Redovisningen den 13/1

- Tenta till 12:00
- Redovisningen startar 13:00 i MA136-146 respektive15:15 i Ma156. Varje grupp SKA vara med på ett av tillfällena. bokningssida kommer...
- Projektor och dator (Windows XP ev Vista) med java 1.6 och ppt finns tillgängligt. Vill någon köra MacOSX går det också. Och vill någon använda egen dator så är det OK
- Varje grupp har 10 minuter på sig att presentera sitt spel

Tentan

- 13/1 9-12 i skrivsal 8 östra paviljongen (om de inte flyttat den till ngn annan lokal)
- Förklara begrepp
  - Ex:
    - Vad är en tråd?
    - Vad gör en LayoutManager?
    - Hur fungerar Observer-Observable designmönstret?
- Trådsäkerhet
  - Ni bör kunna avgöra om ett program är trådsäkert och fixa ev brister i ett trådat program
- Inga egentliga större programmeringsuppgifter
- Frågor?

Vad har vi gått igenom

- Reflection
- Inre och Nästlade klasser
- JFC - Swing - GUI programmering
- XML - DOM - SAX - Validering
- Trådar - Trådsäkerhet
- Design patterns
- Databaser
- JSP och Web Services
- JavaME
- JUnit
- Lite C++ och C#

Trådsäkerhet

- Hur åstadkomma
  - synchronized block och metoder
  - Specifikt för GUI
    - SwingUtilities.invokeLater & SwingUtilities.invokeAndWait
- Saker som gör det svårare
  - Komplicerad interaktion mellan trådarna
  - Publika/åtkomliga attribut
  - Att referenser till objekt finns tillgängliga i flera klasser
  - Tips!
    - Håll nere interaktionen mellan trådarna så mkt som möjligt
    - Dokumentera

Design patterns

Designmönster/Design patterns

- Vad är det?
  - Beprövade lösningar till återkommande programmeringsproblem
  - Plattformsoberoende
  - Beskrivs ofta med hjälp av UML
- Baseras på en bok
  - Design Patterns, Gamma/Helm/Johnson/Vlissides (Gang of Four)
- Kan/bör återanvändas
- Ca 25 st i orginalboken
- Ger även en terminologi med vilken man kan uttrycka designidéer på ett enklare sätt

Varför använda Design patterns?

- Beprövade lösningar sparar tid
- ”Man ska inte uppfinna hjulet igen”
- Gör det möjligt för oss att dra nytta av andra personers erfarenheter på ett snabbt sätt

Design patterns

- Om jag t.ex. gör ett program som löser problem X och sedan ska göra ett helt annat system där samma problem X uppstår hur gör jag då för att återanvända lösningen?
- Design patterns är just en hjälpmedel för detta.
- Tillåter samtidigt oss att modifiera lösningarna så att de passar ett speciellt problem.

Design patterns

- Ett design pattern:
  - Visar på hur man ska lösa ett problem rent allmänt (dvs inte i ett visst programspråk)
  - Diskuterar olika för och nackdelar lösningen har.
  - Hänvisningar till andra typer av lösningar på samma problem
  - Ger exempel på hur man kan göra och även på "riktiga" användningar av lösningen.

Klassifikation av design patterns

- Creational Patterns
  - Handlar om hur objekt skapas
- Structural Patterns
  - Handlar om hur klasser och objekt är strukturerade
- Behavioral Patterns
  - Handlar om hur klasser och objekt interagerar

Creational Patterns

- Abstract factory
  - Skapar objekt med okänd klass
- Builder
  - Skiljer skapandet från representationen
- Factory method
  - Sparar instantierandet till subklasser
- Prototype
  - Skapar nya objekt från en prototyp
- Singleton
  - Försäkran om enbart ett objekt av viss klass
  - Exempel: Display i JavaME

Structural Patterns

- Adapter - Konverterar klassens gränssnitt
- Bridge - Skiljer abstraktion från impl
- Composite - Aggregat som träd
  - Ex: Component/Container hierarkin
- Decorator - Lägger till egenskaper
  - Kolla in pizza exemplet
- Facade - Ger enhetligt gränssnitt
  - Kolla in pizza exemplet
  - JOptionPane
- Flyweight - Hanterar många småobjekt
- Proxy - Surrogat för objekt (accesskontröll)

Behavioral Patterns:1

- Chain of responsibility - Överlämnar objekt
- Command - Kommando som objekt
  - Exempel ni stött på Action
- Interpreter - Språk ger grammatik&tolk
- Iterator - Accessar element sekventiellt
  - I java finns interfacet Iterator som används av tex collectionklasserna
- Mediator - Kapslar in objektsamarbete
- Memento - Sparar objektets tillstånd

Behavioral Patterns:2

- Observer - Sprider ett objekts ändring till andra
  - Exempel: Händelselyssnarna
- State - Tillståndsmaskin som byter klass
- Strategy - Kapslar in algoritmer
- Template method - Låter subklassen göra delar av algoritmen
- Visitor - Representerar en operation som ska utföras på en objektstruktur

Designprinciper

- ”Programmera mot ett interface, inte en implementation”
  - Handlar om beroenden mellan klasser
  - Lätt att lägga till beroenden, svårt att ta bort
- Avgränsa det som ändras
  - Det ska vara lätt att byta ut/lägga till delar som kan ändras
- Sträva mot ”lösa” beroenden mellan klasser
  - Oftast ska klasser veta så lite som möjligt om varandra
  - Synlighetsmodifierare