Linköpings Universitet
Institutionen för datavetenskap (IDA)
UPP-gruppen 2012-10-24

Objektorientering i liten skala

Mål

I denna lab skall du skriva ett objektorienterat program. Programmet skall delas upp i klasser (en objektbeskrivning). Varje klass skall ha ett (och endast ett) ansvar. Klassen representerar något från verkligheten och vad man kan göra med det (ha det till). En klass kan också representera något abstrakt (t.ex. ett komplext tal). Ett exempel på ett påtagligt objekt är en kortlek. En klass som representerar en kortlek har ett ansvar - att hålla reda på korten i kortleken (representeras antagligen av någon slags lista med kort i klassen). Det man kan göra med en kortlek är att blanda den, och dra det översta kortet. Kanske kan man göra flera saker. Det är dock viktigt att dessa saker är hårt knutna till klassens ansvar. Att "beräkna poäng på pokerhand" är ett exempel på uppgift som inte har med kortleken att göra. Den uppgiften passar bättre till klassen "pokerhand", "pokerregler" eller liknande. Saker man kan göra med en klass löses typiskt med medlemsfunktioner.

När du är klar skall du kunna skapa en liten klass med en konstruktor, några medlemsfunktioner och några medlemsvariabler, samt kunna använda klassen från ett huvudprogram. För att få en bra struktur bör du använda en källkodsfil och en headerfil per klass, samt en källkodsfil för ditt huvudprogram.

Läsanvisningar

- Tommy Olsson - objektorientering i ett nötskal.
  - http://www.ida.liu.se/~tao/pub/prog/OOP-Nutshell.pdf
  - http://www.ida.liu.se/~tao/pub/lang/cpp/Checklista-Klassdesign.pdf
- Klasser (class)
  - Medlemsvariabler (member variable)
  - Åtkomstskydd (public, private)
  - Medlemsfunktioner (member function, method)
  - Konstruktorer (constructor)
- Headerfilen för den givna klassen *Console*
- Slumpgenerering C (#include <cstdlib> // srand, rand; #include <ctime> // time)
- Slumpgenerering C++ (#include <random> // std::random_device)
- Viktiga begrepp du kan läsa om i t.ex. Code complete:
  - Abstraction, Information hiding, Encapsulation
  - Class responsibility, Coupling, Cohesion
  - Skillnad: Class, object, instance

Förberedande övning

För att bekanta dig med klasser är det bra med lite förberedande övning. En lämplig övning är att göra om din lösning på lab 2, men nu skapa en klass för att representera ett komplext tal och vad man kan göra med det. En konstruktor och operationerna (medlemsfunktionerna) plus och minus kanske räcker för att du skall komma in i rätt tanke- och skrivsätt.

Uppgift 1 - Pong

Spelet Pong från 1972 var ett av de första riktigt framgångsrika datorspelen. Spelet går i ursprungsversion ut på att två spelare med var sin "paddle" styr en boll över en streckad linje som representerar ett nät. Du skall skapa en enklare variant för en spelare. Spelet i din version består av fyra väggar som formar ett rum, en boll och ett slagträ (eng. "paddle"). Slagträt representeras av en linje framför nedersta väggen i rummet. Bollen placeras någonstans i rummet och rör sig med någon riktning och hastighet. Om bollen når en vägg eller slår emot slagträt så studsar den bort igen precis så som vi normalt förväntar oss. Spelaren skall kunna flytta slagträt i sidled åt vänster genom att trycka 'a', och i sidled åt höger genom att trycka 'd'. Spelaren skall när som helst kunna avsluta spelet genom att trycka 'q'. Om bollen når den nedre väggen är bollen ur spel och spelaren har förlorat. Målet är att hålla bollen i spel så länge som möjligt.

Börja med att göra en objektorienterad analys och design. Implementera sedan ditt spel steg för steg. Det finns inga krav på att spelet skall vara vackert, häftigt eller roligt att spela. Kraven ligger på att koden skall vara välstrukturerad med väl separerade ansvarsområden (minst två egna klasser), samt lätt att läsa, förstå och ändra. I programmeringstermer skall du försöka uppnå "high cohesion", "low coupling" och "information hiding" med din klassdesign (du når kanske inte hela vägen).

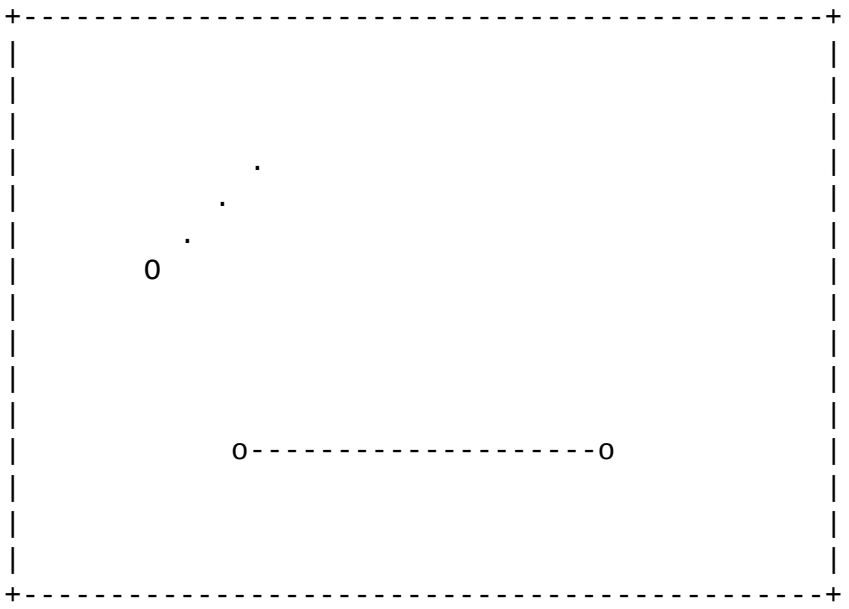
Här kommer en lista med några steg på vägen. Tänk noga igenom vilken klass (eller huvudprogrammet) som skall ha ansvar för vad:

1. Ett grundprogram som ritar ut väggar och bollen på förutbestämd plats.
2. Lägg till att bollen rör sig i med förutbestämd fart och riktning.
3. Lägg till att bollen studsar mot väggar.
4. Lägg till "slagträt" på förutbestämd position.
5. Lägg till att slagträt kan flyttas i sidled.
6. Lägg till att bollen studsar även mot slagträt.
7. Vid start: Slumpa ut bollen inom ett fördefinerat område.
8. Vid start: Slumpa ut fart och riktning inom fördefinerade intervall.
9. Se till att programmet avslutar när det ska.

Du får så klart göra spelet häftigare än du behöver. Här kommer även en lista på möjliga utökningar:

1. Frivilligt: Ge väggarna olika studsegenskaper.
2. Frivilligt: Gör olika studs beroende på var bollen träffar slagträt och hur slagträt rör sig.
3. Frivilligt: Lägg till en andra spelare.
4. Frivilligt: Ersätt *Console*-klassen med en klass som nyttjar riktig grafik, t.ex. libSDL.
5. Frivilligt: Låt fantasin flöda...

Figuren visar hur spelet skulle kunna se ut. Punkerna är tillagda i efterhand för att visa på hur bollen rört sig (punkterna är alltså inget du behöver programmera).



Filseparering

Headerfiler har filändelsen ".h" och inkluderas från de källkodsfiler som behöver funktionaliteten. Headerfiler innehåller bara deklarationer som det är meningen att andra programmerare skall kunna använda. Allt annat göms (om möjligt) i motsvarande källkodsfil. Källkodsfilen innehåller definitioner av alla (medlems)funktioner och hjälpfunktioner.

Headerfiler brukar ha en "guard" för att skydda mot dubbelinkludering. Denna nyttjar preprocessor-instruktioner. Du kan titta i de givna filerna för klassen *Console* hur det är gjort. En enkel "guard" brukar se ut som följer:

```
#ifndef MY_HEADER_H
#define MY_HEADER_H
// Alla deklarationer i filen my_header.h ...
#endif // MY_HEADER_H
```

När du kompilerar ett program med flera filer måste du lista alla källkodsfiler som ingår på kommandoraden (men aldrig headerfiler!). I detta fall behöver du även programbiblioteket "ncurses" som inte är default (*Console*-klassen använder detta bibliotek). Kompileringskommandot kan till slut se ut så här (lägg till övriga bra flaggor själv):

```
g++ pong.cc klassnamn1.cc klassnamn2.cc -lncurses
```

Klassen *Console*

Den givna klassen *Console* har ansvar för programmets in- och utdata (skärm/tangentbord). Deklarationer av de funktioner du kan använda hittar du i klassens "public"-del i headerfilen. Klassen ger möjlighet att placera tecken var som helst i terminalen. Övre vänstra hörnet har koordinaten $(0, 0)$ och nedre högra hörnet har koordinaten $(getWidth() - 1, getHeight() - 1)$. Var nästa tecken placeras avgörs genom anrop till $setPos()$. Ett tecken skrivs sedan med $put()$. För att läsa tecken från tangentbordet används funktionen $get()$. Funktionen returner sant om ett tecken lästes, och själva tecknet ges tillbaka via referensparameter (utparameter). Om $get()$ returnerar falskt betyder det att ingen tangent trycktes ned inom cirka 1/10 sekund. Detta tillåter en mycket rustik huvudloop i "spelet":

```
Console console;
char command;
bool game_over = false;

while ( ! game_over )
{
  if ( console.get(command) )
  {
    // process user input
  }
  else
  {
    // move ball
  }
}
```

Timingen kommer inte vara perfekt med detta och det finns risk att bollen antingen stannar eller flyttar sig fortare medan användaren trycker på knappar. Detta är inget vi bryr oss om men du får

naturligtvis försöka få till bra timing.