UMEÅ UNIVERSITET
Institutionen för datavetenskap

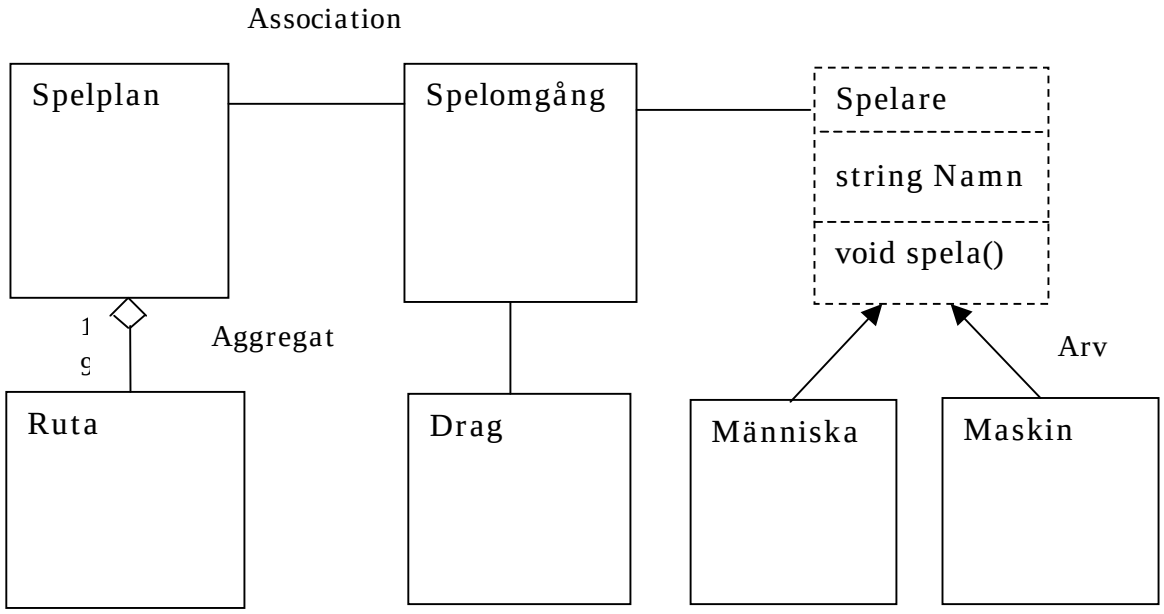
2000-01-11

Kalle Prorok,
Thomas Johansson

PRELIMINÄRT LÖSNINGSFÖRSLAG TILL
Tentamen i
Objektorienterad programmering för ingenjörer
(TDBB09)
2000-01-12, kl 09.00 - 15.00

UPPGIFT 1 – OO (9p)

a) Rita ett klassdiagram för ett luffarschackprogram. Minst 6 klasser, varav en abstrakt, och Arv, Association, Aggregat liksom några attribut och metoder ska ingå. Beskriv din notation! (7p)



b) Vad är dynamisk bindning bra för? (2p)
För att kunna anropa en metod med samma namn men med olika implementation beroende på aktuella objektet i run-time. Detta ger stor flexibilitet

UPPGIFT 2 – Struktur (8p)

a) Beskriv hur undantagshanteringen fungerar i C++ inklusive lite om standardundantagens hierarki och hur man deklarerar vad funktioner kan tänkas sända för något och vad som händer i samband med detta. (4p)
s219:Ett undantag (som är ett objekt) kastas då man anger `throw` och fångas av närmaste `catch` i omgivande `try`-block som anser sig matcha det kastade objektets datatyp(klass). På vägen dit anropas destruktorer för att rensa stacken. Standardundantagen är ett arvsträd med huvudgrenarna std::logic_error och std::runtime_error definierade i <stdexcept>.
s222:Ex på deklaration: int fkn() ***throw(overflow)*** *{}*
Om fkn genererar(ev via anrop till subfkn) något annat undantag genereras/anropas unexpected.

b) Skriv en divisionsrutin *int dividera(int, int)* som testar sina argument och om divisorn (talet under bråkstrecket) är skilt från noll utför heltalsdivisionen. Är divisorn 0 så skall ett undantag genereras. Skriv ett kort huvudprogram som demonstrerar hur funktionen fungerar med undantagshantering. (4p)
Nåt i stil med:

```
#include <iostream>
#include <stdexcept>
int dividera(int t, int n) throw(invalid_argument)
{
  if (n == 0) throw invalid_argument(”Division by zero”);
  return t/n; // do the division
}
void main(void)
{
  int n;
  try
  {
    cout << ”Ge nämnare: ”; cin >> n;
    cout << ”Resultat: ” << dividera(100,n);
  }
  catch (std::logic_error& e)
  {
    std::cerr << ”Fel: ” << e.what() << ”\n”;
  }
}
```

UPPGIFT 3 – Java (4p)

Java skiljer sig på ett antal avgörande punkter från C++, bland annat när det gäller *arv*, *pekare*, *instansiering av objekt* och *portabilitet*. Beskriv vari skillnaden ligger för varje av de ovan uppräknade punkterna.

Arv – Java saknar multpelt arv, i stället finns en sorts 'lättviktsklasser' som kallas interface. Flera av dem kan implementeras ('ärvas') samtidigt.

Pekare – java saknar pekare, i stället finns referenser, som fungerar ungefär som referenser i C++
Instansiering – i C++ kan objekt skapas på flera sätt – med new, som lokala variabler, automatiskt av kompilatorn. I java finns bara new.
Protabilitet – java kompileras till sk. bytecode som kan exekveras på alla maskiner och system som har en sk Java Virtual Machine, en interpretator för bytecode.

UPPGIFT 4 – C++ (6p)

a) Varför har man gjort om castningen (typomvandlingen) i C++ (jmf m C) ? Beskriv skillnaden mellan åtminstone 2 av de 4 sätten. (3p)
För att göra det tydligare i koden samt lite säkrare när så är möjligt. s308:
const_cast<T>(arg) tar bort konstantheten.
dynamic_cast<T>(arg) medger omvandlingar inom en arvshierarki. Returnerar 0 om fel.
reinterpret_cast<T>(arg) Omvandlar t.ex pekare. (noll koll).
static_cast<T>(arg) ber kompilatorn att konvertera

b) Beskriv några (minst 3) av de olika streams-klasserna. (3p)
s341. C++ in och utmatningsklasser.
ostream utmatning t.ex cout
istream inmatning t.ex cin
ifstream, ofstream, fstream används för filhantering
strstream för att styra om till läsning/skrivning i minnet (via en sträng)
De har ofta definierat <<, >> och manipulatorer

UPPGIFT 5 – S T L (5p)

Beskriv hur algoritmbibliotekets for_each() fungerar och ge ett litet programexempel.

Anropar en funktion för varje element i en datastruktur med elementet som parameter.

```
#include <iostream>
#include <algorithm>

void show(char *p)
{
  std::cout << p;
}

main(int argc char *argv[])
{
  for_each(argv+1,argv+argc,show);
}
```

UPPGIFT 6 – Templates (8p)

Inga Gör skulle vilja ha ett klassbibliotek utvecklat åt sig för beräkningar där talen inte är exakta. Hon har tänkt använda det för volymsberäkningar där mätningarna har ett mätfel. Implementera en typparametriserad klass `Matt` som löser hennes problem. Vid multiplikation adderas de relativa felen och relativt fel räknas ut som absolut fel delat med mätvärdet.

Exempel:
350±10 (”350 plus minus 10”)
Det absoluta felet är 10, relativa felet 10/350 = 0.02857.
5±1, absolut fel 1, relativt fel 1/5= 0.2.

Om Inga skall multiplicera dessa mätvärden med varandra måste hon beräkna produkten av talen och summan av de relativa felen:

350±10 * 5±1 = 1750, relativt fel 0.02857 + 0.2

Sedan fås det absoluta felet genom att multiplicera med värdet:

1750 * (0.02857 + 0.2)) = 400,

dvs svaret är 1750±400.

Följande körexempel ska fungera:

Inga har mätt höjden på en låda till 350±10 mm, djupet är exakt 40 mm och bredden 5±1 mm. Volymen beräknas då (2 gånger) till 70000±16000 kubikmillimeter med hjälp av programmet nedan:

```
#include <matt.h>
int main()
{
  Matt<long>    hojd(350,10), djup(40), bredd(5,1);
  Matt<double> dhojd(350,10),ddjup(40),dbredd(5,1);
  std::cout <<  "Volym = " <<  hojd *  djup *  bredd;
  std::cout << " Volym = " << dhojd * ddjup * dbredd;
  std::cin.get(); // Keep screen open
  return 0;
}
// Output: Volym = 70000+-16000 Volym = 70000+-16000
```

Not. Om du inte vet hur man implementerar med templates kan du göra en enklare lösning men få färre poäng.

Nåt i stil med:

```
#include <iostream>
#include <limits>
template <class T>
class Matt
```

```
{
  T _m,_dev;
public:
  Matt(T m,T dev=0):_m(m),_dev(dev) {}
  Matt<T> operator*(Matt m2);
  void print(std::ostream& os) { os << _m << "+-"<<_dev;}
};

template <class T> Matt<T> Matt<T>::operator*(Matt m2)
{
  Matt<T> r = _m * m2._m;
  double relfel = 0.0;
//avoid / 0 and calculate with floating point division
  if (_m != 0) relfel += static_cast<double>(_dev) / _m;
  if(m2._m != 0) relfel += static_cast<double>(m2._dev)/m2._m;
 // compensate integer round-off "Överkurs"
  std::numeric_limits<T> t; // Kolla om T är en heltalstyp
  if (t.is_integer) r._dev = r._m * relfel + 0.5;
  else r._dev = r._m * relfel;
  return r;
}

template<class T> std::ostream& operator<<(std::ostream&os,
Matt<T> m)
{
 m.print(os);
 return os;
}
```

< SLUT PÅ UPPGIFTER >