Objects First With Java
A Practical Introduction Using BlueJ

Att formge applikationer

Designing applications

Huvudbegrepp

- Att upptäcka klasser
- CRC-kort
- Gränssnittdesign
- Mönster (Patterns)

Analys och design

- Stort, komplext område.
- Verb/substantiv-metoden kan användas för små problemställningar.
- CRC-kort ger ett stöd för design-processen.

Verb/substantiv-metoden

- Substantiven i en beskrivning anger ‘saker’.
  - Ger upphov till klasser och objekt.
- Verben anger vad som skall göras.
  - Ger upphov till interaktion mellan objekt.
  - Aktioner definierar beteenden - beskrivs av koden i metoder.

En problembeskrivning

•Ett platsbokningssystem för en biografkedja skall hantera bokningar för flera biografer.
•Varje biograf har sina platser arrangerade i rader.
•Besökare kan reservera plats och får då ett radnummer och ett stolsnummer.
•Man kan boka flera platser intill varandra.
•Varje bokning gäller för en filmvisning på en given tid.
•Filmer visas på fastställda dagar och tider och schemaläggs på de biografer där de visas.
•Platsbokningssystemet lagrar kundernas telefonnummer.

Substantiv och verb

Cinema booking system
Stores (seat bookings)
Stores (telephone number)

Theatre
Has (seats)

Movie

Customer
Reserves (seats)
Is given (row number, seat number)
Requests (seat booking)

Time

Date

Seat booking

Show
Is scheduled (in theatre)

Seat

Seat number

Telephone number

Row

Row number

Användning av CRC-kort

- Beskrevs först av Kent Beck och Ward Cunningham.
- Varje CRC-kort innehåller:
  - Ett klassnamn.
  - Klasssens ansvar (responsibilities).
  - Klassens medarbetare (collaborators).

Ett CRC-kort

Class name	Collaborators
Responsibilities	

Scenarier

- Aktiviteter som systemet skall utföra eller ge stöd för.
  - Kallas också användningsfall - use cases.
- Används för att identifiera interaktioner mellan objekt (collaborations).

Ett exempel

CinemaBookingSystem	Collaborators
Can find shows by title and day.	Show
Stores collection of shows.	Collection
Retrieves and displays show details.	
...	

Scenarier som utgångspunkt vid analys

- Kan användas för att analysera om en problembeskrivning är tydlig och fullständig.
- För större projekt måste analysfasen få ta tillräckligt med tid.
- Analysfasen övergår sedan i designfasen.
  - Felaktigheter som upptäcks under designfasen innebär ofta att delar av analysen måste göras om. Felaktigheter som inte upptäcks kan bli mycket kostsamma i ett senare skede ...

Utformning av klasser

- Scenarieanalysen hjälper till för att klargöra hur tillämpningen skall struktureras.
  - Varje CRC-kort blir en klass.
  - Collaborations anger hur objekt av klassen samarbetar med andra objekt.
- Responsibilities anger vilka metoder som skall göras public och ev. vilka strukturer som skall klassen ansvarar för.

Utformning av klassernas gränssnitt

- Gör en genomgång av scenariet med hjälp av metodanrop, parametrar och returvärden.
- Skriv ner signaturerna för de metoder som behövs.
- Gör skelettklasser där de metoder som skall bli public kodas in som stubs.
- Ju mer tid man lägger ner på designfasen desto säkrare och snabbare implementationsfas.

Dokumentation

- Skriv klasskommentarerna.
- Skriv metodkommentarerna.
- Beskriv i första hand avsikten.
- Dokumenteringen i den här fasen syftar främst till att:
  - man fokuserar på vad snarare än hur.
  - man inte glömmer något viktigt!

Samarbete

- Arbete i grupp blir nödvändigt när det gäller större projekt.
- Dokumentationen blir då än mer viktig än om man arbetar ensam med ett projekt.
- Enkel O-O design, med löst kopplade komponenter underlättar samarbetet.

Prototyper

- En prototyp ger kunskap om ett system innan det egentliga arbetet börjar.
  - Framförallt kan de viktiga problemen identifieras.
- Delar av systemet kan simuleras.
  - Metodanrop ger alltid ett fixt resultat.
  - Omgivningen och slumpen kan kontrolleras.

Vattenfallsmodellen

- Analys
- Design
- Implementation
- Uttestning av enheter
- Integration - test av hela programmet
- Leverans

Iterativ utveckling

- Gör prototypen tidigt.
- Diskutera (ofta) med beställaren.
- Iterera över sekvensen:
  - Analys
  - Design
  - Prototypändring
  - Beställarens synpunkter
- Underlätta ändringar/utvidgningar redan från början.

Användning av design patterns

- När man arbetat med ett antal problemställningar upptäcker man att det finns relationsstrukturer mellan klasser som dyker upp gång på gång.
- Design patterns hjälper till att klargöra vilka relationsstrukturer det finns och när de skall användas.

Pattern structure

- Ett namn.
- Problemet som strukturen skall appliceras på.
- Problemet beskrivet i termer av:
  - Strukturer, deltagare, samarbete.
- Konsekvenser.
  - Resultat, fördelar.

Decorator

- Utvidgar funktionaliteten hos ett objekt.
- En decorator-objekt fungerar som ett omslag (wrapper) till ett annat objekt.
  - Har samma gränssnitt som ursprungsobjektet.
  - Metodanrop fångas upp men går sedan till ursprungsobjektet ...
  - ... men the Decorator tillför ytterligare möjligheter.
- Exampel: `java.io.BufferedReader`
  - Förser ett enkelt `Reader`-objekt med en buffert.

Singleton

- Ser till att det bara finns en instans av ett objekt.
  - Alla klienter skall använda samma objekt.
- Konstruktorn är private för att förhindra instansiering utifrån!

```
class Parser
{
        private static Parser instance = new Parser();

        public static Parser getInstance()
        {
                return instance;
        }

        private Parser()
        {
                ...
        }

}
```

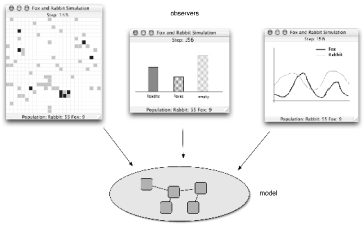
Factory-metoder

- En klient behöver ett objekt av en speciell typ.
- En factory-metod kan returnera ett objekt av en klass som den själv implementerar eller ett objekt av en subklass.
- Typen på objektet som returneras kan bero på sammanhanget.
- Exempel: `iterator`-metoden i Collection-klasserna.

Observer

- Ger en view av en intern beskrivning av en modell.
- Innebär att objektet som “observeras” måste signalera en tillståndsförändring till alla Observers.
- Exempel `SimulatorView`i foxes-and-rabbits-projektet.

Observers



Sammanfattning

- Lär tekniker som underlättar samarbete med andra.
- Gör flexibla, utvidgningsbara klasstrukturer.
  - Var medveten om att färdiga design patterns ger ett bra stöd för detta.
- Samla erfarenheter från dina egna och andras lyckade (och misslyckade) konstruktioner.