Föreläsning 8:
Att skriva egna klasser.

Christian.Soderberg@cs.lth.se

16 februari 2013

Idag

- Att skriva egna klasser (som `Turtle`, `Die`, etc).
- Detta avsnitt är helt centralt i kursen, och behandlas under flera föreläsningar.
- Under övningen kan ni bland annat skriva en egen `Turtle`-klass.

Att skriva egna klasser

- Alla program är egna klasser, så vi har redan skrivit många klasser.
- Det viktigaste nya idag är att vi kommer att titta på våra klasser 'utifrån' – dvs vi kommer att skapa objekt av dem, och anropa metoder på dem *från andra objekt*.
- Det mesta och viktigaste av det vi skall göra idag kan ni egentligen redan, ni har bara betraktat det från ett annat perspektiv (förmodligen).

Hur ser en klass ut?

- Vi har sett att en klass kan innehålla:
  - *Globala variabler*: dessa variabler kan vi använda i alla underprogram, och de finns kvar så länge objektet finns kvar.
  - *Underprogram*: detta är programkod som bland annat kan ändra värdet på våra globala variabler.
- Vi skall idag lära oss att använda 'globala variabler' och 'underprogram' för att beskriva tillståndet hos objekt av olika slag.

Programklasser och vanliga klasser

- Det är bara 'program-klasser' som behöver innehålla

```
public static void main(String[] args)
```

  och en run-procedur. Det är dessa klasser vi kan köra som program från kommandofönstret.
- I de program som vi skrivit hittills har vi aldrig lämnat huvudprogram-klassen.
- De flesta klasser är inte program-klasser, utan beskriver objekt som vi kan använda i våra program (som `Turtle`, `Die`, `String`, `GraphicsWindow`, ...).

Exempel – klassen `Turtle`

- Hur vet ett `Turtle`-objekt var det befinner sig?
- Hur vet det vilken riktning det har?
- Hur vet det om pennan är uppe eller nere?
- Hur påverkar pennans läge sköldpaddan?

Exempel på klass

- Om vi definierar en klass `Turtle` med bland annat följande globala variabler:

```
class Turtle {
    double x, y;    // position
    int dir;        // riktning
    ...
}
```

  så kommer alla `Turtle`-objekt vi skapar att få var sin uppsättning av dessa variabler.
- Vi kallar variablerna *attribut*, eller *tillståndsvariabler*, eftersom de beskriver tillståndet hos våra `Turtle`-objekt.

Att anropa metoder på ett objekt

- När vi anropar en metod på ett objekt, så körs ett 'underprogram' inne i objektet.
- Om vi skriver `t.forward(10)` så körs underprogrammet `forward` inne i det `Turtle`-objekt som `t` refererar till, vi kallar `forward` en *metod*, eller *operation* (det är samma sak).
- En metod kan ändra tillståndet i sköldpaddan genom att ändra dess tillståndsvariabler (exempelvis `x` och `y`).

Angående klassen `Turtle`

- Under övningen är det tänkt att ni skall implementera en *osynlig* sköldpadda.
- Metoderna `penUp`, `penDown`, `left` och `right` skall bara se till att sköldpaddan uppdaterar sitt tillstånd – det händer ingenting på skärmen när vi anropar dem (om vi har osynliga sköldpaddor).
- Det är först när vi kör `forward` som vi utnyttjar informationen om pennans läge och sköldpaddans riktning.

Egna klasser, terminologi

- Varje klass beskriver ett slags objekt.
- Inuti ett objekt lagrar vi objektets tillstånd i variabler – vi har tidigare kallat dem *globala variabler*, vi kommer nu istället att kalla dem *attribut* och/eller *tillståndsvariabler*.
- Vi kommer att kalla underprogrammen för *metoder* och/eller *operationer*, och vi kan alltså anropa dem 'utifrån'.

Att skapa objekt

När vi skapar ett nytt objekt händer följande med våra attribut:

- De som uttryckligen ges ett värde vid deklarationen får detta värde.
- De som inte ges ett värde nollställs automatiskt.
- *Konstruktorn* kan därefter ge attributen lämpliga värden, så att objektet får ett väldefinierat starttillstånd.

Att skydda tillståndet

- Ofta vill vi dölja/skydda tillståndet hos våra objekt, och `private`-deklarerar därför våra attribut.
- De metoder som vi vill att användaren skall kunna anropa `public`-deklarerar vi.