



# C++:n pikaopetus

*Katsaus: C++ Javan näkökulmasta*

Ville Leppänen

# Sisältö

- Samanlaisuudet
- Eroavuudet
- Kielen rakenteiden läpikäynti
- C++:n erityispiirteitä.
- Esimerkkejä.
- Muutama sana STL:stä.

# Luonnehdintoja

Kai Koskimies: "maailmassa on todennäköisesti vain kourallinen ihmisiä, jotka osaavat C++:n täydellisesti".

**Erot aika pieniä, tyyppisysteemistä ja muistin näkyvyydestä suurimmat erot.**

# teht12.cc; Esan kurssista

```
/*
   TKO_5533 C++
   demo 1, tehtavan 2 ratkaisu
*/

#include <iostream>
#include <limits>

using namespace std;

double polynom(double x, double a0, double a1 = 0.0,
               double a2 = 0.0, double a3 = 0.0);

int mini4(int a, int b = numeric_limits<int>::max(),
          int c = numeric_limits<int>::max(),
          int d = numeric_limits<int>::max());
```

# teht12.cc; Esan kurssista

```
int main()
{
    double d = polynom(2.0, 6.0, 2.0);
    cout << d << endl;

    cout << numeric_limits<int>::max() << endl;
    cout << numeric_limits<int>::min() << endl;
    cout << mini4(1, 2, 3, 4) << endl;
    cout << mini4(2, 1, 3, 4) << endl;
    cout << mini4(2, 3, 1, 4) << endl;
    cout << mini4(2, 3, 4, 1) << endl;
}

double polynom(double x, double a0, double a1, double a2, double a3)
{
    return a0 + x * (a1 + x * (a2 + x * a3));
}
```

# teht12.cc; Esan kurssista

```
int mini4(int a, int b, int c, int d)
{
    int min = a;

    if (b < min)
        min = b;

    if (c < min)
        min = c;

    if (d < min)
        min = d;

    return min;
}
```

# teht32.h; Esan kurssista

```
#include <iostream>
#include <cstdlib>
/* Kokonaislukuja sisältävä binääripuu. */
class BinIntTree {

protected:
    /* Puun tietoaalkio (kokonaisluku). */
    int item;

    BinIntTree* left; /* Vasen jälkeläinen. */

    BinIntTree* right; /* Oikea jälkeläinen. */

    BinIntTree* parent; /* Vanhempi. */
```

# teht32.h; Esan kurssista

```
public:
```

```
    /* Alustaa uuden puun, jonka ainoaksi tietoalkioksi asetetaan  
       argumenttina saatu kokonaisluku. */  
    BinIntTree(int first_item) {  
        item = first_item;  
        left = 0;  
        right = 0;  
        parent = 0;  
    }  
  
    /* Destruktori. */  
    ~BinIntTree() {  
        if (left != 0) delete left;  
        if (right != 0) delete right;  
    }
```

# teht32.h; Esan kurssista

```
/* Lisää puuhun argumenttina saadun alkion. */  
void insert(int new_item) {  
    BinIntTree* new_tree = new BinIntTree(new_item);  
    this->rec_insert(new_tree);  
}  
  
... poistettu metodeja ...  
  
/* Onko nykyinen puu lehtisolmu? */  
bool is_leaf() {  
    return (left == 0) && (right == 0);  
}  
  
};
```

# teht33.c; Esan kurssista

```
/*
   TKO_5533 C++
   demo 3, tehtävän 3 ratkaisu
*/

#include <iostream>

class Vertailtava {
public:
    virtual bool operator<(Vertailtava& toinen) = 0;

    virtual bool operator>=(Vertailtava& toinen)
    {
        return !(*this < toinen);
    }
}
```

# teht33.c; Esan kurssista

```
virtual bool operator<=(Vertailtava& toinen)
{
    return !(toinen < *this);
}

virtual bool operator>(Vertailtava& toinen)
{
    return !(*this <= toinen);
}

virtual bool operator==(Vertailtava& toinen)
{
    return (*this <= toinen) && (*this >= toinen);
}
};
```

# teht33.c; Esan kurssista

```
class Ympyra : public Vertailtava {
private:
    double r;
public:
    Ympyra(double sade) : r(sade) {}
    double anna_sade() { return r; }
    bool operator<(Vertailtava& toinen)
    {
        return (r < static_cast<Ympyra&>(toinen).anna_sade());
    }
};
```

# teht33.c; Esan kurssista

```
int main()
{
    Ympyra y(10.0);
    Ympyra yy(20.0);

    std::cout << "ympyran y sade on " << y.anna_sade() << "\n";
    std::cout << "ympyran yy sade on " << yy.anna_sade() << "\n";

    std::cout << "ympyroiden vertailu yy < y: " << (yy < y) << "\n";
    std::cout << "ympyroiden vertailu yy >= y: " << (yy >= y) << "\n";
}
```

# Samanlaisuudet 1/2

- Molemmat OO-kieliä, joissa OO:n tavalliset ominaisuudet: kapselointi, perintä, polymorfismi, . . .
- Luokat ovat konstruktio omien tyyppien luomiseksi kummassakin.
- Kummassakin lauseet ovat muodoltaan hyvin samanlaisia.
- Lohkon sisällä paikallisia muuttujia voi esitellä missä kohtaa vain.
- Molemmissa metodien ylikuormitus.

# Samanlaisuudet 2/2

- Molemmissa konstruktorit.
- Vahvasti tyypitettyjä.
- Molemmissa poikkeukset.
- Metodien muoto suurin piirtein sama.

# Eroavuuksia 1/4

- C++:ssa *osoittimet*, Javassa ei.
- Java ei tue oletusparametreja (argumentteja).
- Javan tyyppikonstruktiot ovat luokka ja taulukkokonstruktio; C++:ssa on muitakin: struct, union, ...
- Javassa ei ole operaattorien ylikuormitusmahdollisuutta (osa on valmiiksi ylikuormitettu).
- Javaan ei liitty esiprosessoria eikä vastaavasti sen direktiivejä.

# Eroavuuksia 2/4

- Javassa kaikki automaattiset tyyppikonversiot ovat turvallisia esitystarkkuuden mielessä.
- C++:ssa kaikkien muuttujien ja metodien ei tarvitse sijaita jonkin luokan sisällä: globaaleja muuttujia ja metodeita.
- Java ei tue moniperintää, C++ tukee.
- C++: destruktorit ja delete-operaattori.
- Javassa automaattinen roskien keruu; C++: osittain käyttäjän vastuulla.
- Javassa ei ole luokkamalleja, geneerisyyttä.

# Eroavuuksia 3/4

- Luokkien muoto on aika sama, C++: lopussa puolipiste.
- C++:ssa käytettävät luokat ja metodit esitellään ennen käyttöä (header-tiedostot).
- char on C++:ssa 8-bit, Javassa 16-bit.
- C++:ssa perustyyppien muuttujia ei automaattisesti alusteta, vaan sisältö on mitä sattuu.
- Javassa kaikki olioiden käsittely tapahtuu viitteen kautta. C++:ssa oliot voivat olla myös ns. *laajennettua tyyppiä*: koko sisältö on muuttujan arvona!

# Eroavuuksia 4/4

- *C++-kielestä on olemassa useita versiota!*  
Periaatteessa kukin kääntäjä määrittää kielen.  
Javassa kielen rakenne pysynyt paremmin hallinnassa.

# Kielen rakenteiden läpikäynti

- Ohjelmien tekemisestä.
- Lauseet.
- Metodi.
- Luokka ja header-tiedostot.
- Tyypit ja muuttujat.
- Lausekkeet.
- (I/O, merkkijonojen käsittely)

# Ohjelmien tekeminen 1/2

- Kääntäjä (gcc, g++), linkittäjä ja esiprosessori.
- Kirjoitetaan joukko header-tiedostoja (loppuliitteet: .h, .hpp).
- Kirjoitetaan joukko C++-ohjelmätiedostoja (.cpp, .cxx).
- Käännetään ohjelmätiedostot binääriksi (.obj, .o).
- Linkitetään käännetyt binäärit suorituskelpoiseksi konekieliseksi tiedostoksi (.exe, ...).

# Ohjelmien tekeminen 2/2

- Kääntämisen yhteydessä esiprosessori sisällyttää header-tiedostoja ohjelmatiedostoihin ja muutenkin prosessori header- ja ohjelmatiedostojen sisältöä niiden direktiivien osalta (korvauksia, koodin poistoa, ...).

- INCLUDE-polulta

```
#include <otsikko>           (uusi; nimettyyn)  
#include <otsikko.h>        (nimettömään)
```

- Samasta hakemistosta

```
#include "otsikko.h"
```

# Sisäkkäinen include?

## MoboMemEngineInterface.h:

```
// MoboMemEngineInterface.h
// (c) Antti Juustila, 2004

#ifndef __MOBOMEMENGINEINTERFACE_H
#define __MOBOMEMENGINEINTERFACE_H

#include <e32base.h>
#include <e32std.h>
#include <badesca.h>
#include "MoboMemDefines.h"

    ... poistettu ...

#endif
```

# Lauseet 1/5 (hyvin samanlaisia)

- Lohkolause: täysin samanlainen.
- `if`: täysin samanlainen, paitsi ehtolauseen tyyppin osalta.
- Tyyppi `bool` ja arvot `true` (1) ja `false`. `bool` ja `int` konvertoitavissa toisikseen.  
*if:n ehto halutaan int-tyyppisenä!*  
Arvo nolla on epätosi, muut tulkitaan todeksi!  
Samoin muut lauseet!

Klassikko:

```
int luku = 0;  
if (luku == 10) ... // epätosi  
if (luku = 10) ... // tosi
```

# Lauseet 2/5

- switch: sama syntaksi ja toiminta; haarautumisehdon tyyppin täytyy mukautua kokonaisluvuksi.
- return: täysin samanlainen.
- break, continue: samanlaisia, sama käyttötarkoitus; useiden silmukoiden rikkominen?? (ei?)
- goto (???): Älä käytä!
- while: sama, paitsi ehtolausekkeen tulkinta.
- do-while: sama, paitsi ehdon tulkinta.
- for: käytännössä täysin sama.

# Lauseet 3/5

- Muuttujien esittely: lähes identtinen.  

```
bool hasTable = true;
```
- Asetuslause: samanlainen. Erot tulevat lähinnä mahdollisista muuttujien tyyppi-ilmauksista.
- delete ja new-lauseke: new:lla luodut oliot ns. dynaamisiin muuttujiin tulee tuhota tällä.  
(Varaa keosta.)

# Lauseet 4/5

```
tyyppi *muuttuja = new tyyppi;
```

```
...
```

```
delete muuttuja; // sisältö
```

```
tyyppi *taulukko = NULL;
```

```
taulukko = new tyyppi[lkm];
```

```
...
```

```
delete [] taulukko;
```

```
taulukko = NULL;
```

# Lauseet 5/5

```
tyyppi **taulu2D;  
taulu2D = new tyyppi*[lkm1];  
...  
for (ind=0; ind<lkm1; ind++)  
    taulu2D[ind] = new tyyppi[lkm2];  
...  
for (ind=0; ind<lkm1; ind++)  
    delete [] taulu2D[ind];  
delete [] taulu2D;  
taulu2D = NULL;
```

# Metodi / aliohjelma 1/8

- ★ Metodin muoto lähes sama kuin Javassa.

```
tyyppi nimi(parametritluettelo)
{
    määrittelyitä;
    lauseita;
}
```

- ★ Tavallinen nimi: globaali tai osa luokkaa.  
Luokka::nimi kertoo, että osa luokkaa  
Luokka.  
(Luokkien toteutus voi olla irti niiden  
määrittelystä.)

# Metodi / aliohjelman 2/8

- ★ Metodit esiteltävä ennen käyttöä, ns. aliohjelman prototyypillä:

```
void Ali();  
int main() { Ali(); return 0; }  
void Ali() {  
    cout<<"Kokeilu!";  
} // Ali
```

- ★ Prototyyppejä:

```
void Ali(void);  
int AliB(int, int);
```

# Metodi / aliohjelman 3/8

- ★ *Parametrit* voivat olla arvo-, osoitin- tai viittausparametreja:

```
void A(int a) { a = a * a; }
```

```
void B(int *b) { *b = (*b) * (*b); }
```

```
void C(int &c) { c = c * c; }
```

kutsu:

```
int x = 10;
```

```
A(x); // x = 10
```

```
B(&x); // x = 100.
```

```
C(x); // x = 10000.
```

# Metodi / aliohjelman 4a/8

★ *Tietuetyyppinen parametri* samoin:

```
struct PVM { int pp, int kk, int vv; }
```

```
// tyyppi:void A(PVM)
```

```
void A(PVM a) { a.pp = 10; }
```

kutsu:

```
PVM x = ...
```

```
A(x);
```

# Metodi / aliohjelman 4b/8

```
// tyyppi: void B(PVM *)  
void B(PVM *b) {  
    (*b).kk = 3;    // sama kuin  
    b->kk = 3;  
}
```

```
// tyyppi: void c(PVM &)  
void C(PVM &c) { c.vv = 3; }
```

kutsu:

```
PVM x = ...
```

```
B(&x);
```

```
C(x);
```

# Metodi / aliohjelman 5/8

- ★ Tuloksen tyypit samoin!
- ★ Taulukko parametrina: osoitinparametri; taulukko on osoite 1. alkioon.

```
void alusta(int[], int);  
int main() {  
    int taulu[5];  
    alusta(taulu, 5);  
    ...  
}  
void alusta(int t[], int lkm) {  
    for (int i=0; i<lkm; i++) t[i] = 0;  
}
```

# Metodi / aliohjelma 6/8

- ★ Vakioparametri: const määrittelyyn.
- ★ (Vaihtuva määrä parametreja.)
- ★ Oletusarvot:

```
void A(int a = 10) { ... }
```

Oletusparametrit loppuun; ei tarvitse esiintyä kutsussa.

(Automaattinen (yli)kuormitus.)

- ★ Kuormitus kuten Javassa.
- ★ (inline-aliohjelmat)

# Metodi / aliohjelma 7a/8

- ★ main-metodin parametrit:

```
int main(int argc, char **argv)
{ ... }
```

- ★ (*Aliohjelmaan voi olla osoite.*)
- ★ Rekursiiviset aliohjelmat: kuten Javassa, mutta pitää esitellä ennen määrittelyä.
- ★ Aliohjelmat ovat joko luokassa tai globaaleja. Samoin voi olla globaaleja muuttujia.

# Metodi / aliohjelman 7b/8

## ★ Staattiset paikalliset muuttujat

```
void A() {  
    static int x = 1;  
    cout<<x;  
    x++;  
}
```

Tulostaa eri kutsukerroilla eri arvon: 1, 2, ...

# Metodi / aliohjelma 8/8

- ★ extern-määre: viitataan muualla toteutettuun globaaliin muuttujaan / aliohjelmaan. (mielummin #include)
- ★ *Nimiavaruus*: joukko tunnuksia (muuttuja, aliohjelma, luokka).

```
using namespace std;  
namespace oma {  
    void A(int);  
    int luku;  
}
```

```
using namespace oma;
```

# Lausekkeet

- ★ Enimmäkseen samanlaisia.
- ★ Tyyppiyhteensopivuus mielessä int ja bool yhteensopivia molempiin suuntiin.
- ★ Erityinen muistipaikan osoitteen selvittämisooperaattori: `&x`.
- ★ Osoitin tyyppisen muuttujan sisältö: `*y`.
- ★ Taulukon alkio: `t[10]`.
- ★ Kenttä: `x.k` tai `*y.k` eli `y->k`.
- ★ Luokan ominaisuus `Luokka::luku`.
- ★ Globaalimuuttuja `::luku`.

# Luokat 1/7

- ★ Erikseen *määrittely ja toteutus*.
- ★ Määrittely:

```
class Henkilo {
    private:
        int ika;
        char[25] nimi;
        Sotu *sotu;
    public:
        void asetaSotu(Sotu *);
        virtual void asetaIka(int);
        ...
};
```

## ★ Toteutus

```
void Henkilo::asetaiIka(int i)
{ ika = i; }
...
```

- ★ Määre virtual: mahdollistaa dynaamisen sidonnan. C++:ssa oletuksena staattinen sidonta.

# Luokat 3/7

- ★ Määre static: kuten Javassa.
- ★ **Luonti:**
  - ★ Automaattinen; pinosta; paikallinen muuttuja  
`Henkilo h = Henkilo();`
  - ★ Dynaaminen; keosta;  
`Henkilo *g = new Henkilo();`
  - ★ Staattinen; globaali; static; varataan suorituksen alussa.

## ★ Muodostimet ja hajoittimet

```
PVM::PVM( ) { ... }  
PVM::PVM(int p, inv k)  
  { ... }  
  
PVM::~~PVM( )  
  { ... } // hajoitin
```

# Luokat 5/7

## ★ Kopiointimuodostimen käyttö:

```
PVM uusi(old); // old autom.  
PVM uusi = old;  
PVM uusi(*o); // o dyn.  
PVM uusi = *o;
```

Voi tehdä itse; kääntäjä gen. autom.

## ★ Suojausmääreet public, protected ja private kuten Javassa.

# Luokat 6/7

## ★ *Aliluokka:*

```
class Yli { ... };
```

```
class Ali :tapa Yli { ... };
```

Tapa vaikuttaa siihen, miten perittävät näkyvät.

- ★ Mitä peritään: kuten Javassa. (Moniperintä.)
- ★ Uudelleen määrittely mahdollista.
- ★ Konstruktorien suoritusjärjestys: kuten Javassa.

# Luokat 7/7

- ★ Erikoistulkinta, down cast:

```
T *os = dynamic_cast<T *>(olioos);
```

- ★ Puhdas virtuaalinen metodi: runkona '= 0'.  
Abstrakti luokka, jos yksikin sellainen.
- ★ friend: yksityisten osien käyttöoikeus toiselle luokalle.

# Tyypit (ja muuttujat) 1/6

- ★ int, short, long, float, double, long double, bool, char, void  
Lisäksi unsigned xx
- ★ Merkkijonot string tai char[pituus].  
Merkkijonon lopetusmerkki '\0.'
- ★ Lueteltu tyyppi enum

```
enum paivat {MA = 1, TI, KE,  
            TO, PE, LA, SU };
```

```
...  
paivat muuttuja = MA;
```

Aina int-koodaus, oletus: nollasta.

# Tyypit (ja muuttujat) 2/6

## ★ Tietuetyyppi struct

```
struct PankkiTili {  
    string omistajanNimi;  
    long koodi;  
    float saldo;  
};
```

```
PankkiTili p =  
    { "Ville", 898733235, 21.2 };
```

*Luokkaan verrannollinen.*

# Tyypit (ja muuttujat) 3/6

## ★ oneof-rakenne union

```
union Tyontekija {  
    Pomo pomo;  
    Assari assari;  
  
    ...  
};
```

# Tyypit (ja muuttujat) 4/6

★ Taulukko: yksi- tai useampi dim.

```
int x[5] = {1, 2, 3, 4, 5};  
int luvut[2][3] =  
    { {1, 2, 3}, {4, 5, 6} };  
int y[5] = {1}; // muut 0
```

Jokaiselle alkiolle varataan tilaa.

Javasta poiketen eivät ole olioita, { ... }  
on alustus, ei luonti.

Kokoarvot vakioita; #define.

Viittaukset normaalisti indeksointien avulla.

# Tyypit (ja muuttujat) 5/6

## ★ *Osoitintyyppi:*

```
int *x = NULL;  
Henkilo *y = 0;  
Auto a;
```

```
int y = 0;  
x = &y; // x:n sisältönä y:n os.  
*x = 7; // y muuttuu
```

**Luokkien mukaiset muuttujat eivät oletusarvoisesti osoitintyyppisiä!**

★ (Alias; `int &x = y;`)

# Tyypit (ja muuttujat) 6/6

- ★ `void`
- ★ `typedef int ii;` (synonyymi)
- ★ Lauseke `sizeof(tyyppi)` kertoo kyseistä tyyppiä olevan muuttujan tarvitseman tilan tavuina.  
`sizeof lauseke;`

# Lopuksi

- ★ <http://www.nic.funet.fi/c++opas/>
- ★ Ks. kirjoista.
- ★ Ks. Esan C++-kurssin kotisivulta
- ★ ...