

Ohjelmistotekniikka

Matti Luukkainen, Jussi Mäki-Kahra, Pekka Mäki-Mikola ja 9
ohjaajaa

16.3.2021

- ▶ Kurssilla tutustutaan ohjelmistokehityksen periaatteisiin sekä menetelmiin ja sovelletaan niitä toteuttamalla pienehkö harjoitustyö

- ▶ Kurssilla tutustutaan ohjelmistokehityksen periaatteisiin sekä menetelmiin ja sovelletaan niitä toteuttamalla pienehkö harjoitustyö
- ▶ Kurssi osa *aineopintoja*
- ▶ Pakollisina *esitietoina*
 - ▶ Ohjelmoinnin jatkokurssi
 - ▶ Tietokantojen perusteet
- ▶ Hyödyllinen esitieto: Tietokone työvälineenä

- ▶ Kurssilla tutustutaan ohjelmistokehityksen periaatteisiin sekä menetelmiin ja sovelletaan niitä toteuttamalla pienehkö harjoitustyö
- ▶ Kurssi osa *aineopintoja*
- ▶ Pakollisina *esitietoina*
 - ▶ Ohjelmoinnin jatkokurssi
 - ▶ Tietokantojen perusteet
- ▶ Hyödyllinen esitieto: Tietokone työvälineenä
- ▶ Kurssimateriaali
 - ▶ <https://github.com/ohjelmistotekniikka-hy/kevat-2021>
 - ▶ <https://github.com/ohjelmistotekniikka-hy/python-kevat-2021>

- ▶ Kolmella ensimmäisellä viikolla ohjauksessa tai omatoimisesti tehtävät **laskarit**
 - ▶ palautetaan "internetiin"
- ▶ Viikolla 2 aloitetaan itsenäisesti tehtävä **harjoitustyö**
- ▶ Työtä edistetään pala palalta *viikoittaisten tavoitteiden* ohjaamana
- ▶ Kurssilla ei ole koetta

- ▶ Kolmella ensimmäisellä viikolla ohjauksessa tai omatoimisesti tehtävät **laskarit**
 - ▶ palautetaan "internetiin"
- ▶ Viikolla 2 aloitetaan itsenäisesti tehtävä **harjoitustyö**
- ▶ Työtä edistetään pala palalta *viikoittaisten tavoitteiden* ohjaamana
- ▶ Kurssilla ei ole koetta
- ▶ Harjoitustyö tulee tehdä kurssin aikataulujen puitteissa
- ▶ Kesken jäänyttä harjoitustyötä ei voi jatkaa seuraavalla kurssilla (syksyllä 2021)
- ▶ Muista siis varata riittävästi aikaa (10-15h viikossa) koko periodin ajaksi!

Luento, deadlinet ja ohjaus

- ▶ Kurssilla on vain yksi luento
- ▶ Laskareiden ja harjoitustyön välitavoitteiden viikoittaiset deadlinet *tiistaina klo 23:59*

Luento, deadlinet ja ohjaus

- ▶ Kurssilla on vain yksi luento
- ▶ Laskareiden ja harjoitustyön välitavoitteiden viikoittaiset deadlinet *tiistaina klo 23:59*
- ▶ Pajaa zoomissa, ks kurssisivu
- ▶ Myös kurssin Telegramissa voi kysellä apua ongelmatilanteissa

- ▶ Jaossa 60 pistettä jotka jakautuvat seuraavasti
 - ▶ Viikkodeadlinet 17p
 - ▶ osa viikkopisteistä tulee laskareista
 - ▶ Koodikatselmointi 2p
 - ▶ Dokumentaatio 12p
 - ▶ Testaus 5p
 - ▶ Lopullinen ohjelma 24p
 - ▶ laajuus, ominaisuudet ja koodin laatu

Arvosteluperusteet

- ▶ Jaossa 60 pistettä jotka jakautuvat seuraavasti
 - ▶ Viikkodeadlinet 17p
 - ▶ osa viikkopisteistä tulee laskareista
 - ▶ Koodikatselmointi 2p
 - ▶ Dokumentaatio 12p
 - ▶ Testaus 5p
 - ▶ Lopullinen ohjelma 24p
 - ▶ laajuus, ominaisuudet ja koodin laatu
- ▶ Arvosanaan 1 riittää 30 pistettä, arvosanaan 5 tarvitaan noin 55 pistettä.
- ▶ Läpipääsyyn vaatimuksena on lisäksi vähintään 10 pistettä lopullisesta ohjelmasta

TEORIA

- ▶ Kun ollaan tekemässä suurempaa ohjelmistoa jonkun muun kuin koodarin itsensä käyttöön, tarvitaan systemaattinen työskentelymenetelmä
 - ▶ muuten riskinä mm. että lopputulos ei vastaa käyttäjän tarvetta

Ohjelmistotuotanto

- ▶ Kun ollaan tekemässä suurempaa ohjelmistoa jonkun muun kuin koodarin itsensä käyttöön, tarvitaan systemaattinen työskentelymenetelmä
 - ▶ muuten riskinä mm. että lopputulos ei vastaa käyttäjän tarvetta
- ▶ Menetelmästä riippumatta ohjelmiston systemaattinen kehittäminen, eli *ohjelmistotuotanto* (engl. software engineering) sisältää useita erilaisia aktiviteetteja/vaiheita

Ohjelmistotuotanto

- ▶ Kun ollaan tekemässä suurempaa ohjelmistoa jonkun muun kuin koodarin itsensä käyttöön, tarvitaan systemaattinen työskentelymenetelmä
 - ▶ muuten riskinä mm. että lopputulos ei vastaa käyttäjän tarvetta
- ▶ Menetelmästä riippumatta ohjelmiston systemaattinen kehittäminen, eli *ohjelmistotuotanto* (engl. software engineering) sisältää useita erilaisia aktiviteetteja/vaiheita
 - ▶ *vaatimusmäärittelyä*
 - ▶ *suunnittelu*
 - ▶ *toteutus*
 - ▶ *testaus*
 - ▶ *ylläpito*

- ▶ Kartoitetaan ohjelman tulevien käyttäjien tai tilaajan kanssa, mitä toiminnallisuutta ohjelmaan halutaan

- ▶ Kartoitetaan ohjelman tulevien käyttäjien tai tilaajan kanssa, mitä toiminnallisuutta ohjelmaan halutaan
- ▶ Tämän lisäksi kartoitetaan ohjelman toimintaympäristön ja toteutusteknologian järjestelmälle asettamia rajoitteita

- ▶ Kartoitetaan ohjelman tulevien käyttäjien tai tilaajan kanssa, mitä toiminnallisuutta ohjelmaan halutaan
- ▶ Tämän lisäksi kartoitetaan ohjelman toimintaympäristön ja toteutusteknologian järjestelmälle asettamia rajoitteita
- ▶ Tuloksena jonkinlainen dokumentti, johon vaatimukset kirjataan
- ▶ Dokumentin muoto vaihtelee: paksu mapillinen papereita tai joukko postit-lappuja tai ...

- ▶ On olemassa lukuisia tapoja dokumentoida vaatimuksen

Vaatimusten kirjaaminen

- ▶ On olemassa lukuisia tapoja dokumentoida vaatimuksen
- ▶ Kurssin ennen vuotta 2018 pidetyissä versioissa käyttäjien vaatimukset dokumentointiin *käyttötapauksina* (engl. use case)
 - ▶ tapa on jo vanhahtava ja olemme jo hylänneet sen sen
- ▶ Kurssilla *Ohjelmistotuotanto* tutustumme nykyään yleisesti käytössä oleviin *käyttäjätarinoihin* (engl. user story)

Vaatimusten kirjaaminen

- ▶ On olemassa lukuisia tapoja dokumentoida vaatimuksen
- ▶ Kurssin ennen vuotta 2018 pidetyissä versioissa käyttäjien vaatimukset dokumentointiin *käyttötapauksina* (engl. use case)
 - ▶ tapa on jo vanhahtava ja olemme jo hylänneet sen sen
- ▶ Kurssilla *Ohjelmistotuotanto* tutustumme nykyään yleisesti käytössä oleviin *käyttäjätarinoihin* (engl. user story)
- ▶ Käytämme tällä kurssilla hieman kevyempää tapaa
- ▶ Kirjaamme järjestelmältä toivotun toiminnallisuuden vapaamuotoisena ranskalaisista viivoista koostuvana feature-listana

Kurssin referenssisovellus: TodoApp

<https://github.com/mluukkai/OtmTodoApp> havainnollistaa monia kurssin asioita ja toimii myös mallina omalle harjoitustyölle

- ▶ *todoapp* eli sovellus, jonka avulla käyttäjien on mahdollista pitää kirjaa omista tekemättömistä töistä, eli *todoista*

Katsotaan esimerkkinä Todo-sovelluksen vaatimusmäärittelyä

Kurssin referenssisovellus: TodoApp

<https://github.com/mluukkai/OtmTodoApp> havainnollistaa monia kurssin asioita ja toimii myös mallina omalle harjoitustyölle

- ▶ *todoapp* eli sovellus, jonka avulla käyttäjien on mahdollista pitää kirjaa omista tekemättömistä töistä, eli *todoista*

Katsotaan esimerkkinä Todo-sovelluksen vaatimusmäärittelyä

- ▶ Aloitetaan tunnistamalla järjestelmän *käyttäjäroolit*
- ▶ Todo-sovelluksella kaksi käyttäjääroolia
 - ▶ *normaalit käyttäjät*
 - ▶ laajemmilla oikeuksilla varustetut *ylläpitäjät*

Kurssin referenssisovellus: TodoApp

<https://github.com/mluukkai/OtmTodoApp> havainnollistaa monia kurssin asioita ja toimii myös mallina omalle harjoitustyölle

- ▶ *todoapp* eli sovellus, jonka avulla käyttäjien on mahdollista pitää kirjaa omista tekemättömistä töistä, eli *todoista*

Katsotaan esimerkkinä Todo-sovelluksen vaatimusmäärittelyä

- ▶ Aloitetaan tunnistamalla järjestelmän *käyttäjäroolit*
- ▶ Todo-sovelluksella kaksi käyttäjääroolia
 - ▶ *normaalit käyttäjät*
 - ▶ laajemmilla oikeuksilla varustetut *ylläpitäjät*
- ▶ Mietitään mitä toiminnallisuuksia kunkin käyttäjäärooli tarvitsee

ToDoApp:in vaatimusmäärittely

- ▶ Todo-sovelluksen *normaalien käyttäjien* toiminnallisuuksia ovat esim. seuraavat
 - ▶ käyttäjä voi luoda järjestelmään käyttäjätunnuksen
 - ▶ käyttäjä voi kirjautua järjestelmään
 - ▶ kirjautumisen jälkeen käyttäjä näkee omat tekemättömät työt eli *todot*
 - ▶ kirjaantunut käyttäjä voi luoda uuden tehtävän eli *todon*
 - ▶ kirjaantunut käyttäjä voi merkitä *todon* tehdyksi, jolloin se häviää listalta

ToDoApp:in vaatimusmäärittely

- ▶ Todo-sovelluksen *normaalien käyttäjien* toiminnallisuuksia ovat esim. seuraavat
 - ▶ käyttäjä voi luoda järjestelmään käyttäjätunnuksen
 - ▶ käyttäjä voi kirjautua järjestelmään
 - ▶ kirjautumisen jälkeen käyttäjä näkee omat tekemättömät työt eli *todot*
 - ▶ kirjaantunut käyttäjä voi luoda uuden tehtävän eli *todon*
 - ▶ kirjaantunut käyttäjä voi merkitä *todon* tehdyksi, jolloin se häviää listalta
- ▶ *Ylläpitäjän* toiminnallisuuksia esim. seuraavat
 - ▶ ylläpitäjä näkee tilastoja sovelluksen käytöstä
 - ▶ ylläpitäjä voi poistaa normaalin käyttäjätunnuksen

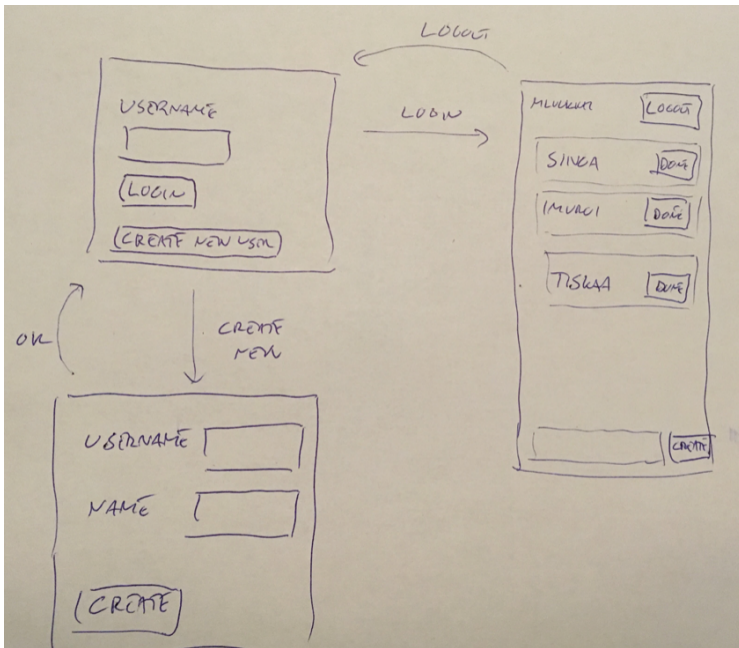
Vaatimusmäärittely: toimintaympäristön rajoitteet, käyttöliittymä

- ▶ Ohjelmiston vaatimukseen kuuluvat myös *toimintaympäristön rajoitteet*
- ▶ Todo-sovellusta koskevat seuraavat rajoitteet:
 - ▶ ohjelmiston tulee toimia Linux- ja OSX-käyttöjärjestelmillä varustetuissa koneissa
 - ▶ toteutetaan Java FX -kirjaston avulla
 - ▶ käyttäjien ja todojen tiedot talletetaan paikallisen koneen levyille

Vaatimusmäärittely: toimintaympäristön rajoitteet, käyttöliittymä

- ▶ Ohjelmiston vaatimuksiin kuuluvat myös *toimintaympäristön rajoitteet*
- ▶ Todo-sovellusta koskevat seuraavat rajoitteet:
 - ▶ ohjelmiston tulee toimia Linux- ja OSX-käyttöjärjestelmillä varustetuissa koneissa
 - ▶ toteutetaan Java FX -kirjaston avulla
 - ▶ käyttäjien ja todojen tiedot talletetaan paikallisen koneen levyille
- ▶ Vaatimusmäärittelyn aikana hahmotellaan yleensä myös sovelluksen käyttöliittymä

Todo-sovelluksen käyttöliittymäluonnos



- ▶ Suunnittelu jakautuu kahteen erilliseen vaiheeseen

- ▶ Suunnittelu jakautuu kahteen erilliseen vaiheeseen
- ▶ *Arkkitehtuurisuunnittelussa* määritellään ohjelman rakenne karkealla tasolla
 - ▶ mistä suuremmista rakennekomponenteista ohjelma koostuu
 - ▶ miten komponentit yhdistetään, eli minkälaisia komponenttien väliset rajapinnat ovat
 - ▶ mitä riippuvuuksia ohjelmalla on esim. ohjelmakirjastoihin, tietokantoihin ja ulkoisiin rajapintoihin

- ▶ Suunnittelu jakautuu kahteen erilliseen vaiheeseen
- ▶ *Arkkitehtuurisuunnittelussa* määritellään ohjelman rakenne karkealla tasolla
 - ▶ mistä suuremmista rakennekomponenteista ohjelma koostuu
 - ▶ miten komponentit yhdistetään, eli minkälaisia komponenttien väliset rajapinnat ovat
 - ▶ mitä riippuvuuksia ohjelmalla on esim. ohjelmakirjastoihin, tietokantoihin ja ulkoisiin rajapintoihin
- ▶ Arkkitehtuurisuunnittelua tarkoittaa *oliosuunnittelu*
 - ▶ minkälaisista luokista komponentit koostuvat
 - ▶ miten luokat kutsuvat toistensa metodeja sekä mitä apukirjastoja ne käyttävät
- ▶ Myös ohjelmiston suunnittelu, erityisesti sen arkkitehtuuri dokumentoidaan

Testaus

- ▶ Toteutuksen yhteydessä ja sen jälkeen järjestelmää testataan
- ▶ Testausta on monentasoista

Testaus

- ▶ Toteutuksen yhteydessä ja sen jälkeen järjestelmää testataan
- ▶ Testausta on monentasoista
- ▶ *Yksikkötestauksessa* tutkitaan yksittäisten metodien ja luokkien toimintaa.
 - ▶ ohjelmoijan vastuulla

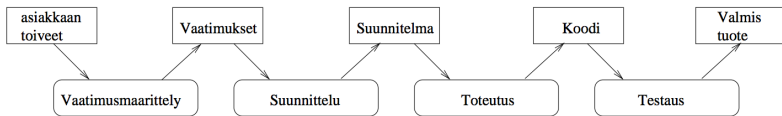
- ▶ Toteutuksen yhteydessä ja sen jälkeen järjestelmää testataan
- ▶ Testausta on monentasoista
- ▶ *Yksikkötestauksessa* tutkitaan yksittäisten metodien ja luokkien toimintaa.
 - ▶ ohjelmoijan vastuulla
- ▶ Kun erikseen ohjelmoidut luokat yhdistetään, suoritetaan *integraatiotestaus*
 - ▶ varmistetaan erillisten osien yhteentoimivuus
 - ▶ ohjelmoijan vastuulla

- ▶ Toteutuksen yhteydessä ja sen jälkeen järjestelmää testataan
- ▶ Testausta on monentasoista
- ▶ *Yksikkötestauksessa* tutkitaan yksittäisten metodien ja luokkien toimintaa.
 - ▶ ohjelmoijan vastuulla
- ▶ Kun erikseen ohjelmoidut luokat yhdistetään, suoritetaan *integraatiotestaus*
 - ▶ varmistetaan erillisten osien yhteentoimivuus
 - ▶ ohjelmoijan vastuulla
- ▶ *Järjestelmätestauksessa* testataan ohjelmistoa kokonaisuutena: toimiiko se vaatimusedokumentin mukaisesti
 - ▶ suoritetaan ohjelman todellisen käyttöliittymän kautta
 - ▶ saattaa tapahtua erillisen laadunhallintatiimin toimesta

- ▶ Ohjelmistoja on 70-luvulta asti tehty vaihe vaiheelta etenevän *vesiputousmallin* (engl. waterfall model) mukaan

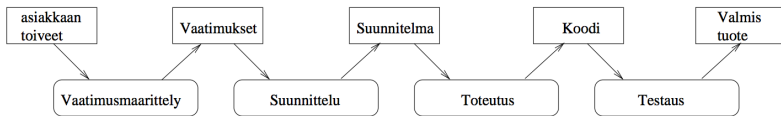
Vesiputousmalli

- ▶ Ohjelmistoja on 70-luvulta asti tehty vaihe vaiheelta etenevän *vesiputousmallin* (engl. waterfall model) mukaan
- ▶ Vesiputousmallissa edellä esitellyt ohjelmistotuotannon vaiheet suoritetaan peräkkäin



Vesiputousmalli

- ▶ Ohjelmistojä on 70-luvulta asti tehty vaihe vaiheelta etenevän *vesiputousmallin* (engl. waterfall model) mukaan
- ▶ Vesiputousmallissa edellä esitellyt ohjelmistotuotannon vaiheet suoritetaan peräkkäin



- ▶ Eri vaiheet ovat yleensä erillisten tiimien tekemiä
- ▶ Edellyttää perusteellista ja raskasta dokumentaatiota

Vesiputousmallin ongelmat

- ▶ Mallin toimivuus perustuu siihen oletukseen, että vaatimukset pystytään määrittelemään täydellisesti etukäteen

Vesiputousmallin ongelmat

- ▶ Mallin toimivuus perustuu siihen oletukseen, että vaatimukset pystytään määrittelemään täydellisesti etukäteen
- ▶ Näin ei useinkaan ole
 - ▶ Asiakkaat eivät osaa ilmaista kaikki ohjelmistolle asettamansa vaatimukset
 - ▶ Vasta käyttäessään valmista ohjelmistoa asiakkaat ymmärtää, mitä he haluavat
 - ▶ Vaikka vaatimukset kunnossa laatimishetkellä, toimintaympäristö voi muuttua ja valmistuessaan ohjelmisto on vanhentunut

Vesiputousmallin ongelmat

- ▶ Mallin toimivuus perustuu siihen oletukseen, että vaatimukset pystytään määrittelemään täydellisesti etukäteen
- ▶ Näin ei useinkaan ole
 - ▶ Asiakkaat eivät osaa ilmaista kaikki ohjelmistolle asettamansa vaatimukset
 - ▶ Vasta käyttäessään valmista ohjelmistoa asiakkaat ymmärtää, mitä he haluavat
 - ▶ Vaikka vaatimukset kunnossa laatimishetkellä, toimintaympäristö voi muuttua ja valmistuessaan ohjelmisto on vanhentunut
- ▶ Toinen suuri ongelma on myöhään aloitettava testaus
 - ▶ Erityisesti integraatiotestauksessa löytyy usein pahoja ongelmia, joiden korjaaminen on hidasta ja kallista

Ketterä ohjelmistokehitys

- ▶ Vesiputousmallin heikkoudet ovat johtaneet 2000-luvun alun jälkeen *ketterien (engl. agile) menetelmien* käyttöönottoon
- ▶ Alussa kartoitetaan pääpiirteissään ohjelmiston vaatimuksia ja hahmotellaan ohjelmiston alustava arkkitehtuuri

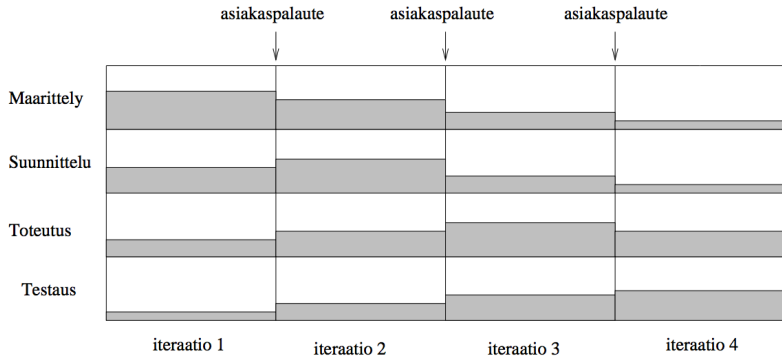
Ketterä ohjelmistokehitys

- ▶ Vesiputousmallin heikkoudet ovat johtaneet 2000-luvun alun jälkeen *ketterien (engl. agile) menetelmien* käyttöönottoon
- ▶ Alussa kartoitetaan pääpiirteissään ohjelmiston vaatimuksia ja hahmotellaan ohjelmiston alustava arkkitehtuuri
- ▶ Tämän jälkeen suoritetaan useita *iteraatioita*, joiden aikana ohjelmistoa rakennetaan pala palalta eteenpäin
- ▶ Kussakin iteraatiossa suunnitellaan ja toteutetaan valmiiksi pieni osa ohjelmiston vaatimuksista

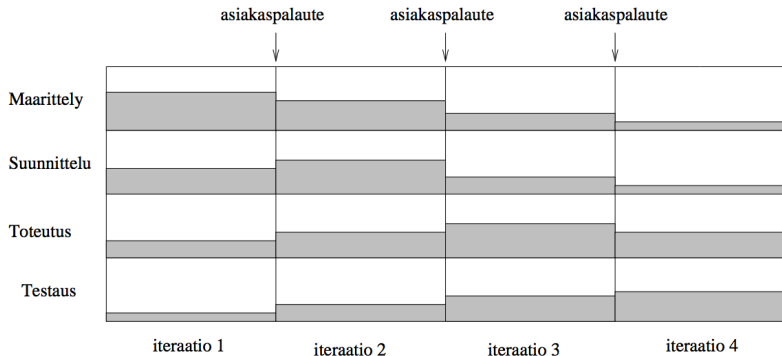
Ketterä ohjelmistokehitys

- ▶ Vesiputousmallin heikkoudet ovat johtaneet 2000-luvun alun jälkeen *ketterien* (*engl. agile*) *menetelmien* käyttöönottoon
- ▶ Alussa kartoitetaan pääpiirteissään ohjelmiston vaatimuksia ja hahmotellaan ohjelmiston alustava arkkitehtuuri
- ▶ Tämän jälkeen suoritetaan useita *iteraatioita*, joiden aikana ohjelmistoa rakennetaan pala palalta eteenpäin
- ▶ Kussakin iteraatiossa suunnitellaan ja toteutetaan valmiiksi pieni osa ohjelmiston vaatimuksista
- ▶ Asiakas pääsee kokeilemaan ohjelmistoa jokaisen iteraation jälkeen
- ▶ Voidaan jo aikaisessa vaiheessa todeta, onko kehitystyö etenemässä oikeaan suuntaan
- ▶ Vaatimuksia voidaan tarvittaessa tarkentaa ja muuttaa

Ketterä ohjelmistokehitys



Ketterä ohjelmistokehitys



Teemme kurssin harjoitustyötä ketterässä hengessä viikon mittaisilla iteraatioilla

TYÖKALUJA

Työkaluja

- ▶ Tarvitsemme ohjelmistokehityksessä suuren joukon käytännön työkaluja.
- ▶ Komentorivi ja versionhallinta
 - ▶ olet jo ehkä käyttänyt muilla kursseilla komentoriviä ja git-versionhallintaa
 - ▶ molemmat ovat tärkeässä roolissa ohjelmistokehityksessä
 - ▶ harjoitellaan viikon 1 laskareissa

- ▶ Tarvitsemme ohjelmistokehityksessä suuren joukon käytännön työkaluja.
- ▶ Komentorivi ja versionhallinta
 - ▶ olet jo ehkä käyttänyt muilla kursseilla komentoriviä ja git-versionhallintaa
 - ▶ molemmat ovat tärkeässä roolissa ohjelmistokehityksessä
 - ▶ harjoitellaan viikon 1 laskareissa
- ▶ Maven
 - ▶ Olet todennäköisesti ohjelmoinut Javaa NetBeansilla ja tottunut painamaan “vihreää nappia” tai “mustaa silmää”
 - ▶ tutkimme kurssilla hieman miten Javalla tehdyn ohjelmiston *hallinnointi* tapahtuu NetBeansin “ulkopuolella”
 - ▶ koodin kääntäminen, koodin sekä testin suorittaminen ja koodin paketoiminen NetBeansin ulkopuolella suoritettavissa olevaksi jar-paketiksi
 - ▶ Java-projektien hallinnointiin on olemassa muutamia vaihtoehtoja, käytämme joillekin TiKaPesta tuttua *mavenia*

- ▶ Ohjelmistojen testaus tapahtuu nykyään automatisoitujen testityökalujen toimesta

- ▶ Ohjelmistojen testaus tapahtuu nykyään automatisoitujen testityökalujen toimesta
- ▶ Java-maailmassa testausta dominoi lähes yksinvaltiaan tavoin JUnit
- ▶ Tulet kurssin ja myöhempienkin opintojesi aikana kirjoittamaan paljon JUnit-testejä
- ▶ Viikon 2 laskareissa harjoitellaan JUnitin perusteita

- ▶ Automaattisten testien lisäksi koodille voidaan määritellä erilaisia automaattisesti tarkastettavia tyylillisiä sääntöjä
 - ▶ ylläpidetään koodin luettavuutta ja varmistetaan, että koodi noudateta samoja tyylillisiä konventioita




Checkstyle

- ▶ Automaattisten testien lisäksi koodille voidaan määritellä erilaisia automaattisesti tarkastettavia tyylillisiä sääntöjä
 - ▶ ylläpidetään koodin luettavuutta ja varmistetaan, että koodi noudateta samoja tyylillisiä konventioita
- ▶ Käytämme kurssilla tarkoitukseen *Checkstyle*-nimistä työkalua

Checkstyle

- ▶ Automaattisten testien lisäksi koodille voidaan määritellä erilaisia automaattisesti tarkastettavia tyylillisiä sääntöjä
 - ▶ ylläpidetään koodin luettavuutta ja varmistetaan, että koodi noudateta samoja tyylillisiä konventioita
- ▶ Käytämme kurssilla tarkoitukseen *Checkstyle*-nimistä työkalua
- ▶ Ohjelmoinnin perusteet ja jatkokurssi käyttivät Checkstyleä valvomaan ohjelman sisennystä
- ▶ Kurssilla kontrolloimme mm. muuttujien nimentää, sulkumerkkien sijoittelua ja välilyönnin käytön systemaattisuutta

- ▶ Osa ohjelmiston dokumentointia on lähdekoodin luokkien julkisten metodien kuvaus
- ▶ Javassa lähdekoodi dokumentoidaan käyttäen *JavaDoc*-työkalua
- ▶ Dokumentointi tapahtuu kirjoittamalla koodin yhteyteen sopivasti muotoiltuja kommentteja

```
5  
6
7
8
9  
10
11
12
13  
14
```

```
/**
 * Uuden todon lisääminen kirjautuneena olevalle käyttäjälle
 *
 * @param   content   luotavan todon sisältö
 *
 * @return  true jos todon luominen onnistuu, muuten false
 */

public boolean createTodo(String content) {
    Todo todo = new Todo(content, loggedIn);
}
```

Sovelluksen JavaDocia voi tarkastella selaimen avulla

All Classes

Packages

todoapp.dao
todoapp.domain
todoapp.ui

All Classes

FileTodoDao
FileUserDao
Main
Todo
TodoDao
TodoService
User
UserDao

OVERVIEW
PACKAGE
CLASS
USE
TREE
DEPRECATED
INDEX
HELP

PREV PACKAGE
NEXT PACKAGE
FRAMES
NO FRAMES

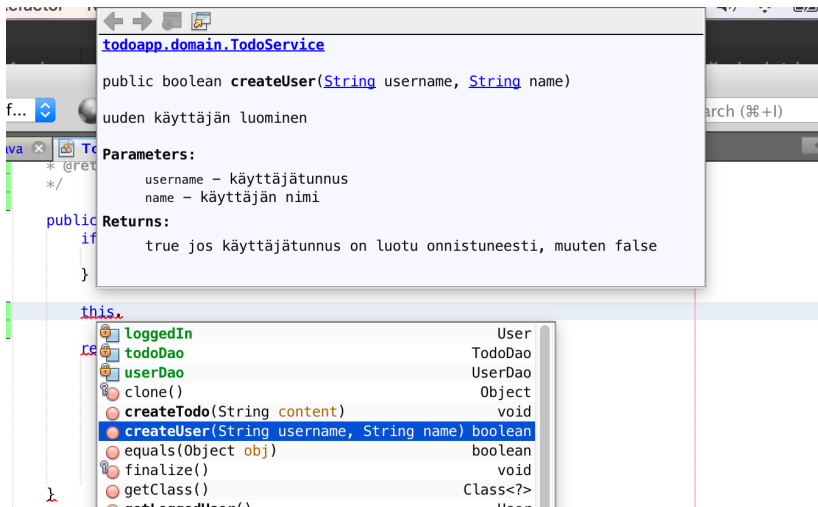
Package todoapp.domain

Class Summary

Class	Description
Todo	Yksittäistä työtä kuvaava luokka
TodoService	Sovelluslogiikasta vastaava luokka
User	Järjestelmän käyttäjää edustava luokka

OVERVIEW
PACKAGE
CLASS
USE
TREE
DEPRECATED
INDEX
HELP

NetBeans osaa näyttää ohjelmoimissa koodiin määritellyn JavaDocin



UML

- ▶ Dokumentoinnissa ja suunnittelun tukena tarvitaan ohjelman rakennetta ja toimintaa havainnollistavia kaavioita

UML ja dokumentointi

- ▶ Dokumentoinnissa ja suunnittelun tukena tarvitaan ohjelman rakennetta ja toimintaa havainnollistavia kaavioita
- ▶ *UML* eli *Unified Modeling Language* on 1997 standardoitu olio-ohjelmistojen mallintamiseen tarkoitettu mallinnuskieli
- ▶ UML sisältää 13 erilaista kaaviotyyppiä
- ▶ UML oli aikoinaan todella suosittu, nyt sen suosio on hiipumaan päin, muutama tärkein kaaviotyyppi kannattaa kuitenkin osata

UML ja dokumentointi

- ▶ Dokumentoinnissa ja suunnittelun tukena tarvitaan ohjelman rakennetta ja toimintaa havainnollistavia kaavioita
- ▶ *UML* eli *Unified Modeling Language* on 1997 standardoitu olio-ohjelmistojen mallintamiseen tarkoitettu mallinnuskieli
- ▶ UML sisältää 13 erilaista kaaviotyyppiä
- ▶ UML oli aikoinaan todella suosittu, nyt sen suosio on hiipumaan päin, muutama tärkein kaaviotyyppi kannattaa kuitenkin osata
- ▶ Käytämme kurssilla luokka-, pakkaus- ja sekvenssikaavioita

Luokkakaaviot

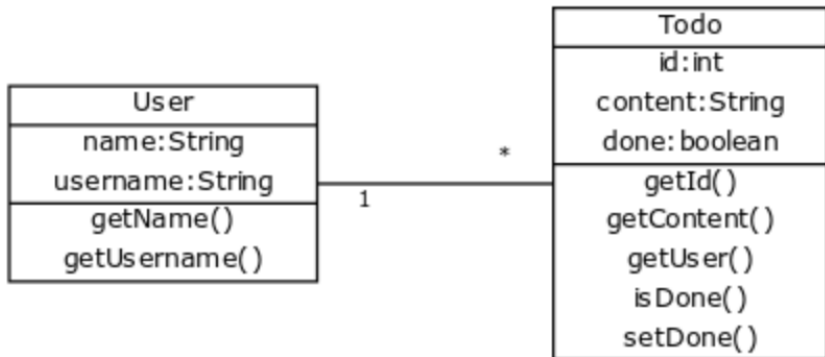
- ▶ Kurssin Tietokantojen perusteet aiemmista versioista tuttujen *luokkakaavioiden* käyttötarkoitus on luokkien ja niiden välisten suhteiden kuvailu
- ▶ Todo-sovelluksen oleellista tietosisältöä kuvaavat luokat

```
public class User {  
    private String name;  
    private String username;  
    // ...  
}
```

```
public class Todo {  
    private int id;  
    private String content;  
    private boolean done;  
    private User user;  
    // ...  
}
```

Todo-sovelluksen tietosisällön luokkakaavio

- ▶ Yhdellä käyttäjällä voi olla montaa Todoa
- ▶ Todo liittyy aina yhteen käyttäjään



Todo-sovelluksen tietosisällön luokkakaavio

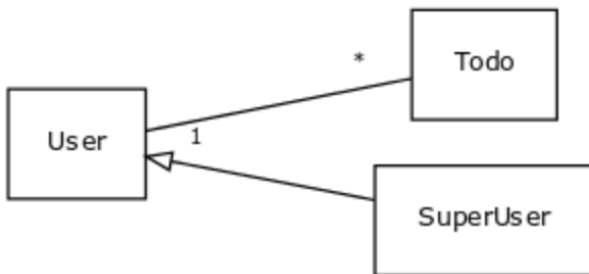
- Yleensä ei ole mielekästä kuvata luokkia tällä tarkkuudella, eli **luokkakaavioihin riittää merkitä luokan nimi**



- Kaaviota parempi paikka metodien kuvaamiselle on koodiin liittyvä JavaDoc-dokumentaatio

Rajapinnan toteutus ja perintä luokkakaaviossa

- ▶ Jos Todo-sovelluksessa olisi normaalin käyttäjän eli luokan *User* perivä ylläpitäjää kuvaava luokka *SuperUser*, merkattaisiin se luokkakaavioon seuraavasti

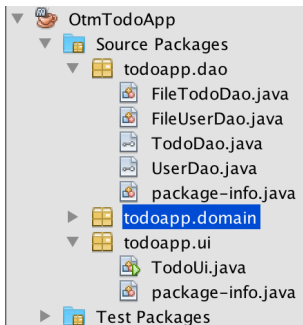


- ▶ Rajapinnan toteutus merkitään samalla tavalla eli valkoisella nuolenpäällä

- ▶ Ohjelmiston korkeamman tason rakenne näkyy yleensä siinä miten koodi on jaettu *pakkauksiin*

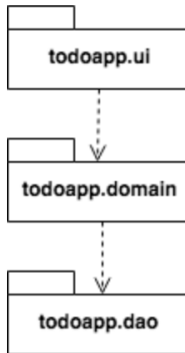
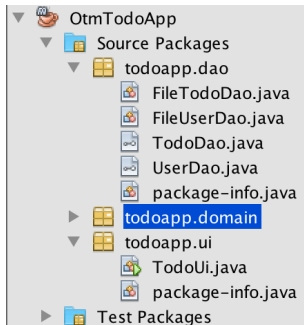
Pakkauskaavio

- ▶ Ohjelmiston korkeamman tason rakenne näkyy yleensä siinä miten koodi on jaettu *pakkauksiin*
- ▶ Todo-sovelluksen koodi on sijoitettu *pakkauksiin* seuraavasti:



Pakkauskaavio

- Pakkausrakenne voidaan kuvata UML:ssä pakkauskaaviolla



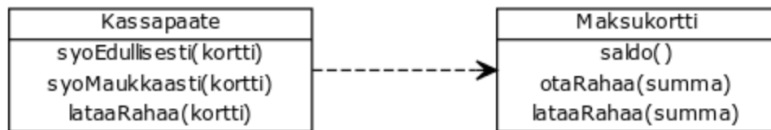
- Pakkausten välille on merkitty riippuvuus jos pakkauksen luokat käyttävät toisen pakkauksen luokkia

Toiminnallisuuden kuvaaminen

- ▶ Luokka- ja pakkauskaaviot kuvaavat ohjelman rakennetta
- ▶ Ohjelman toiminta ei kuitenkaan tule niistä ilmi millään tavalla

Toiminnallisuuden kuvaaminen

- ▶ Luokka- ja pakkauskaaviot kuvaavat ohjelman rakennetta
- ▶ Ohjelman toiminta ei kuitenkaan tule niistä ilmi millään tavalla
- ▶ Esim. Ohpen Unicafe-tehtävä



- ▶ Vaikka kaavioon on nyt merkitty metodien nimet, ei ohjelman toimintalogiikka selviä kaaviosta
- ▶ Esim. mitä tapahtuu, kun maksukortilla jolla on rahaa 3 euroa, ostetaan edullinen lounas?

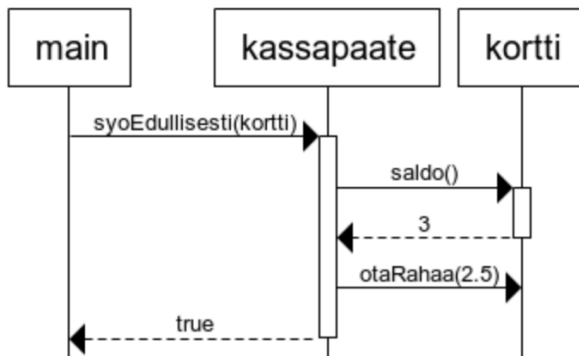
Sekvenssikaavio

- ▶ Kehitetty alunperin kuvaamaan verkossa olevien ohjelmien keskinäisen kommunikoinnin etenemistä
- ▶ Sopivat jossain määrin kuvaamaan, miten ohjelman oliot kutsuvat toistensa metodeja suorituksen aikana

Mitä tapahtuu, kun maksukortilla jolla on rahaa 3 euroa, ostataan edullinen lounas?

```
public boolean syoEdullisesti(Maksukortti kortti) {  
    if (kortti.saldo() < EDULLISEN_HINTA) {  
        return false;  
    }  
  
    kortti.otaRahaa(EDULLISEN_HINTA);  
    this.edulliset++;  
    return true;  
}
```

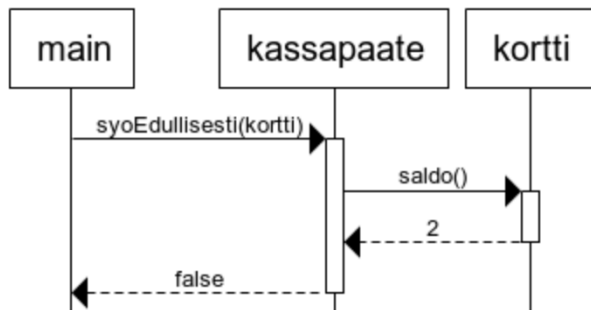
Onnistunut ostos sekvenssikaaviona



- ▶ Oliot ovat laatikoita joista lähtee alas “elämänlanka”
- ▶ Aika etenee ylhäältä alas
- ▶ Metodikutsut ovat nuolia, jotka yhdistävää kutsuvan ja kutsutun olion elämänlangat
- ▶ Paluuarvo merkitään katkoviivalla

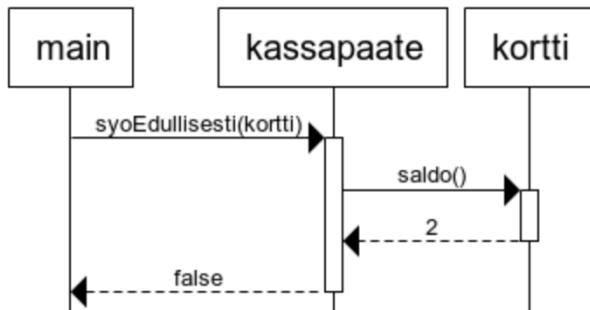
Epäonnistunut ostos sekvenssikaaviona

Mitä tapahtuu, jos maksukortin saldo on 2 euroa, eli vähemmän kuin lounaan hinta:



Epäonnistunut ostos sekvenssikaaviona

Mitä tapahtuu, jos maksukortin saldo on 2 euroa, eli vähemmän kuin lounaan hinta:



- ▶ Sekvenssikaaviot kuvaavat siis yksittäistä tapahtumasarjaa
- ▶ Toiminnallisuuden kuvaamiseen tarvitaankin yleensä useampi sekvenssikaavio

HARJOITUSTYÖ

- ▶ Kurssin pääpainon muodostaa viikolla 2 aloitettava harjoitustyö
- ▶ Harjoitustyössä toteutetaan itsenäisesti ohjelmisto omavalintaisesta aiheesta
- ▶ Harjoitustyötä tehdään itsenäisesti, mutta tarjolla on pajaohjausta

Älä plagioi

- ▶ Kurssilla seurataan Helsingin yliopiston opintokäytäntöjä
- ▶ Plagiarismi ja opintovilppi, eli esimerkiksi netissä olevien tai kaverilta saatujen vastausten kopiointi ja niiden palauttaminen omana työnä on kiellettyä
- ▶ Todettu opintovilppi johtaa kurssisuorituksen hylkäämiseen ja toistuva opintovilppi voi johtaa opinto-oikeuden määräaikaiseen menettämiseen

Työn eteneminen

- ▶ Edetään viikottaisten tavoitteiden mukaan
- ▶ Työ on saatava valmiiksi kurssin aikana ja sitä on toteutettava tasaisesti, muuten kurssi katsotaan keskeytetyksi

Työn eteneminen

- ▶ Edetään viikottaisten tavoitteiden mukaan
- ▶ Työ on saatava valmiiksi kurssin aikana ja sitä on toteutettava tasaisesti, muuten kurssi katsotaan keskeytetyksi
- ▶ Samaa ohjelmaa ei voi jatkaa seuraavalla kurssilla (eli syksyllä 2021), vaan työ on aloitettava uudella aiheella alusta

Työn eteneminen

- ▶ Edetään viikottaisten tavoitteiden mukaan
- ▶ Työ on saatava valmiiksi kurssin aikana ja sitä on toteutettava tasaisesti, muuten kurssi katsotaan keskeytetyksi
- ▶ Samaa ohjelmaa ei voi jatkaa seuraavalla kurssilla (eli syksyllä 2021), vaan työ on aloitettava uudella aiheella alusta
- ▶ Koko kurssin arvostelu perustuu pääasiassa harjoitustyöstä saataviin pisteisiin
- ▶ Osa pisteistä kertyy viikoittaisten välitavoitteiden kautta, osa taas perustuu työn lopulliseen palautukseen

- ▶ Harjoitustyön ohjelmointikieli on Java tai Python
- ▶ Ohjelmakoodin muuttujat, luokat ja metodit **kirjoitetaan englanniksi**
- ▶ Dokumentaatio voidaan kirjoittaa joko suomeksi tai englanniksi

- ▶ Harjoitustyön ohjelmointikieli on Java tai Python
- ▶ Ohjelmakoodin muuttujat, luokat ja metodit **kirjoitetaan englanniksi**
- ▶ Dokumentaatio voidaan kirjoittaa joko suomeksi tai englanniksi
- ▶ Web-sovelluksia kurssilla ei sallita
 - ▶ Sovelluksessa voi toki olla webissä toimivia komponentteja, mutta sovelluksen käyttöliittymän tulee olla ns. desktop-sovellus

Ohjelman toteutus

- ▶ Toteutus etenee “iteratiivisesti ja inkrementaalisesti”
 - ▶ Heti ensimmäisellä viikolla toteutetaan pieni käyttökelpoinen osa toiminnallisuudesta
 - ▶ ohjelman ydin pidetään koko ajan toimivana, uutta toiminnallisuutta lisäten, kunnes tavoiteltu laajuus on saavutettu

Ohjelman toteutus

- ▶ Toteutus etenee “iteratiivisesti ja inkrementaalisesti”
 - ▶ Heti ensimmäisellä viikolla toteutetaan pieni käyttökelpoinen osa toiminnallisuudesta
 - ▶ ohjelman ydin pidetään koko ajan toimivana, uutta toiminnallisuutta lisäten, kunnes tavoiteltu laajuus on saavutettu
- ▶ Iteratiiviseen tapaan tehdä ohjelma liittyy kiinteästi automatisoitu testaus
- ▶ Uutta toiminnallisuutta lisättäessä ja vanhaa muokatessa täytyy varmistua, että kaikki vanhat ominaisuudet toimivat edelleen
- ▶ Jotta ohjelmaa pystyisi testaamaan, on tärkeää että sovelluslogiikkaa ei kirjoiteta käyttöliittymän sekaan

Ohjelman toteutus

- ▶ Toteutus etenee “iteratiivisesti ja inkrementaalisesti”
 - ▶ Heti ensimmäisellä viikolla toteutetaan pieni käyttökelpoinen osa toiminnallisuudesta
 - ▶ ohjelman ydin pidetään koko ajan toimivana, uutta toiminnallisuutta lisäten, kunnes tavoiteltu laajuus on saavutettu
- ▶ Iteratiiviseen tapaan tehdä ohjelma liittyy kiinteästi automatisoitu testaus
- ▶ Uutta toiminnallisuutta lisättäessä ja vanhaa muokatessa täytyy varmistua, että kaikki vanhat ominaisuudet toimivat edelleen
- ▶ Jotta ohjelmaa pystyisi testaamaan, on tärkeää että sovelluslogiikkaa ei kirjoiteta käyttöliittymän sekaan
- ▶ Graafiseen käyttöliittymään suositellaan JavaFX:ää, Pythonilla tkinteriä tai Pygamea
- ▶ Tiedon talletus joko tiedostoon tai tietokantaan suositeltavaa

Ohjelman toteutus

- ▶ Tavoitteena on tuottaa ohjelma, joka voitaisiin antaa toiselle opiskelijalle ylläpidettäväksi ja täydennettäväksi
 - ▶ koodin on siis oltava ymmärrettävää ja jatkokehityksen mahdollistavaa

Ohjelman toteutus

- ▶ Tavoitteena on tuottaa ohjelma, joka voitaisiin antaa toiselle opiskelijalle ylläpidettäväksi ja täydennettäväksi
 - ▶ koodin on siis oltava ymmärrettävää ja jatkokehityksen mahdollistavaa
- ▶ Lopullisessa palautuksessa on oltava lähdekoodin lisäksi dokumentaatio ja automaattiset testit sekä Java-sovelluksissa jar-tiedosto

Ohjelman toteutus

- ▶ Tavoitteena on tuottaa ohjelma, joka voitaisiin antaa toiselle opiskelijalle ylläpidettäväksi ja täydennettäväksi
 - ▶ koodin on siis oltava ymmärrettävää ja jatkokehityksen mahdollistavaa
- ▶ Lopullisessa palautuksessa on oltava lähdekoodin lisäksi dokumentaatio ja automaattiset testit sekä Java-sovelluksissa jar-tiedosto
- ▶ Toivottava dokumentaation taso käy ilmi referenssisovelluksesta <https://github.com/mluukkai/OtmTodoApp>

Hyvän aiheen ominaisuudet

- ▶ **Itseäsi kiinnostava aihe**
- ▶ Riittävän mutta ei liian laaja
 - ▶ Vältä eppisiä aiheita, aloita riittävän pienestä
 - ▶ Valitse aihe, jonka perustoiminnallisuuden saa toteutettua nopeasti, mutta jota saa myös laajennettua helposti
 - ▶ Hyvässä aiheessa on muutamia logiikkaluokkia, tiedostojen tai tietokannan käsittelyä ja sovelluslogiikasta eriytetty käyttöliittymä

Hyvän aiheen ominaisuudet

- ▶ **Itseäsi kiinnostava aihe**
- ▶ Riittävän mutta ei liian laaja
 - ▶ Vältä eppisiä aiheita, aloita riittävän pienestä
 - ▶ Valitse aihe, jonka perustoiminnallisuuden saa toteutettua nopeasti, mutta jota saa myös laajennettua helposti
 - ▶ Hyvässä aiheessa on muutamia logiikkaluokkia, tiedostojen tai tietokannan käsittelyä ja sovelluslogiikasta eriytetty käyttöliittymä
- ▶ Kurssilla pääpaino on
 - ▶ Toimivuus ja varautuminen virhetilanteisiin
 - ▶ Luokkien vastuut
 - ▶ Ohjelman selkeä rakenne
 - ▶ Laajennettavuus ja ylläpidettävyys

Hyvän aiheen ominaisuudet

- ▶ **Itseäsi kiinnostava aihe**
- ▶ Riittävän mutta ei liian laaja
 - ▶ Vältä eepisiä aiheita, aloita riittävän pienestä
 - ▶ Valitse aihe, jonka perustoiminnallisuuden saa toteutettua nopeasti, mutta jota saa myös laajennettua helposti
 - ▶ Hyvässä aiheessa on muutamia logiikkaluokkia, tiedostojen tai tietokannan käsittelyä ja sovelluslogiikasta eriytetty käyttöliittymä
- ▶ Kurssilla pääpaino on
 - ▶ Toimivuus ja varautuminen virhetilanteisiin
 - ▶ Luokkien vastuut
 - ▶ Ohjelman selkeä rakenne
 - ▶ Laajennettavuus ja ylläpidettävyys
- ▶ **Tällä kurssilla ei ole tärkeää:**
 - ▶ Tekoäly
 - ▶ Grafiikka
 - ▶ Tietoturva
 - ▶ Tehokkuus

Huonon aiheen ominaisuuksia

- ▶ Kannattaa yrittää välttää aiheita, joissa pääpaino on tiedon säilömisessä tai liian monimutkaisessa käyttöliittymässä
- ▶ Paljon tietoa säilövät, esim. yli 3 tietokantataulua tarvitsevat sovellukset sopivat yleensä paremmin kurssille
Tietokantasovellus
- ▶ Käyttöliittymäkeskeisissä aiheissa voi olla vaikea keksiä sovelluslogiikkaa, joka on enemmän tämän kurssin painopiste

- ▶ Hyötyohjelmat
 - ▶ Aritmetiikan harjoittelua
 - ▶ Tehtävägeneraattori, joka antaa käyttäjälle tehtävän sekä mallivastauksen (esim. matematiikkaa, fysiikkaa, kemiaa, ...)
 - ▶ Code Snippet Manageri
 - ▶ Laskin, funktiolaskin, graafinen laskin
 - ▶ Budjetointisovellus
 - ▶ Opintojen seurantasovellus
 - ▶ HTML WYSIWYG-editor (What you see is what you get)

Esimerkkejä aiheista

- ▶ Reaaliaikaiset pelit
 - ▶ Tetris
 - ▶ Pong
 - ▶ Pacman
 - ▶ Tower Defence
 - ▶ Asteroids
 - ▶ Space Invaders
 - ▶ Yksinkertainen tasohyppypeli, esimerkiksi The Impossible Game

Esimerkkejä aiheista

- ▶ Vuoropohjaiset pelit
 - ▶ Tammi
 - ▶ Yatzy
 - ▶ Miinaharava
 - ▶ Laivanupotus
 - ▶ Yksinkertainen roolipeli tai luolastoseikkailu
 - ▶ Sudoku
 - ▶ Muistipeli
 - ▶ Ristinolla (mielivaltaisen kokoisella ruudukolla?)

Esimerkkejä aiheista

- ▶ Korttipelit
 - ▶ En Garde
 - ▶ Pasiassi
 - ▶ UNO
 - ▶ Texas Hold'em
- ▶ Omaan tieteenalaan, sivuaineeseen tai harrastukseen liittyvät hyötyohjelmat
 - ▶ Yksinkertainen fysiikkasimulaattori
 - ▶ DNA-ketjujen tutkija
 - ▶ Keräilykorttien hallintajärjestelmä
 - ▶ Fraktaaligeneraattori

Arvosteluperusteet tarkemmin

- ▶ Kurssin maksimi on 60 pistettä
- ▶ Ennen loppupalautusta jaossa 19 pistettä
 - ▶ Viikkodeadlinet 17p
 - ▶ Koodikatselmointi 2p

Arvosteluperusteet tarkemmin

- ▶ Kurssin maksimi on 60 pistettä
- ▶ Ennen loppupalautusta jaossa 19 pistettä
 - ▶ Viikkodeadlinet 17p
 - ▶ Koodikatselmointi 2p
- ▶ Loppupalautus ratkaise 41 pisteen kohtalon
 - ▶ Dokumentaatio 12p
 - ▶ Testaus 5p
 - ▶ Lopullinen ohjelma 24p
 - ▶ Laajuus, ominaisuudet ja koodin laatu

Arvosteluperusteet tarkemmin

- ▶ Kurssin maksimi on 60 pistettä
- ▶ Ennen loppupalautusta jaossa 19 pistettä
 - ▶ Viikkodeadlinet 17p
 - ▶ Koodikatselmointi 2p
- ▶ Loppupalautus ratkaise 41 pisteen kohtalon
 - ▶ Dokumentaatio 12p
 - ▶ Testaus 5p
 - ▶ Lopullinen ohjelma 24p
 - ▶ Laajuus, ominaisuudet ja koodin laatu
- ▶ Arvosanaan 1 riittää 30 pistettä, arvosanaan 5 tarvitaan noin 55 pistettä
- ▶ Läpipääsyyn vaatimuksena on lisäksi vähintään 10 pistettä lopullisesta ohjelmasta

Harjoitustyön vaikutus kurssipisteisiin

Ohjelman pisteet jakautuvat seuraavasti

- ▶ käyttöliittymä 4p
 - ▶ 0p yksinkertainen tekstikäyttöliittymä
 - ▶ 1-2p monimutkainen tekstikäyttöliittymä
 - ▶ 2-3p yksinkertainen graafinen käyttöliittymä
 - ▶ 4p laaja graafinen käyttöliittymä

Harjoitustyön vaikutus kurssipisteisiin

Ohjelman pisteet jakautuvat seuraavasti

- ▶ käyttöliittymä 4p
 - ▶ 0p yksinkertainen tekstikäyttöliittymä
 - ▶ 1-2p monimutkainen tekstikäyttöliittymä
 - ▶ 2-3p yksinkertainen graafinen käyttöliittymä
 - ▶ 4p laaja graafinen käyttöliittymä
- ▶ tiedon pysyväistalletus 4p
 - ▶ 0p ei pysyväistalletusta
 - ▶ 1-2p tiedosto
 - ▶ 3-4p tietokanta
 - ▶ 3-4p internet

Harjoitustyön vaikutus kurssipisteisiin

Ohjelman pisteet jakautuvat seuraavasti

- ▶ käyttöliittymä 4p
 - ▶ 0p yksinkertainen tekstikäyttöliittymä
 - ▶ 1-2p monimutkainen tekstikäyttöliittymä
 - ▶ 2-3p yksinkertainen graafinen käyttöliittymä
 - ▶ 4p laaja graafinen käyttöliittymä
- ▶ tiedon pysyväistalletus 4p
 - ▶ 0p ei pysyväistalletusta
 - ▶ 1-2p tiedosto
 - ▶ 3-4p tietokanta
 - ▶ 3-4p internet
- ▶ sovelluslogiikan kompleksisuus 3p
- ▶ ohjelman laajuus 5p

Harjoitustyön vaikutus kurssipisteisiin

Ohjelman pisteet jakautuvat seuraavasti

- ▶ käyttöliittymä 4p
 - ▶ 0p yksinkertainen tekstikäyttöliittymä
 - ▶ 1-2p monimutkainen tekstikäyttöliittymä
 - ▶ 2-3p yksinkertainen graafinen käyttöliittymä
 - ▶ 4p laaja graafinen käyttöliittymä
- ▶ tiedon pysyväistalletus 4p
 - ▶ 0p ei pysyväistalletusta
 - ▶ 1-2p tiedosto
 - ▶ 3-4p tietokanta
 - ▶ 3-4p internet
- ▶ sovelluslogiikan kompleksisuus 3p
- ▶ ohjelman laajuus 5p
- ▶ ulkoisten kirjastojen hyödyntäminen 1p
- ▶ suorituskelpoinen jar-tiedosto 1p
- ▶ koodin laatu 6p

Harjoitustyön toimivuus

- ▶ Koneiden konfiguraatioissa on eroja, ja tällä kurssilla *ei riitä* että harjoitustyössä tekemäsi sovellus toimii vain omalla koneellasi

Harjoitustyön toimivuus

- ▶ Koneiden konfiguraatioissa on eroja, ja tällä kurssilla *ei riitä* että harjoitustyössä tekemäsi sovellus toimii vain omalla koneellasi
- ▶ Harjoitustyösi pitää pystyä joka viikko suorittamaan, kääntämään ja testaamaan komentoriviltä käsin laitoksen linux-koneilla (tai uusimmat päivitykset sisältävällä cubbli-linuxilla)
 - ▶ muussa tapauksessa työtä ei tarkasteta ja menetät viikon/loppupalautuksen pisteet
- ▶ Varminta käyttää Javan versiota 11

Harjoitustyön toimivuus

- ▶ Koneiden konfiguraatioissa on eroja, ja tällä kurssilla *ei riitä* että harjoitustyössä tekemäsi sovellus toimii vain omalla koneellasi
- ▶ Harjoitustyösi pitää pystyä joka viikko suorittamaan, kääntämään ja testaamaan komentoriviltä käsin laitoksen linux-koneilla (tai uusimmat päivitykset sisältävällä cubbli-linuxilla)
 - ▶ muussa tapauksessa työtä ei tarkasteta ja menetät viikon/loppupalautuksen pisteet
- ▶ Varminta käyttää Javan versiota 11
- ▶ Pääset testaamaan ohjelmaasi laitoksen koneella myös kotoa käsin käyttämällä etätyöpöytää

Koodin laatuvaatimukset

- ▶ Kurssin tavoitteena on, että tuotoksesi voisi ottaa kuka tahansa kaverisi tai muu opiskelija ylläpidettäväksi ja laajennettavaksi
- ▶ Lopullisessa palautuksessa tavoitteena on *Clean code* eli selkeä, ylläpidettävä ja toimivaksi testattu koodi