Ohjelmoinnista yleensä

Ohjelma

Modernien laitteiden toiminta perustuu niitä ohjaavien ***ohjelmien*** (engl. *program*) käyttöön. Tällä kurssilla keskitytään yleiskäyttöisten tietokoneiden ohjelmien tekoon. Esimerkiksi autoissa ja pankkiautomaateissa on toisenlaisia ohjelmia, joita kutsutaan sulautetuiksi järjestelmiksi. Niissä ohjelma on kiinteä osa järjestelmää ja toimii vain yhdellä tavalla laitteen ohjaamiseksi.

Tietokoneissa, puhelimissa ja tableteissa on ***sovelluksia*** (engl. *application*), jotka tarjoavat käyttäjälle palveluja tiettyyn aihepiiriin liittyvien tehtävien suorittamiseksi.

Muunlaisia ohjelmia kuin sovelluksia ovat yleisluontoiset ***järjestelmäohjelmat*** (engl. *system software*) kuten käyttöjärjestelmät ja ajuriohjelmat (engl. *drivers*), jotka hallinnoivat laitteiston (engl. *hardware*) käyttöä.

Ohjelmoijan tahtävänä on määrittää, miten tietokoneen halutaan toimivan ohjelmaa ajettaessa. Esim. ”Jos käyttäjä painaa nappia, lisää kyseinen käyttäjä kurssille ilmoittautuneiden listaan."

Sovellusohjelman käyttäjällä on käytössään kopio ohjelmasta. Kun hän käynnistää ohjelman, ohjelma ***ajetaan*** eli ***suoritetaan*** (engl. *run*, *execute*). Tällöin tietokone noudattaa sille ohjelmassa ennalta määrättyjä käskyjä. Kun tietokoneella on tallessa sovellusohjelma, koneen voi määrätä suorittamaan tuon ohjelman. Käyttäjä voi antaa tietokoneen käyttöjärjestelmälle käskyn (esim. kuvaketta klikkaamalla), joka käynnistää sovelluksen. Tällöin tietokone toimii niin kuin kyseinen ohjelma määrää, ja jotain ohjelman käyttäjän kannalta hyödyllistä tapahtuu.

Sanalla "ohjelma" voidaan viitata kumpaan tahansa seuraavista:

- Ohjelmateksti eli ***ohjelmakoodi***, joka kuvaa sarjan käskyjä tietokoneen suoritettavaksi.
- ***Ohjelman suoritus*** eli ne toimenpiteet, jotka tietokone käy läpi, kun se on määrätty suorittamaan ohjelmakoodin sisältämät käskyt.

Ohjelmointi

Ohjelmointi on tietokoneen toimintaohjeiden laatimista, luovaa ongelmien ratkaisemista. Itseltä kysyttävä kysymys on:

"Millainen ohjelma on laadittava, jotta saadaan mahdollisimman hyvä ratkaisu ongelmaan?"

Ensin pitää miettiä, millaisia ratkaisuja ongelmalle yleensäkään on ja seuraavaksi, miten ratkaisumenetelmä – algoritmi – kuvataan tietokoneelle.

Ohjelmointi on siinä mielessä kiinnostavaa ja haastavaa, että se edellyttää mm. ongelmanratkaisukykyä, logiikkaa, luovuutta ja huolellisuutta. Luovuutta ja logiikkaa siis samassa päässä. Ei ihan joka pojan (tai tytön) hommaa mutta perusteet voi oppia jokainen. Pari perusasiaa kun muistaa, niin pääsee hyvin alkuun.

- Prosessori tekee vain tasan sen, mitä käsketään.
- Se suorittaa käskyjä yksitellen, järjestyksessä.
- Prosessori ei ymmärrä mitään käskyjen merkityksestä ohjelmakokonaisuudessa.
- Se ei osaa päätellä, arvioida eikä soveltaa.

Tietokone on siis "tyhmä". Se on kuitenkin nopea, tarkka ja hyvämuistinen.

Ohjelmaa kirjoitettaessa täytyykin luoda yksityiskohtainen, kirjaimellinen kuvaus siitä, mitä tietokoneen halutaan tekevän, kun ohjelma ajetaan. Jotta tämä onnistuisi, on käytettävien ilmaisujen oltava *täsmällisiä* ja *yksiselitteisiä*. Ei siis malliin "puhu kovempaa" tai "lisää maitoa". Kuinka paljon kovempaa? kuinka paljon lisää? Tämän vuoksi ohjelmakoodi kirjoitetaan jollakin ***ohjelmointikielellä***. Ohjelmointikielessä on rajattu määrä hyvin määriteltyjä sanoja ja ilmaisutapoja.

Ohjelmointi ongelmien ratkaisemisena

Ohjelmoijan työ lähtee tyypillisesti liikkeelle ongelmasta tai tarpeesta, johon etsitään ratkaisua. Alla on muutamia esimerkkejä (enimmäkseen varsin haastavista) realistisista ongelmista, joita voidaan ratkoa tietokoneen avulla:

- Halutaan puhua puhelimella Internetin yli toiselle puolelle maapalloa,
- on saatava yritykselle uudenlainen laskutussovellus,
- halutaan tarjota käyttäjälle mahdollisimman hyviä muita ostossuosituksia, kun hän ostaa tietyn tuotteen verkkokaupasta,
- tarvitaan työkalu, jolla voidaan muokata mittausdataa sopivaan muotoon jatkokäsittelyä varten,
- halutaan simuloida alkeishiukkasten liikettä ilmiöiden ennustamiseksi,
- halutaan simuloida tunteikkaiden siivekkäiden liikettä painovoimakentässä kohti sorkkaeläimiä,
- ja niin edelleen. Ja niin edelleen.

Tuttuihin ongelmiin voi etsiä aina aiempaa parempia vastauksia, ja uusia kiinnostavia ongelmia luodaan koko ajan sekä ohjelmoijien itsensä toimesta että muiden.

Algoritmit

Ohjelmoijat puhuvat usein algoritmeista (algorithm). Lyhyesti kuvailtuna algoritmi on vaiheittainen ratkaisumenetelmä jollekin ongelmalle. Leivonnankin puolella on algoritmeja, vaikka termiä ei siellä käytetäkään: esimerkiksi mestarileipurin idea siitä, miten kakku pitäisi tehdä, on hänen keksimänsä algoritmi. Hän kirjoitti tämän ajatuksensa muistiin reseptitekstiksi. Vastaavasti ohjelmoija toteuttaa algoritmeja — ongelmien ratkaisuja — ohjelmakoodiin.

Osaongelmien ratkaisuiksi voidaan käyttää eri algoritmeja. Esimerkiksi tekstinkäsittelyohjelmassa voidaan käyttää yhtä algoritmia oikolukuun ja toista etsimistoiminnon toteuttamiseen. Ohjelmoija voi keksiä osan käyttämistään algoritmeista itse ja soveltaa muiden kehittämiä algoritmeja joihinkin osaongelmiin. Tavallisesti ainakin tapa, jolla algoritmeja tietyssä uudessa ohjelmassa yhdistellään, on ohjelmoijan itse kehittämä.

Ohjelmointikieliä

Aikojen alussa tietokoneet ohjelmointiin konekielellä (machine code). Konekieli on se kieli, jota tietokone ymmärtää. Windows-koneiden konekieli siis eroaa vaikkapa Macien konekielestä. Konekieliset käskyt ovat vain nollista ja ykkösistä koostuvia (2^n, nykyään jo 64 bittiä pitkiä) jonoja. Ensimmäinen askel käskyjen luettavammaksi tekemisessä oli nollien ja ykkösten korvaaminen helpommin muistettavilla nimillä. Assemblerin (konekielen tulkin) tehtävä oli sitten korvata nämä nimet nollilla ja ykkösillä. Myös assemblerin käskyt ovat siis koneesta eli prosessorin piirisarjasta riippuvaisia. Assemblerilla koodin kirjoittaminen hidasta (ja kallista), mutta hyvin ohjelmoidut ohjelmat ovat vastaavasti tehokkaita. Nykyäänkin monien ohjelmien keskeiset (usein kutsutut)

osat on kirjoitettu konekielellä maksimaalisen tehokkuuden varmistamiseksi. Konekieltä ja assembler-koodia kutsutaan matalan tason ohjelmointikieliksi.

Korkean tason ohjelmointikielissä konekieliset käskyt on korvattu ohjelmointikielen määritelmän mukaisilla käskyillä, jotka käännetään yhdeksi tai useammaksi konekielitason käskyksi. Kääntäjä (compiler) muuttaa ohjelmointikieliset ohjelmat konekielelle. Ohjelmointikielet eivät ole riippuvaisia käytetystä piirisarjasta, riittää kun kullekin keskusyksikölle on kirjoitettu oma kääntäjänsä. Korkean ja matalan tason ohjelmointikielen välinen ero on siis alustasidonnaisuudessa. Ensimmäiset kääntäjät on tietenkin kirjoitettu konekielellä, mutta nykyään kääntäjätkin tehdään korkean tason ohjelmointikielillä. Korkean tason ohjelmointikielten tehtävänä on irrottaa ohjelmointi mahdollisimman kauas käytetystä tekniikasta. Yleensä ohjelmointikielistä puhuttaessa tarkoitetaan vain korkean tason ohjelmointikieliä.

Osa ohjelmointikielistä käännetään ennen ohjelman suoritusta (esim. C), jolloin konekielinen ohjelma on etukäteen valmiina odottamassa ajamistaan. Osa käännetään ajon aikana, eli ohjelmakoodi tulkataan (tulkki, interpreter) ohjelman suorituksen yhteydessä (esim. PHP, Basic, Perl). Lisäksi on näiden välimuotoja: esimerkiksi C# ja Java-koodi käännetään etukäteen tavukoodiksi (byte code), joka ei ole konekieltä. Ohjelma suoritetaan tulkin tulkitessa tätä koodia käsky kerrallaan. Tavukoodin kääntäminen konekielelle sujuu lähdekoodia nopeammin ja se on siirrettävissä (ainakin samanversioisten) saman kielen eri tulkkien välillä alustalta toiselle. Niinpä se soveltuu esim. www-sivuille.

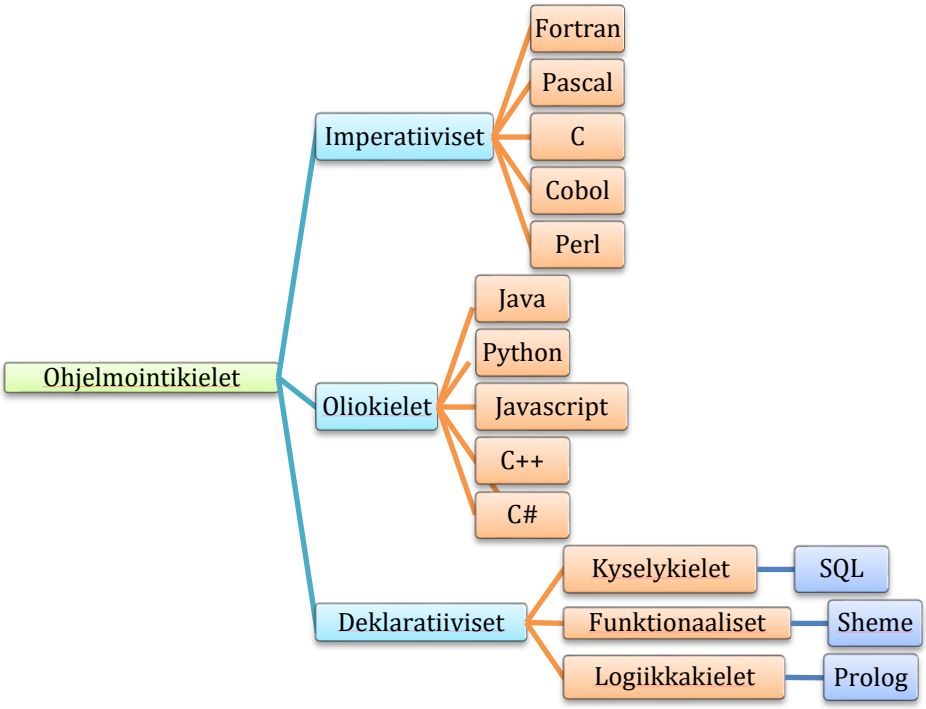
Erilaisia ohjelmointikieliä

Imperatiivisissa ohjelmointikielissä kuvataan täsmällisesti askel askeleelta toimenpiteet, joita noudattaen ongelma saadaan ratkaistua. Tämän kurssin alku, jossa on toistoja ja haarautumisia, on tyypillistä imperatiivista ohjelmointia.

Kurssin loppuosa edustaa taas **olio-ohjelmointia**. Siinä tieto ja tietoa käsittelevät metodit muodostavat yksikön, jota kutsutaan olioksi. Metodit ovat ainoa keino, jolla käyttäjä voi päästä käsiksi olion tietoihin.

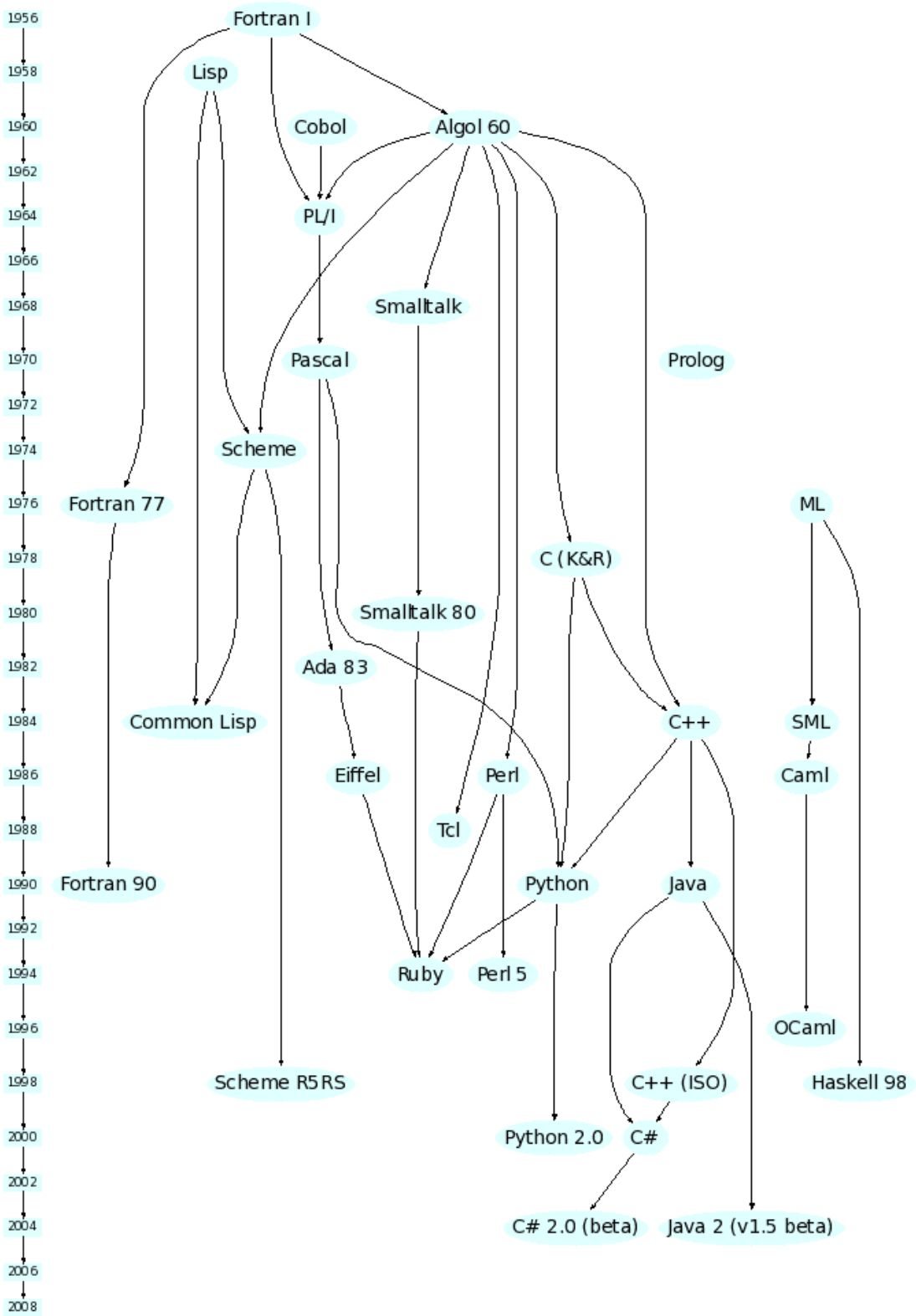
Kolmas kategoria ohjelmointikieliä ovat **deklaratiiviset kielet**. Niissä kuvataan vain ongelma joukkona ominaisuuksia, ei sen ratkaisua. Ohjelmalle esitetään joukko sääntöjä ja se kertoo, löytyikö kaikki säännöt toteuttavia kohteita. Tähän kategoriaan kuuluvat tietokantojen **kyselykielet** (SQL), **funktionaaliset kielet** ja **logiikkakielet**. Kyselykielet kertovat hakuehdot, joiden perusteella tietokannasta haetaan asioita. Funktionaaliset kielet taas esittävät kaiken funktioina.

Seuraavassa kuvassa on muutamia esimerkkejä kielistä.



Seuraavan sivun kuvassa on ohjelmointikielten ”sukupuu”.

Ohjelmointikieliä



C#

C# ("see sharp") on olio-ohjelmointikieli vuodelta 2002, jonka Microsoft kehitti käytettäväksi sen .Net -ympäristössä. Sen juuret ovat erityisesti Java- ja C++-kielissä mutta jokainen jotakin C:n sukuista ohjelmointikieltä (C, C++, JavaScript, php, ...) osaava tunnistaa rakenteet.

.Net on alusta, joka tarjoaa työkalut ja tekniikat hajautettujen ja verkkosovellusten tekemiseen. Sen kaksi pääosaa ovat Common Language Runtime (CLR) ja .Net Framework Class Library (FCL). CLR vastaa virtuaalikonetta, joka on käytössä esimerkiksi Javassa ja FCL on laaja kirjasto valmiita luokkia moniin eri tarkoituksiin. Ympäristö tarjoaa käännös- ja ajoaikaisen tuen kaikille tietyt vaatimukset täyttäville kielille. Ympäristössä voi siis käyttää muitakin kieliä mutta C# on kehitetty vain ja ainoastaan tähän tarkoitukseen.

C#:ssa on muutamia ominaisuuksia, jotka auttavat ohjelmoijaa luotettavien ohjelmien teossa. Käyttämättömät oliot, joihin ei enää ole yhteyttä, joutuvat automaattiseen roskien keruuseen. Poikkeusten käsittely auttaa virhetilanteiden havaitsemisessa ja niistä toipumisessa. Lisäksi kieli on suunniteltu tyyppiturvalliseksi, eli alustamattomia muuttujia ei voi käyttää, taulukon rajojen yli ei voi mennä ja laittomia tyypinmuunnoksia ei voi tehdä. Kieli on vahvasti tyypitetty. Tämä tarkoittaa sitä, että ennen muuttujan käyttöä sille on määrättävä tyyppi ja tuota tyyppiä ei voi sen jälkeen enää muuttaa.

C#:n kaikki tietotyypit pohjautuvat olio-tyyppiin object. Kaikki tietotyypit ovat siis luokkia, joilla on saman laiset perustoiminnot ja ominaisuudet.

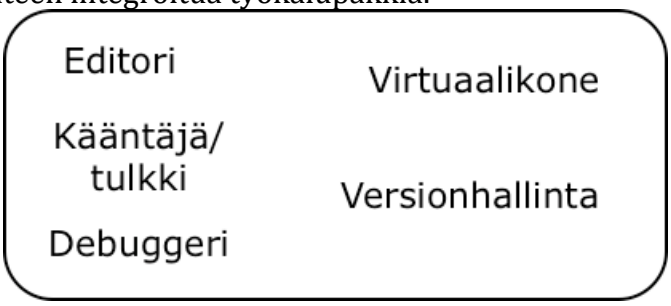
Ohjelmoijan työkalut: apuohjelmat

Ohjelmointityön apuna käytetään erilaisia apuohjelmia, siis valmiita, toisten ohjelmoijien jo tekemiä ohjelmia. Esimerkiksi:

- Ohjelmakoodi kirjoitetaan tiedostoihin jonkin ***editorin*** avulla.
- ***Kääntäjä/tulkki*** (engl. *compiler/interpreter*) on ohjelma, joka muokkaa ihmisen tuottaman ohjelmakoodin tietokoneen suoritettavaksi paremmin sopivaan muotoon.
- ***Debuggerin*** avulla voi tarkkailla ohjelman sisäistä toimintaa samalla kun ohjelmaa ajetaan. Tämä on hyödyllistä erityisesti virheitä (eli *bugeja*) etsittäessä sekä ohjelmointia opetellessa.
- Vähitellen kehittyvää ohjelmistoa voi ylläpitää paremmin käyttämällä sopivaa apuohjelmaa ***versionhallintaan*** (engl. *version control, revision control*).
- ***Virtuaalikonetta*** käytetään apuna joillakin kielillä kirjoitettujen ohjelmien ajamiseen.

Voidaan joko käyttää erillisiä apuohjelmia...

...tai yhteen integroitua työkalupakkia.



Ohjelmoijan työkalut: IDE

Sovelluskehitin (IDE, Itegrated Development Environment) kokoaa yhteen ohjelmien kehitystyössä käytettäviä työkaluja. Sovelluskehittimen avulla voi myös kätevästi jakaa tuotetun ohjelmakoodin haluamallaan tavalla erillisiin projekteihin. Tämä helpottaa lukuisten kooditiedostojen hallinnointia.
Muita nimiä sovelluskehittimelle: ***integroitu kehitysympäristö***,

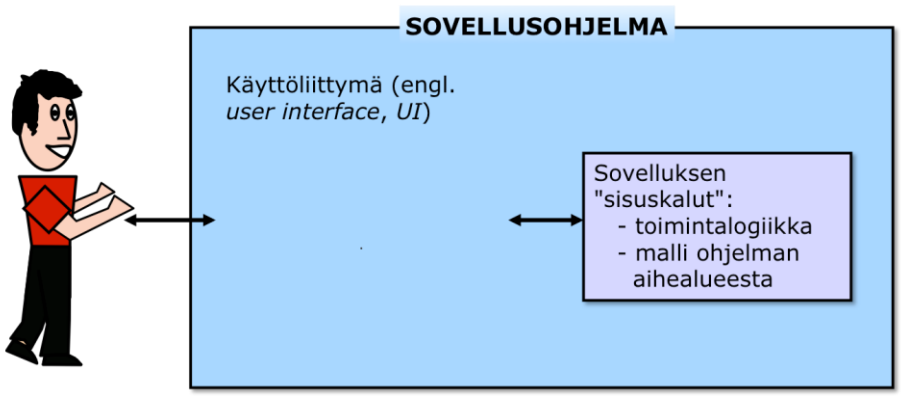
Microsoft Visual Studio-sovelluskehitin

Tällä kurssilla käytetään Visual Studio-nimistä sovelluskehitintä. Siitä on olemassa erilaisia versioita mutta meille riittää ilmainen Community –versio.

- *https://www.visualstudio.com/*

Visual Studio on monipuolinen ja monimutkainen kokonaisuus, kuten monet muutkin IDE:t... mutta sen kaikkia yksityiskohtia ei ole tarkoituskaan tällä kurssilla opetella. Alkuun pääsee hyvin perustoiminnot osaamalla.

Sovellusohjelma = UI + sisuskalut



Käyttäjä tietää (ehkä) miten käyttöliittymää käytetään. Ohjelmoijan on pystyttävä kuvaamaan koneelle miten käyttöliittymä toimii ja miten sisuskalut toimivat.