

# Ohjelmistotekniikka

Matti Luukkainen, Olli Keski-Hyynilä ja 6 ohjaajaa

29.10.2018

# Ohjelmistotekniikka

- ▶ Kurssilla tutustutaan ohjelmistokehityksen periaatteisiin sekä menetelmiin ja sovelletaan niitä toteuttamalla pienehkö harjoitustyö
- ▶ Kurssi nykyään osa *aineopintoja*
- ▶ Pakollisina *esitietoina*
  - ▶ Ohjelmoinnin jatkokurssi
  - ▶ Tietokantojen perusteet
- ▶ Hyödyllinen esitieto: Tietokone työvälineenä
- ▶ Kurssimateriaali  
<https://github.com/mluukkai/Ohjelmistotekniikka2018>
- ▶ Kurssi on sisällöltään ja kurssikoodiltaan sama kuin viime kevään kurssi *Ohjelmistotekniikan menetelmät*
  - ▶ viime kevään OTM taas on nimeltään sama, mutta sisällöltään radikaalisti poikkeava ennen vuotta 2018 pidetystä kurssista  
OTM

- ▶ 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 (keväällä 2019)
- ▶ Muista siis varata riittävästi aikaa (10-15h viikossa) koko periodin ajaksi!

# Luento, deadlinet ja ohjaus

- ▶ Kurssilla on vain yksi luento **nyt** eli ma 29.10. klo 14-16 B123
- ▶ Laskareiden ja harjoitustyön välitavoitteiden viikoittaiset deadlinet *tiistaina klo 23:59*
- ▶ Paja salissa BK107

alku	ma	ti	ke	to	pe
10					x
12	x	x	x		
14	x	x	x		
16					
18					

- ▶ Paja alkaa huomenna 30.10

- ▶ Jaossa 60 pistettä jotka jakautuvat seuraavasti
  - ▶ Viikkodeadlinet 17p
  - ▶ Koodikatselmointi 2p
  - ▶ Dokumentaatio 10p
  - ▶ Testaus 7p
  - ▶ 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

# Ohjelmistotuotanto

- ▶ Kun ollaan tekemässä suurempaa ohjelmistoa ulkoiselle asiakkaalle, tarvitaan systemaattinen työskentelymenetelmä
  - ▶ muuten riskinä mm. että lopputulos ei vastaa asiakkaan tarvetta



# Ohjelmistotuotanto

- ▶ Kun ollaan tekemässä suurempaa ohjelmistoa ulkoiselle asiakkaalle, tarvitaan systemaattinen työskentelymenetelmä
  - ▶ muuten riskinä mm. että lopputulos ei vastaa asiakkaan tarvetta
- ▶ Sovellettavasta menetelmästä riippumatta ohjelmiston systemaattinen kehittäminen, eli *ohjelmistotuotanto* (engl. Software engineering) sisältää useita erilaisia aktiviteetteja/vaiheita
  - ▶ *vaatimusmäärittelyssä* selvitetään kuinka ohjelmiston halutaan toimivan
  - ▶ *suunnittelussa* mietitään, miten halutun kaltainen ohjelmisto tulisi rakentaa
  - ▶ *toteutusvaiheessa* määritelty ja suunniteltu ohjelmisto koodataan
  - ▶ *testauksen* tehtävä on varmistaa ohjelmiston laatu
    - ▶ ei ole liian buginen
    - ▶ toimii kuten vaatimusmäärittely sanoo
  - ▶ *ylläpitovaiheessa* ohjelmisto on jo käytössä ja siihen tehdään bugikorjauksia ja mahdollisia laajennuksia

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

- ▶ Kartoitetaan ohjelman tulevien käyttäjien tai tilaajan kanssa, mitä toiminnallisuutta ohjelmaan halutaan
- ▶ Ohjelman toiminnalle siis määritellään asiakkaan vaatimukset
- ▶ Tämän lisäksi kartoitetaan ohjelman toimintaympäristön ja toteutusteknologian järjestelmälle asettamia rajoitteita

# Vaatimusmäärittely

- ▶ Kartoitetaan ohjelman tulevien käyttäjien tai tilaajan kanssa, mitä toiminnallisuutta ohjelmaan halutaan
- ▶ Ohjelman toiminnalle siis määritellään asiakkaan vaatimukset
- ▶ Tämän lisäksi kartoitetaan ohjelman toimintaympäristön ja toteutusteknologian järjestelmälle asettamia rajoitteita
- ▶ Tuloksena on useimmiten jonkinlainen dokumentti, johon vaatimukset kirjataan
- ▶ Dokumentin muoto vaihtelee suuresti, se voi olla paksu mapillinen papereita tai vaikkapa joukko postit-lappuja

# Vaatimusten kirjaaminen

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

# Vaatimusten kirjaaminen

- ▶ On olemassa lukuisia tapoja dokumentoida vaatimuksen
- ▶ Kurssin ennen tätä vuotta pidetyissä versioissa käyttäjien vaatimukset dokumentointiin *käyttötapauksina* (engl. use case)
  - ▶ tapa on jo vanhahtava ja hylkäämme 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

Osoitteesta <https://github.com/mluukkai/OtmTodoApp> löytyy sovellus, joka havainnollistaa monia kurssin asioita ja toimii 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

Osoitteesta <https://github.com/mluukkai/OtmTodoApp> löytyy sovellus, joka havainnollistaa monia kurssin asioita ja toimii mallina omalle harjoitustyölle

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

Katsotaan esimerkkinä Todo-sovelluksen vaatimusmäärittelyä

- ▶ Vaatimusmäärittely aloitetaan tunnistamalla järjestelmän erityyppiset *käyttäjäroolit*
- ▶ Todo-sovelluksesta tunnistetaan kaksi käyttäjäroolia
  - ▶ normaalit käyttäjät
  - ▶ laajemmilla oikeuksilla varustetut ylläpitäjät
- ▶ Kun sovelluksen käyttäjäroolit ovat selvillä, mietitään mitä toiminnallisuuksia kunkin käyttäjäroolin halutaan pystyvän tekemään sovelluksen avulla



# 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 todon
  - ▶ kirjaantunut käyttäjä voi merkitä todon tehdyksi, jolloin todo 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 todon
  - ▶ kirjaantunut käyttäjä voi merkitä todon tehdyksi, jolloin todo 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
  - ▶ käyttäjien ja töiden 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
  - ▶ käyttäjien ja töiden tiedot talletetaan paikallisen koneen levyille
- ▶ Vaatimusmäärittelyn aikana hahmotellaan yleensä myös sovelluksen käyttöliittymä

Vaatimusten kirjaamisesta voi ottaa tarkemmin mallia sovelluksen GitHub-repositoriosta <https://github.com/mluukkai/OtmTodoApp>

# Suunnittelu

- ▶ Ohjelmiston suunnittelu jakautuu yleensä kahteen erilliseen vaiheeseen

- ▶ Ohjelmiston suunnittelu jakautuu yleensä 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. tietokantoihin ja ulkoisiin rajapintoihin

# Suunnittelu

- ▶ Ohjelmiston suunnittelu jakautuu yleensä 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. tietokantoihin ja ulkoisiin rajapintoihin
- ▶ Arkkitehtuurisuunnittelua tarkoittaa *oliosuunnittelu*, missä mietitään ohjelmiston yksittäisten komponenttien rakennetta
  - ▶ minkälaisista luokista komponentit koostuvat
  - ▶ miten luokat kutsuvat toistensa metodeja sekä mitä apukirjastoja ne käyttävät
- ▶ Myös ohjelmiston suunnittelu, erityisesti sen arkkitehtuuri dokumentoidaan usein jollain tavalla

# Testaus

- ▶ Toteutuksen yhteydessä ja sen jälkeen järjestelmää testataan
- ▶ Testausta on monentasoista
- ▶ *Yksikkötestauksessa* tutkitaan yksittäisten metodien ja luokkien toimintaa.
  - ▶ Yksikkötestauksen tekee usein testattavan komponentin ohjelmoija



# Testaus

- ▶ Toteutuksen yhteydessä ja sen jälkeen järjestelmää testataan
- ▶ Testausta on monentasoista
- ▶ *Yksikkötestauksessa* tutkitaan yksittäisten metodien ja luokkien toimintaa.
  - ▶ Yksikkötestauksen tekee usein testattavan komponentin ohjelmoija
- ▶ Kun erikseen ohjelmoidut luokat yhdistetään, suoritetaan *integraatiotestaus*
  - ▶ varmistetaan erillisten osien yhteentoimivuus
  - ▶ integraatiotestaus tapahtuu useimmiten ohjelmoijien toimesta

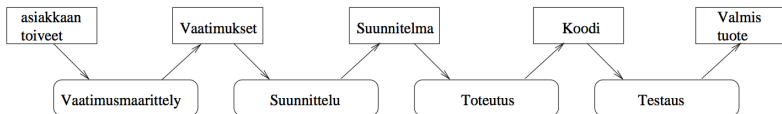
# Testaus

- ▶ Toteutuksen yhteydessä ja sen jälkeen järjestelmää testataan
- ▶ Testausta on monentasoista
- ▶ *Yksikkötestauksessa* tutkitaan yksittäisten metodien ja luokkien toimintaa.
  - ▶ Yksikkötestauksen tekee usein testattavan komponentin ohjelmoija
- ▶ Kun erikseen ohjelmoidut luokat yhdistetään, suoritetaan *integraatiotestaus*
  - ▶ varmistetaan erillisten osien yhteentoimivuus
  - ▶ integraatiotestaus tapahtuu useimmiten ohjelmoijien toimesta
- ▶ *Järjestelmätestauksessa* testataan ohjelmistoa kokonaisuutena ja verrataan, että se toimii vaatimusdokumentissa sovitun määritelmän mukaisesti
  - ▶ järjestelmätestaus suoritetaan ohjelman todellisen käyttöliittymän kautta
  - ▶ järjestelmätestauksen saattaa tapahtua erillisen laadunhallintatiimin toimesta

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

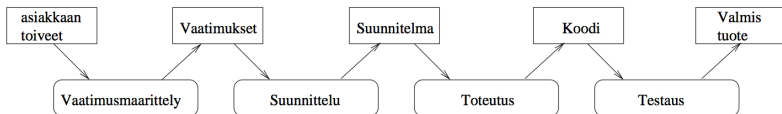
# Vesiputousmalli

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



# Vesiputousmalli

- ▶ Ohjelmistojen 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 ennen suunnittelun ja ohjelmoinnin aloittamista
  - ▶ Näin ei useinkaan ole. On lähes mahdotonta, että asiakkaat osaisivat tyhjentävästi ilmaista kaikki ohjelmistolle asettamansa vaatimukset
  - ▶ Vasta käyttäessään valmista ohjelmistoa asiakkaat alkavat ymmärtää, mitä he olisivat ohjelmalta halunneet
  - ▶ Vaikka vaatimukset olisivat kunnossa laatimishetkellä, saattaa toimintaympäristö muuttua kehitysaikana niin ratkaisevasti, että valmistuessaan ohjelmisto on vanhentunut

# Vesiputousmallin ongelmat

- ▶ Mallin toimivuus perustuu siihen oletukseen, että vaatimukset pystytään määrittelemään täydellisesti ennen suunnittelun ja ohjelmoinnin aloittamista
  - ▶ Näin ei useinkaan ole. On lähes mahdotonta, että asiakkaat osaisivat tyhjentävästi ilmaista kaikki ohjelmistolle asettamansa vaatimukset
  - ▶ Vasta käyttäessään valmista ohjelmistoa asiakkaat alkavat ymmärtää, mitä he olisivat ohjelmalta halunneet
  - ▶ Vaikka vaatimukset olisivat kunnossa laatimishetkellä, saattaa toimintaympäristö muuttua kehitysaikana niin ratkaisevasti, että 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
- ▶ Ketterä ohjelmistokehityksen 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
- ▶ Ketterä ohjelmistokehityksen alussa kartoitetaan pääpiirteissään ohjelmiston vaatimuksia ja hahmotellaan ohjelmiston alustava arkkitehtuuri
- ▶ Tämän jälkeen suoritetaan useita *iteraatioita* (joista käytetään yleisesti myös nimitystä sprintti), 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
- ▶ Ketterä ohjelmistokehityksen alussa kartoitetaan pääpiirteissään ohjelmiston vaatimuksia ja hahmotellaan ohjelmiston alustava arkkitehtuuri
- ▶ Tämän jälkeen suoritetaan useita *iteraatioita* (joista käytetään yleisesti myös nimitystä sprintti), 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

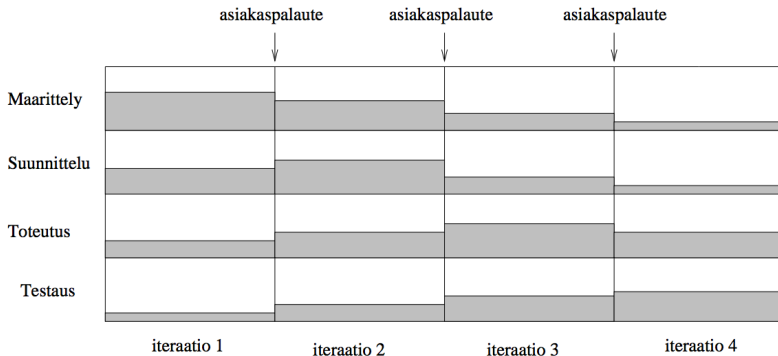


Figure 1

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

# 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
- ▶ 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 muutamaia vaihtoehtoja. Käytämme monille TiKaPesta tuttua *mavenia*

- ▶ Ohjelmistojen testaus tapahtuu nykyään ainakin yksikkö- ja integraatiotestien osalta 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

# Checkstyle

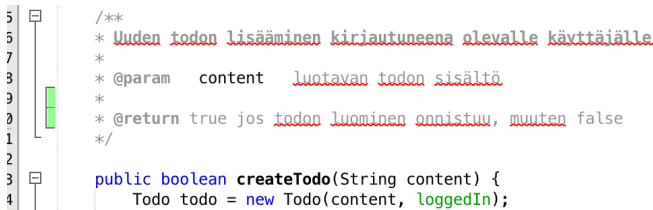
- ▶ Automaattisten testien lisäksi koodille voidaan määritellä erilaisia automaattisesti tarkastettavia tyylillisiä sääntöjä
- ▶ Näiden avulla ylläpidetään koodin luettavuutta ja varmistetaan, että joka puolella koodia noudatetaan samoja tyylillisiä konventioita
- ▶ Käytämme kurssilla tarkoitukseen *Checkstyle*-nimistä työkalua



# Checkstyle

- ▶ Automaattisten testien lisäksi koodille voidaan määritellä erilaisia automaattisesti tarkastettavia tyylillisiä sääntöjä
- ▶ Näiden avulla ylläpidetään koodin luettavuutta ja varmistetaan, että joka puolella koodia noudatetaan 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  * Uuden todon lisääminen kirjautuneena olevalle käyttäjälle  
7  *  
8  * @param content luotavan todon sisältö  
9  *  
10 * @return true jos todon luominen onnistuu, muuten false  
11 */  
12  
13    
14 public boolean createTodo(String content) {  
    Todo todo = new Todo(content, loggedIn);  
}
```

Figure 2

## 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
<b>Todo</b>	Yksittäistä työtä kuvaava luokka
<b>TodoService</b>	Sovelluslogiikasta vastaava luokka
<b>User</b>	Järjestelmän käyttäjää edustava luokka

OVERVIEW
**PACKAGE**
CLASS
USE
TREE
DEPRECATED
INDEX
HELP

Figure 3

NetBeans osaa näyttää ohjelmoidessa koodiin määritellyn JavaDocin

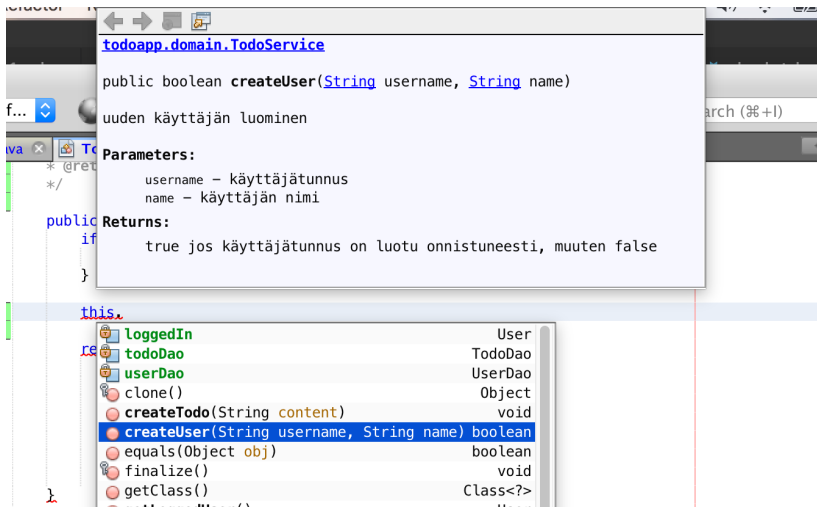


Figure 4

UML

- Ohjelmistojen dokumentoinnissa ja suunnittelun tukena tarvitaan usein jonkinlaisia ohjelman rakennetta ja toimintaa havainnollistavia kaavioita

# UML ja dokumentointi

- ▶ Ohjelmistojen dokumentoinnissa ja suunnittelun tukena tarvitaan usein jonkinlaisia 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

- ▶ Ohjelmistojen dokumentoinnissa ja suunnittelun tukena tarvitaan usein jonkinlaisia 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

- ▶ Luokkakaavioiden käyttötarkoitus on ohjelman 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

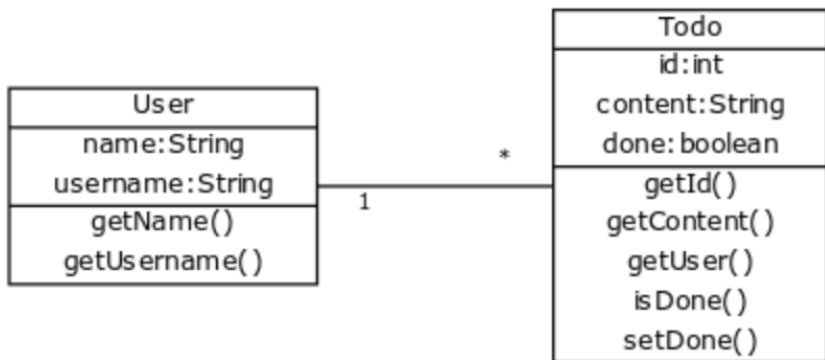


Figure 5

# Todo-sovelluksen tietosisällön luokkakaavio

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



Figure 6

- 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

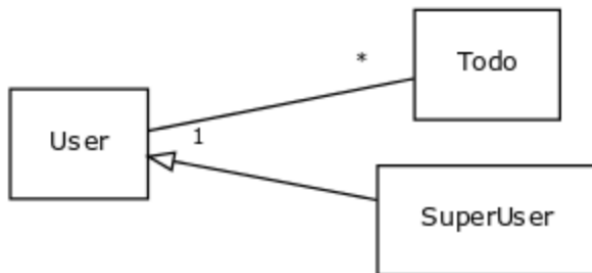


Figure 7

- ▶ Rajapinnan toteutus merkitään samalla tavalla eli valkoisella nuolenpäällä

# Riippuvuus

- ▶ UML-kaavioissa olevat “viivat” kuvaavat luokkien olioiden välistä *pysyvää yhteyttä*
- ▶ Joissain tilanteissa on mielekästä merkata kaavioihin myös ei-pysyvää suhdetta kuvaava katkoviiva, eli *riippuvuus*

# Riippuvuus

- ▶ UML-kaavioissa olevat “viivat” kuvaavat luokkien olioiden välistä *pysyvää yhteyttä*
- ▶ Joissain tilanteissa on mielekästä merkata kaavioihin myös ei-pysyvää suhdetta kuvaava katkoviiva, eli *riippuvuus*
- ▶ Ohjelmoinnin perusteista tutussa Unicafe-tehtävässä Kassapäätte *käyttää* hetkellisesti *maksukorttia* lounaiden maksuun
- ▶ Kassapäätteen ja maksukortin välillä ei kuitenkaan ole pidempiaikaista suhdetta

# Riippuvuus

- ▶ UML-kaavioissa olevat “viivat” kuvaavat luokkien olioiden välistä *pysyvää yhteyttä*
- ▶ Joissain tilanteissa on mielekästä merkata kaavioihin myös ei-pysyvää suhdetta kuvaava katkoviiva, eli *riippuvuus*
- ▶ Ohjelmoinnin perusteista tutussa Unicafe-tehtävässä Kassapääte *käyttää* hetkellisesti *maksukorttia* lounaiden maksuun
- ▶ Kassapääteen ja maksukortin välillä ei kuitenkaan ole pidempiaikaista suhdetta

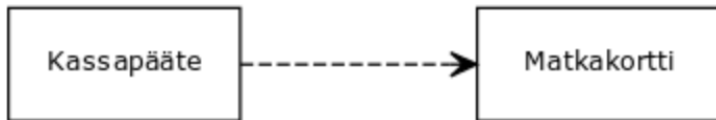


Figure 8

# Maksukortti ja kassapääte

```
public class Maksukortti {  
    private double saldo;  
    // ...  
}
```

```
public class Kassapaate {  
    private int edulliset;  
    private int maukkaat;  
  
    public boolean syoEdullisesti(Maksukortti kortti) {  
        // ...  
        kortti.otaRahaa(EDULLISEN_HINTA);  
    }  
}
```

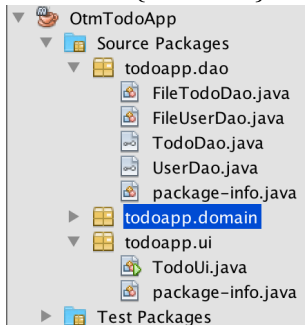


# Pakkauskaavio

# Pakkauskaavio

- Todo-sovelluksen koodi on sijoitettu *pakkauksiin* seuraavasti:

..... { .columns } :: { .column width="50%" }



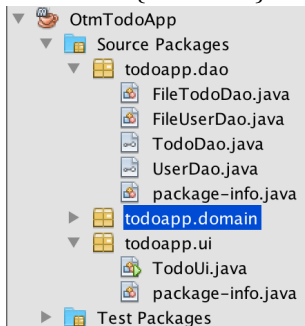
::: ::: { .column width="50%" }

.....

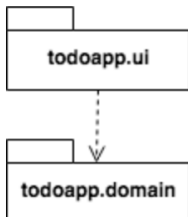
# Pakkauskaavio

- Pakkausrakenne voidaan kuvata UML:ssä pakkauskaaviolla

..... { .columns } ::: { .column width="50%" }

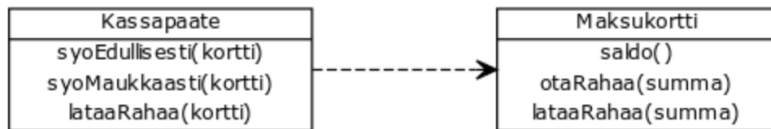


... : { .column width="50%" }



# Toiminnan 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

- ▶ Tietokantojen perusteiden viikolla 4 on lyhyt maininta sekvenssikaavioista
- ▶ Sekvenssikaaviot on alunperin kehitetty kuvaamaan verkossa olevien ohjelmien keskinäisen kommunikoinnin etenemistä
- ▶ Sekvenssikaaviot sopivat jossain määrin myös kuvaamaan myös sitä, miten ohjelman oliot kutsuvat toistensa metodeja suorituksen aikana

# Sekvenssikaavio

Koodia katsomalla näemme, että lounaan maksaminen tapahtuu siten, että ensin kassapäät kysyy kortin saldoa ja jos se on riittävä, vähentää kassapäät lounaan hinnan kortilta ja palauttaa *true*

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

# Onnistunut ostos sekvenssikaaviona

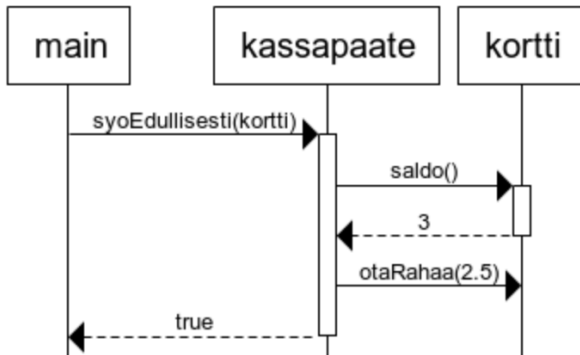


Figure 9

- ▶ Sekvenssikaaviossa oliot ovat laatikoita, joista lähtee alaspäin olion “elämänlanka”
- ▶ Aika etenee ylhäältä alas
- ▶ Metodikutsut ovat nuolia, jotka yhdistävää kutsuvan ja kutsutun olion elämänlangat



# Epäonnistunut ostos sekvenssikaaviona

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

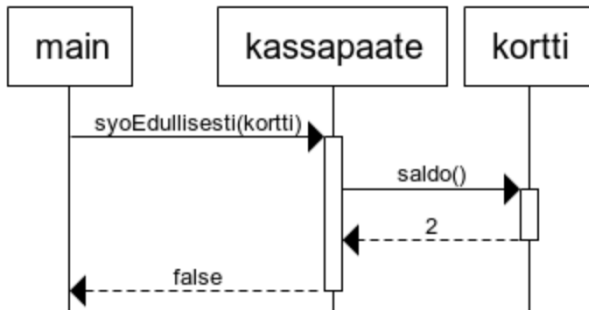


Figure 10

# Epäonnistunut ostos sekvenssikaaviona

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

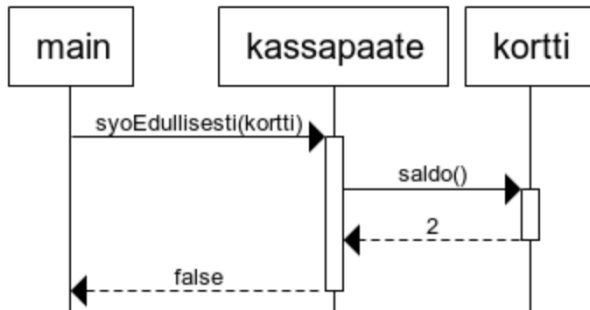


Figure 10

- ▶ 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
- ▶ Tavoitteena on soveltaa ja syventää ohjelmoinnin perus- ja jatkokursseilla opittuja taitoja ja harjoitella tiedon omatoimista etsimistä
- ▶ Harjoitustyötä tehdään itsenäisesti, mutta tarjolla on runsaasti pajaohjausta

# 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 keväällä 2019), 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
- ▶ Ohjelmakoodin muuttujat, luokat ja metodit **kirjoitetaan englanniksi**
- ▶ Dokumentaatio voidaan kirjoittaa joko suomeksi tai englanniksi

- ▶ Harjoitustyön ohjelmointikieli on Java
- ▶ 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 osa toiminnallisuudesta
  - ▶ ohjelman ydin pidetään koko ajan toimivana, uutta toiminnallisuutta lisäten, kunnes tavoiteltu laajuus on saavutettu
- ▶ Ohjelman rakenteeseen kannattaa kysyä vinkkejä pajasta, sekä ottaa mallia Ohjelmoinnin jatkokurssilta sekä kurssisivuilta löytyvistä vihjeistä



# Ohjelman toteutus

- ▶ Toteutus etenee “iteratiivisesti ja inkrementaalisesti”
  - ▶ Heti ensimmäisellä viikolla toteutetaan pieni osa toiminnallisuudesta
  - ▶ ohjelman ydin pidetään koko ajan toimivana, uutta toiminnallisuutta lisäten, kunnes tavoiteltu laajuus on saavutettu
- ▶ Ohjelman rakenteeseen kannattaa kysyä vinkkejä pajasta, sekä ottaa mallia Ohjelmoinnin jatkokurssilta sekä kurssisivuilta löytyvistä vihjeistä
- ▶ 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 osa toiminnallisuudesta
  - ▶ ohjelman ydin pidetään koko ajan toimivana, uutta toiminnallisuutta lisäten, kunnes tavoiteltu laajuus on saavutettu
- ▶ Ohjelman rakenteeseen kannattaa kysyä vinkkejä pajasta, sekä ottaa mallia Ohjelmoinnin jatkokurssilta sekä kurssisivuilta löytyvistä vihjeistä
- ▶ 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
- ▶ 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
- ▶ Lopullisessa palautuksessa on oltava lähdekoodin lisäksi dokumentaatio ja automaattiset testit sekä jar-tiedosto, joka mahdollistaa ohjelman suorittamisen NetBeansin ulkopuolella.
- ▶ Toivottava dokumentaation taso käy ilmi referenssisovelluksesta <https://github.com/mluukkai/OtmTodoApp>

# 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 monimutkaisessa käyttöliittymässä
- ▶ Paljon tietoa säilövät, esim. yli 3 tietokantataulua tarvitsevat sovellukset sopivat yleensä paremmin Tietokantasovellus-kurssille
- ▶ Käyttöliittymäkeskeisissä aiheissa (esim. tekstieditori) 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, ...)
- Telegram- tai Slack-botti
- Code Snippet Manageri
- Laskin, funktiolaskin, graafinen laskin
- Budjetointi-sovellus
- 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 20 pistettä
  - ▶ Viikkodeadlinet 17p
  - ▶ Koodikatselmointi 3p
- ▶ Loppupalautus ratkaise 40 pisteen kohtalon
  - ▶ Dokumentaatio 10p
  - ▶ Testaus 7p
  - ▶ Lopullinen ohjelma 23p
    - ▶ 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 (yht 43) 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 5p
- ▶ suorituskelpoinen jar-tiedosto 1p
- ▶ koodin laatu 6p

# 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
- ▶ **Nimentä**
  - ▶ Käytä mahdollisimman kuvaavia nimiä kaikkialla
  - ▶ Luokkien nimet aina isolla alkukirjaimella
  - ▶ Metodit, attribuutit, parametrit ja muuttujat aina *camel/Case*
  - ▶ Muuttujat, joilla on iso käyttöalue, tulee olla erittäin selkeästi (vaikka pitkästi) nimettyjä.
  - ▶ Lyhyen metodin sisäisille muuttujille riittää yleensä lyhyt nimi
  - ▶ Jos metodia käytetään vähän, tulee nimen olla mahdollisimman kuvaava
  - ▶ Jos metodia käytetään useassa kohdassa koodia, voi sen nimi olla lyhyt ja ytimekäs

## ► Ei pitkiä metodeja

- Sovelluslogiikan metodin pituuden tulee ilman erittäin hyvää syytä olla korkeintaan 10 riviä
- Pitkät metodit tulee jakaa useampiin metodeihin
- Yksi metodi - yksi pieni tehtävä (Single Responsibility)
  - Helpottaa myös testaamista

## ► Ei copy-pastea

- Toistuvan koodin saa lähes aina hävitettyä
- Tapauksesta riippuen luo metodi tai ylikuokka, joka sisältää toistuvan koodin

## ► Luokkien Single Responsibility

- Luokkien tulisi hoitaa vain yhtä asiaa
- Eriyksen tärkeää on erottaa käyttöliittymä ja sovelluslogiikka
  - Kaikki tulostaminen tulisi tapahtua käyttöliittymässä
  - Sovelluslogiikkaan liittyviä operaatioita ei tehdä käyttöliittymässä
- Toisaalta tiettyä asiaa ei pidä hoitaa useissa eri luokissa
- Esimerkiksi tiedoston lukemista tai -kirjoittamista EI tulisi löytyä useasta luokasta
  - Tee oma luokka tiedostojen käsittelylle

## ► Pakkaukset

- << Default package >> Ei saa olla käytössä
- Luokat tulee jakaa loogisesti pakkauksiin
  - Pakkausten nimet aina pienellä (*lowercase*)
- Kaikkien pakkausten tulee olla yhden juuripakkauksen alla, esim. `fi.omanimi`
  - Sovelluslogiikkapakkaus olisi näin tehtynä siis `fi.omanimi.logics`, käyttöliittymä `fi.omanimi.gui`
- Yhdessä pakkauksessa yksi kokonaisuus
  - Esim. yhdessä pakkauksessa käyttäjätileihin liittyvät luokat
  - Toisessa muu logiikka
  - Kolmannessa käyttöliittymän luokat
- Myös testipakkausten nimentä tulee olla oikea

# Yleiset laatuvaatimukset

- ▶ Lopulliseen arvosteluun palautetun ohjelman tulee toimia oikein
  - ▶ Ohjelma ei saa missään tilanteessa kaatua
  - ▶ Ohjelma ei saa printata Exceptioneita (Stack tracea) komentoriville, vaikka virhe ei kaataisi ohjelmaa
- ▶ Varaudu siihen, että käyttäjä yrittää antaa väärää syötearvoja
  - ▶ Esim. ohjelmasi haluaa numeron, tyhmä käyttäjä syöttää tekstiä
- ▶ Pelien sääntöjen tulisi toimia oikein
  - ▶ Esim. muistipelissä ei saa kääntää jo käännettyä palaa
  - ▶ Ristinollassa ei saa asettaa merkkiä ruutuun, jossa on jo merkki
- ▶ Jos ohjelmassasi tapahtuu vakava virhe, ohjelmasi voi esimerkiksi
  - ▶ näyttää käyttäjäystävällisen virheilmoituksen
  - ▶ ja sulkea ohjelman
- ▶ Ohjelmaan jäävät tunnetut ongelmat dokumentoidaan testausdokumenttiin