    void print();
    double distance(Point t);
};
```

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Konstruktorji in destruktorji

- Konstruktor in destruktor imata isto ime, kot je ime razreda.
- Destruktor ima pred imenom še znak ~ (tilda).
- Konstruktorji in destruktorji ne vračajo vrednosti in ne smejo imeti definiranega tipa metode (niti tipa **void**).
- Dinamično rezervirani pomnilniški prostor (**new**) moramo sprostiti sami (**delete**), ostali se sprosti avtomatsko.
- Po principu prekrivanja funkcij (*overloading*) lahko definiramo več konstruktorjev, a le en destruktor.

Konstruktorji in destruktorji

- Privzeti konstruktor je konstruktor brez argumentov.
- Kopirni konstruktor tvori objekt iz že obstoječega. Njegov argument je referenca na že obstoječ objekt tega razreda.
- Pretvorbeni konstruktor tvori nov objekt iz drugega podatkovnega tipa (razreda).
- Ostali konstruktorji imajo drugačne argumente in nimajo posebnega imena.
- Prevajalnik priskrbi privzeti* in kopirni konstruktor ter destruktor, če ga ne zapiše programer.

// Point.h

```
class Point {
private:
    int x, y;
public:
    Point();                // default constructor
    Point(const Point& t);  // copy constructor
    Point(int xy);          // conversion constructor
    Point(int x, int y);    // other constructor
    ~Point();               // destructor
    // methods
    int getX();
    int getY();
    void print();
    double distance(Point t);
};
```

Primer 2 – implementacija konstruktorjev

// Point.h

```
class Point {
private:
    int x, y;
public:
    Point();                // default constructor
    Point(const Point& t);  // copy constructor
    Point(int xy);          // conversion constructor
    Point(int x, int y);    // other constructor
    ~Point();               // destructor
    // methods
    int getX();
    int getY();
    void print();
    double distance(Point t);
};
```

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

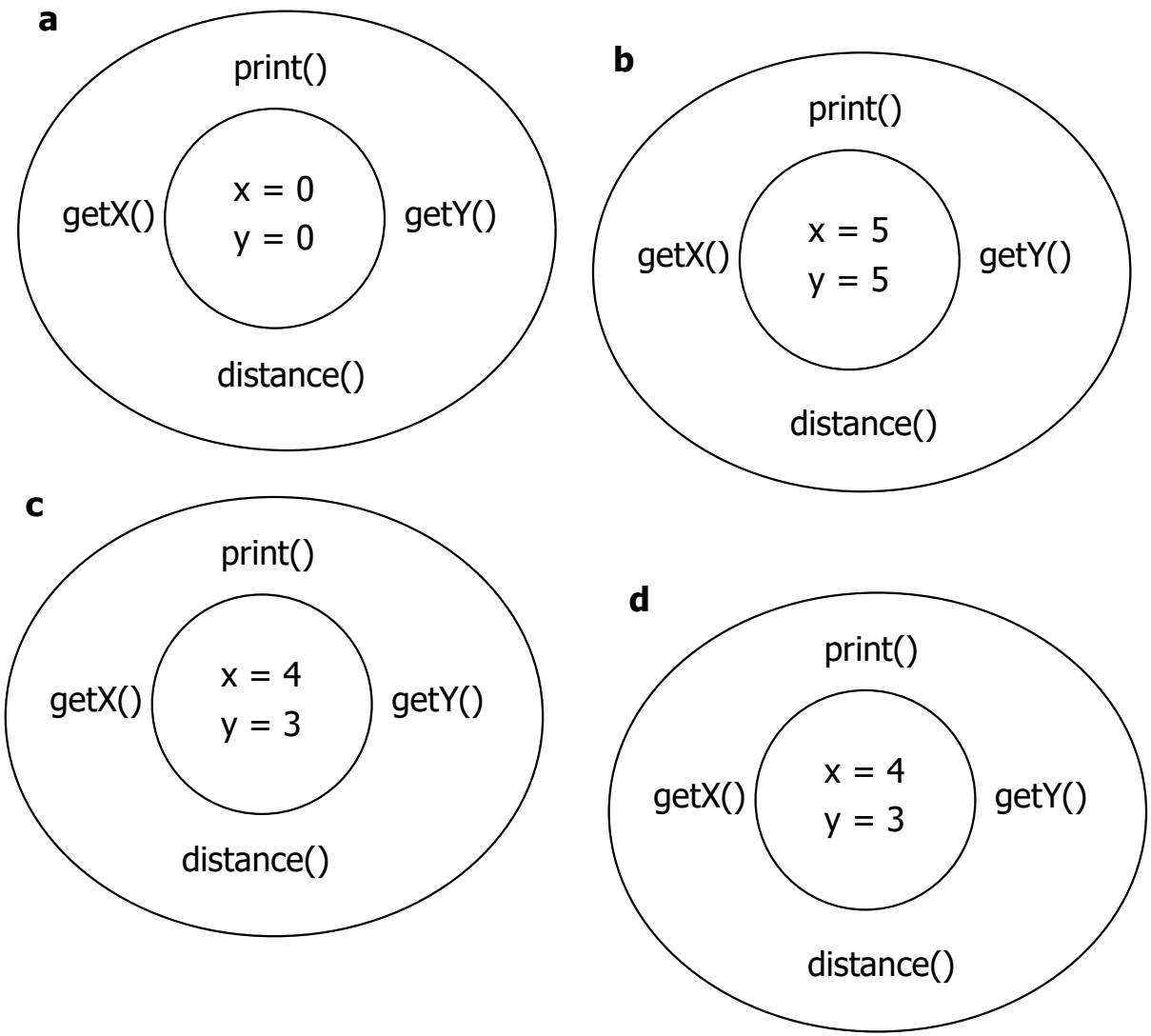
// Point.cpp

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

Point::Point() : x(0), y(0) {
}
Point::Point(const Point& t) : x(t.x),  y(t.y) {
}
Point::Point(int xy) : x(xy), y(xy) {
}
Point::Point(int x, int y)  {
    this->x=x;
    this->y=y;
}
Point::~Point() {
}
int Point::getX() {
    return x;
}
int Point::getY() {
    return y;
}
void Point::print() {
    std::cout << "(" << x << ", " << y << ") " << std::endl;
}
double Point::distance(Point t) {
    return std::sqrt((double)(x - t.x)*(x - t.x)+(y - t.y)*(y -
t.y));
}
```

Primer 2 – implementacija konstruktorjev

// Example02.cpp

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

int main() {
    Point a;
    Point b(5);
    Point c(4,3);
    Point d(c);
    a.print();
    b.print();
    c.print();
    d.print();
    std::cout << a.distance(c) << std::endl;
    //std::cout << a.distance(5) << std::endl;
    return 0;
}
```



Primer 2 – implementacija konstruktorjev

- Konstruktorji z inicializacijskim seznamom (*initialization list*)
- Za glavo metode zapišemo dvopičje in seznam instančnih spremenljivk, ki dobijo vrednosti, zapisane v oklepaju.
- Način zapisa z inicializacijskim seznamom je implementacijsko učinkovitejši!

// Point.cpp

…

```
Point::Point() : x(0), y(0) {
}
Point::Point(const Point& t) : x(t.x),  y(t.y) {
}
Point::Point(int xy) : x(xy), y(xy) {
}
Point::Point(int x, int y) : x(x), y(y) {
}
```

…

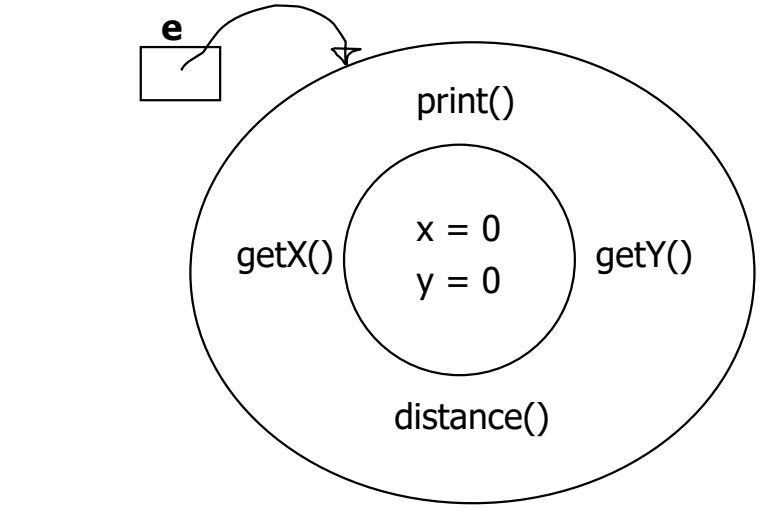
Primer 2 – implementacija konstruktorjev

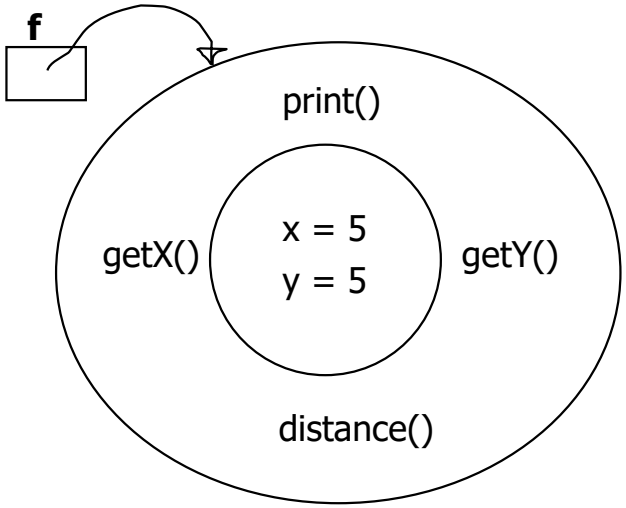
- Alokacije:
  - statična,
  - avtomatična,
  - dinamična.
- Dinamična alokacija objektov:
  - konstruktor se pokliče ob operatorju **new**
  - destruktor se pokliče ob operatorju **delete**

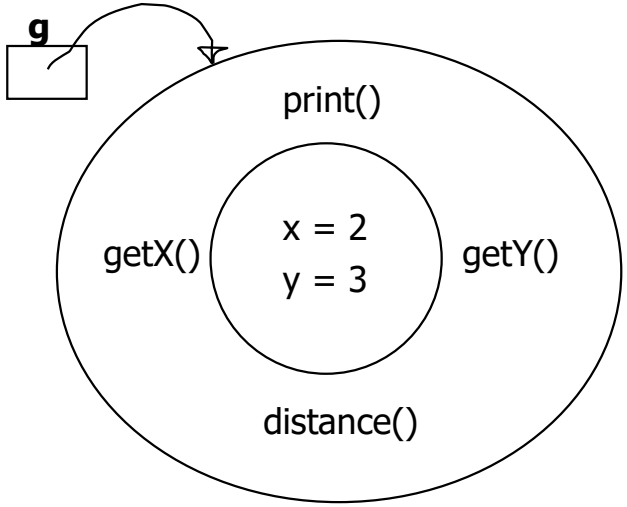
// Example02.cpp

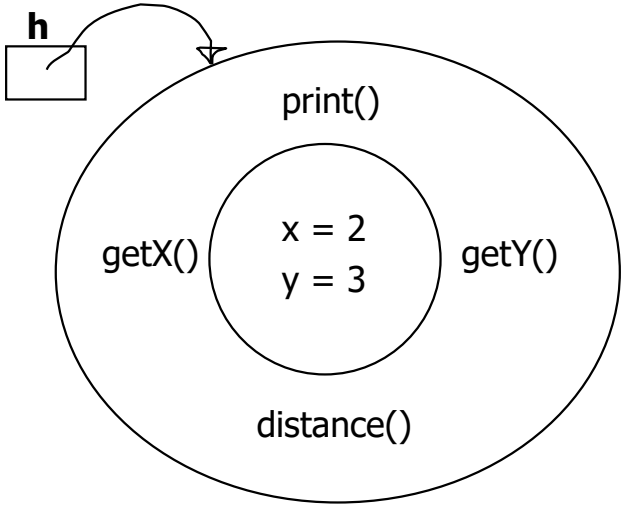
```
...

std::cout << "Dynamic allocation" << std::endl;
Point* e = new Point();
Point* f = new Point(5);
Point* g = new Point(2,3);
Point* h = new Point(*g);
e->print();
f->print();
g->print();
h->print();
delete e;
delete f;
delete g;
delete h;


...
```

Primer 2 – implementacija konstruktorjev









Pravila dobrega programiranja

- Dovolj kratke metode zapišemo znotraj definicije razreda.
- Za vidnost komponent vedno navedimo določili **public** in **private**.
- Zaradi večje preglednosti v razredu navedimo določili **public** in **private** le enkrat.
- **Če kopirni konstruktor in destruktor ne opravljata nobenega dodatnega dela ga naj priskrbi prevajalnik! JIH NE PIŠEMO SAMI**

Pogoste napake programerja

- Razred ali struktura se ne zaključi s podpičjem.
- Definicija izhodnega tipa za konstruktor ali destruktor.
- Vračanje vrednosti iz konstruktorja ali destruktorja s stavkom return.
- Ne definiramo privzetega konstruktorja, če smo definirali druge konstruktorje.
- Definicija destruktorja z argumenti.
- Doseganje privatnih komponent razreda.

Vprašanja