Övningar
Dag 2 – En första klass

Denna övning går ut på att steg för steg bygga upp en klass och skapa objekt. Vi kommer att utgå från en sammansatt datatyp i en `struct` och parallellt bygga en klass. Vi kan därmed se likheter och skillnader. Det intressanta blir sen att se vilka fördelar vi kan se i detta sätt att bygga datatyper.

Enkelt kan vi se en klass som en sammansatt datatyp där vi har kopplat inte bara variabler samman utan även funktioner.

Övning 1
Skapa ett projekt som heter FirstClass. Klipp in koden som finns i `triangles.cpp`. Denna fil innehåller en struct som representerar en triangel med en bas och en höjd. Separata funktioner används för att skapa en triangel och för att beräkna dess area. Provkör programmet och studera koden så att du förstår vad den gör.

Övning 2
Ovanför main-funktionen, skriv in följande stubbe.

```
class Triangle
{
public:
    double base;
    double height;
};
```

Detta är grunden till en klass. I `main`-funktionen innan return skriv in följande.

```
cout << "Tringle as a class" << endl;
Triangle tobj;
tobj.base = 20.0;
tobj.height = 5.0;
cout << "Arean är " << tobj.base * tobj.height / 2.0 << endl;
```

Kompilera och se till att exemplet fungerar. Notera likheten med `struct`.

Begrepp: När vi *deklarerar* variablen tobj *allokeras* (reserveras) samtidigt minne för de variabler som finns i klassen. Detta kallas att ett *objekt instanseras*. Vi har skapat ett objekt av klassen `Triangle`. Detta objekt lagras i variabeln `tobj`.

Övning 3

Vi går direkt vidare och kopplar en funktion till vår klass. Sist i klassen lägg till följande.

```
double getArea()
{
    return base * height / 2.0;
}
```

I `main`-funktionen, ändra din utskrift till

```
cout << "Arean är " << tobj.getArea() << endl;
```

Kompilera och se att exemplet fungerar.

Vad har vi just gjort? Först kan vi notera att funktionen `getArea()` kan läsa `base` och `height` som ligger i klassen utan att vi behöver ha dessa eller variabeln tobj som parameter. Vi kan också se att vi kan anropa funktionen `getArea()` direkt på variabeln `tobj` precis som vi tidigare gjorde med variablerna `base` och `height`. Vi har knutit en funktion till objekt av klassen `Triagle`.

Begrepp: En funktion som är knuten till ett objekt, kallas för *metod*. Variablerna i objektet kallas *attribut*. De metoder som deklareras i klassen kan anropas på objekt.

Övning 4

När objektet skapas, t.ex. när vi skapar en variabel av klassen, så vill vi nästan alltid att variablerna i objektet också får startvärden. Detta är en nackdel med struct. Om vi glömmer att sätta startvärden på variablerna så kan svårhittade fel uppstå. Detta har man åtgärdat i och med införandet av objekt.

När ett objekt skapas, så anropas alltid en speciell metod (funktion knuten till objektet) som kallas *konstruktor*. Konstruktorn kan även ha parametrar. Det gör att vi i denna metod kan sätta startvärden på våra attribut (variablerna i objektet). Vi ska skapa en konstruktor.

Före `getArea()` skriv in följande.

```
Triangle()
{
    base = 10.0;
    height = 5.0;
}
```

I main-funktionen, ta bort raderna där du sätter värden på `tobj.base` och `tobj.height`. Kompilera och se att du fortfarande får en area som är 25,0.

Några saker kan vi alltså notera om **konstruktorn**.

1) Konstruktorn har samma namn som klassen.

2) Konstruktorn anropas automatiskt när objektet skapas (när vi deklarerar variabeln `tobj`).

3) Konstruktorn returnerar inget (inte ens `void`).

Övning 5

Alla trianglar bör knappast vara 10x5 stora. Vi vill kunna styra detta när triangeln skapas. Det innebär också att konstruktorn behöver parametrar. Detta går utmärkt. Lägg till parametrarna `b` och `h` i parameterlistan till konstruktorn.

```
Triangle(double b, double h)
{
    base = b;
    height = h;
}
```

Det här gör också att konstruktorn måste få argument när den anropas, alltså när objektet skapas. När skapas det? Jo, när vi skapar variabeln `tobj`. Alltså ändra den raden till följande.

```
Triangle tobj(10.0, 5.0);
```

Testa att ändra argumenten till konstruktorn och notera att arean ändras.

Konstruktorer kan ha parametrar. Argument måste då anges när objektet skapas.

Övning 6

En stor nackdel med `struct` är att variablerna i den kan ändras av vem som helst till vad som helst. En fördel med klasser är att vi kan styra åtkomsten, alltså vilka funktionen som kan ändra våra variabler i datatypen. Ordet `public` i klassen vi har skrivit indikerar att de attribut (variabler) och metoder (funktioner) som finns listade efteråt är *publika*. Det innebär att alla funktioner i programmet kan komma åt och använda dessa.

I klassen, ändra `public` till `private` och testa att kompilera. Visst får du flera fel? Kompilatorn kommer bl.a. att hävda att konstruktorn `Trinagle()` är privat och därför inte kan anropas.

Låt `private` stå kvar, men före `Triangle()` lägg till `public:`, så att det ser ut så här:

```
class Triangle
{
private:
    double base;
    double height;
public:
    Triangle(double b, double h)
    {
        base = b;
        height = h;
    }
    double getArea()
    {
        return base * height / 2.0;
    }
};
```

Se till att exemplet kompilerar nu.

Voilà! Vi har en klass och skapar objekt!!

Vi reglerar åtkomst med **`public`** och **`private`**. Privata attribut och metoder kan enbart kommas åt av metoder på det egna objektet eller objekt av samma klass.

Övning 7

Skapa en metod, **`void`** `setBase(`**`double`** `newb)`, som gör att vi kan ändra värdet attributet `base`. Gör samma sak för `height`. Du kan även skapa funktioner för att få deras värden. Hur?

Övning 9

Skapa ett nytt projekt som heter Calendar. Kopiera koden från `timestamp.cpp` och skriv om koden så att det blir en klass som heter `Timestamp`. Klassen bör ha en konstruktor som sätter timmar och minuter, samt metoderna `void tick()` och `void step(int h, int m)`, `int getMinutes()` och `int getHours()`. Använd gärna `step()` både i konstruktorn och i tick.

Övning 10

Utgå från filen `car.cpp` och skapa ett projekt. Skriv sedan om koden så att du använder en klass istället.

Utöka car-klassen så att den lagrar antalet hästkrafter också, samt hur många hjul som drivs (dvs tvåhjulsdrift kontra fyrhjulsdrift). Lägg till lämpliga metoder för att ändra och hämta deras värden.

Ändra deklarationen av klassen så att det inte finns något attribut `speed`, lägg istället till två nya attribut `time` och `distance`, som representerar mätvärden som kan användas för att beräkna speed. Vilka funktioners innehåll måste ändras för att det skall fungera ? Använd gärna en privat hjälpfunktion här. Måste något ändras i main-programmet ?