Objektorientering

Klasser

Företag

- Andersson, Pettersson och Lundström skall starta företaget APL-karuseller.
- De anställer 5 arbetare
- De lägger företagets pengar i en skokartong så arbetarna kan köpa in material, maskiner och ta ut sin lön.
- Sedan säger de åt arbetarna att börja tillverka karuseller. Intäkter för sålda karuseller läggs i skokartongen.
- Är det något fel med detta?

Organisation / Struktur / Ansvar !

- Hur skulle ett företag utan avdelningar och ansvarsområden fungera?
- Hur skulle det vara att jobba där?
- Alla företag är indelade i olika avdelningar:
  - personal
  - ekonomi
  - inköp
  - försäljning
  - support
  - produktion
- Tillsammans tillhandahåller avdelningarna något till slutkunden.

Objektorientering

- Organisationsidén är överförd till programvara med målen:
  - överskådlighet: att programmet skall vara lätt att överblicka och logiskt att hitta i
  - ändringsbarhet: det skall var lätta att ändra en del i programmet utan att påverka andra delar
  - felhantering: fel som uppstår skall vara lätt att lokalisera till rätt programdel och inte påverka resten av systemet
  - underhåll: ovanstående gör ett programvarusystem mycket enklare att underhålla
- Olika delar i programvaran får olika ansvarsområden
- Ett sätt att strukturera programvara
  - först skapas strukturen, programmets organisation, ansvarsfördelning
  - sedan försöker man uttrycka detta i programkod i något språk
- Objektorienterng är ett sätt att tänka på programvaran!

En avdelning i ett företag

- Utför EN funktion i företaget, har ETT ansvarsområde (helst)
- Tillhandahåller olika tjänster som andra avdelningar kan utnyttja
- Avdelningen har kontroll på sina resurser
- Andra avdelningar har inte tillträde till resurserna, de måste be om lov

En klass

- Har ETT dokumenterat ansvarsområde
  - Indikeras med klassnamn och dokumentation
- Tillhandahåller olika tjänster som andra programdelar kan utnyttja
  - medlemsfunktioner, metoder
- Klassen har sina egna egenskaper, resurser
  - medlemsvariabler
- Inga andra programdelar tillåts tillträde till klassens resurser, de måste ha lov
  - public, private, friend

Objektorienterad analys (OOA)

- Processen att strukturera ett problem i ansvarsområden, olika klasser
- Första steget i objektorienterad programmering (OOP)

1. Analysera (skriv?) specifikation, krav och användningsscenarion
2. Identifiera objektkandidater, ansvarsområden
3. Identifiera tjänster eller operationer
4. Organisera och strukturera, gallra
5. Testa, utför scenarion
6. Börja om och förbättra

CRC

- Class
- Responsibility
- Collaborators

- Resultat från OOA!

Objektorienterad design (OOD)

- Andra steget i OOP
- Utgår från resultatet av OOA
- Detaljspecificerar klasser och medlemsfunktioner, parametrar, returvärden

Programmera

- Överför OOD till programkod
- Programming in the language versus programming into the language
  - Anpassa inte dina tankar till språket.
  - Utryck dina tankar med hjälp språket!
  - Koden skall vara en spegling av hur du tänkt.
- OOA skall ge en klar bild av dina tankar *innan* du börjar låsa in dig på hur det löses i ett visst språk

Syntax för klasser

```
class NAMN
{
public:
  MEDLEMSFUNKTIONER;
private:
  MEDLEMSVARIABLER;
};
```

Klassexempel: beskrivning

```
// this is in "card.h"
#ifndef _CARD_H_
#define _CARD_H_
// do not use namespaces here
class Card
{
public:
  void print() const;
  int getValue() const;
  string getSuit() const;
private:
  string suit;
  int value;
};
#endif
```

Klassexempel: implementation

```
// This is in ”card.cc”
#include ”card.h”

// All members have access to all private parts

void Card::print() const
{
  cout << value << ” of ” << suit;
}

int Card::getValue() const
{
  return value;
}

string Card::getSuit() const
{
  return suit;
}
```

Klassexempel: användning

```
// This is in ”main.cc”
#include ”card.h”

int main
{
  Card my_card;
  // ERROR - main have no access!
  my_card.suit = ”hearts”;
  my_card.value = ”1”;
}
```

Hur initieras medlemmarna?

```
// Addition in ”card.h”
class Card
{
public:
  // Konstruktor!!
  Card(string const& s, int v);
  // ... resten som förut ...
};
// Addition in ”card.cc”
Card::Card(string const& s, int v) : suit(s), value(v)
{
  // this->suit = s;
  // value = v;
}
```

Hur initieras medlemmarna?

```
// using constructor in ”main.cc”
#include ”card.h”

int main()
{
  Card my_card(”hearts”, 1);
  Card your_card(”spades”, 2);

  Card c; // Error, no deafult constructor
  // Should this be possible to do?
  Card one_to_rule_them_all(”rings”, 9999);

  // punkt-operatorn bör redan vara (lite) bekant
  my_card.print(); // ”1 of hearts”
  your_card.print(); // ”2 of spades”
}
```

Klass, Instans, Objekt

- Klass
  - beskrivning av något, beskrivning av en datatyp
  - ingen variabel, inget minne
- Instans
  - en variabel som överensstämmer med beskrivningen
- Objekt
  - ett objekt är en instans av någon klass

Användare av en klass

- Programmeraren som skapat klassen och underhåller klassen.
  - Vill kunna ändra implementationsmetod senare utan att huvudprogrammet går sönder.
  - Vill inte bekymra sig om hur kod som använder klassen fungerar.
- Programmeraren som använder klassen.
  - Vill kunna använda klassen enkelt, bekvämt och intuitivt.
  - Vill inte kunna använda klassen på fel sätt.
  - Vill veta vad klassen kan användas till, inte hur det åstadkoms.
  - Vill inte behöva uppdatera sin kod varje gång skaparen av klassen fixar en bugg eller lägger till nya funktioner.
- Slutanvändaren som använder programmet.
  - Mest intresseard av att programmet som helhet fungerar korrekt.

Koncept

- Koppling (coupling)
  - Hur många och starka kopplingar det finns mellan olika klasser
  - Skall vara så få och svaga som möjligt
- Samhörighet (cohesion)
  - Hur relevanta medlemmar är för en klass
  - Skall vara så stor samhörighet som möjligt
- Inkapsling (encapsulation)
  - När metoder och data som hör ihop samlas och paketeras under ett tak
  - För det är lättare att hantera som en enhet
  - Ger flexibilitet att ändra interna (privata) detaljer senare
  - T.ex en klass
- Information hiding
  - Information som inte är nödvändig för användaren göms undan eller görs oåtkomlig
  - Åstadkoms t.ex. med inkapsling eller abstraktion
- Abstraktion (abstraction)
  - Döljer detaljerna av HUR något fungerar
  - Sätter ett enkelt namn på något, ett namn som berättar VAD som åstadkoms
  - T.ex. funktionen sinus

Filuppdelning

- Inkluderingsfil (headerfil) KLASSNAMN.h
  - Skydd mot dubbelinkludering (guard)
  - Endast klassdefinition och deklarationer
  - Deklarerar allt användaren kan behöva
  - Inkluderas av alla som vill använda klassen (med ”” och inte <>)
  - Bra plats för dokumenterande kommentarer för hur klassen används, vad den kan användas till
  - Kompileras aldrig (varning för KLASSNAMN.gch!)
- Implementationsfil KLASSNAMN.cc
  - Innehåller implementation av alla klassens medlemsfunktioner
  - Innehåller deklarationer och definitioner av allt användaren inte skall eller behöver känna till, och inte skall använda
  - Inkluderas aldrig
  - Kompileras av den som vill använda klassen (kan förkompileras!)

Ett exempel till: Datum

Skapa en klass för att representera ett datum!

Diskutera, Rita, Förklara.

date.h

```
// guard, protect from double include (watch out for (AVOID) circular include!)
#ifndef _DATE_H_
#define _DATE_H_

// representation of date and convenient functions...
class Date
{
  public:
    Date(int y, int m, int d);
    Date(unsigned long long int sec_since_19700101);
    Date(int y, int w, string dayOfWeek);

    string getDayOfWeek(); // "Monday", "Tuesday", etc
    void   nextDate();
    int    toString(); // 20121108
    int    print();
    int    getYear();  // 2012
    int    getMonth(); // 1-12
    int    getWeek();  // 1-53
    int    getDay();   // 1-31

  private:
    int year;   // other representations possible!
    int month;  // user of class should not know or care
    int day;    // then we can use another representation later on
};

#endif
```

date.cc

```
#include "date.h"

Date::Date(int y, int m, int d) { ... }
Date::Date(unsigned long long sec_since_19700101) { ... }
Date::Date(int y, int w, string dayOfWeek) { ... }

int    Date::toString() { ... }
int    Date::print()    { ... }
void   Date::nextDate() { ... }
int    Date::getYear()  { ... }
int    Date::getMonth() { ... }
int    Date::getDay()   { ... }
string Date::getDayOfWeek() { ... }
int    Date::getWeek()  { ... }

// All kod utelämnad, fundera själv på den
```

Mer exempel

Tärning

Spelet 21 (Blackjack)

Klasser - special

- Konstanta medlemsfunktioner
- Statiska medlemsvariabler
- Referensvariabler som medlem
- Initieringslistor
- Vänner (inte Chandler, Joey, Monica, Phoebe, Rachel och Ross...)
- Explicit konstruktor

Mer om klasser (senare Fö)

- De viktiga ”livscykel”-metoderna:
  - Konstruktor(er)
  - Kopieringskonstruktor
  - Tilldelningsoperator
  - Destruktor
- Överlagrade operatorer
- Arv
- Polymorfi

Reservslides

Templates

- En mall för hur något skall utföras
- Vad det utförs på (datatyp) anges först vid användning
- Tills dess används platshållare
- Mallen kan ställa krav på vilka operationer datatypen måste klara av

Template: exempel

```
template<typename PLATSHÅLLARE>
void swap(PLATSHÅLLARE a, PLATSHÅLLARE b)
{
  PLATSHÅLLARE save = a; // kopiering
  a = b;                 // tilldelning
  b = save;
}
// swap() ställer kraven att datatypen
// PLATSHÅLLARE kan kopieras och tilldelas
```

Template: användning

```
int main()
{
  int a, b;
  float c, d;
  double e, f;
  swap(a, b);
  swap(c, d);
  swap(e, f);
  swap(a, f); //?
}
```

Template: kompilering

- Templatekod kan inte kompileras
- Platshållaren är okänd!
- Kan bara kompileras efter att platshållaren bestämts
- Undantag till reglerna om headerfiler och implementationsfiler
- Templateimplementationsfilen kompileras inte, den inkluderas sist i templateheaderfilen