# Laboration 3 – blockmullvad

*Mål:* Du ska lära dig mer om att strukturera kod genom att skriva och använda egna klasser. Du får öva på att skriva metoder och deklarera attribut. Du får även använda statiska variabler.

## Förberedelseuppgifter

- Påbörja gärna datorarbetet nedan.

## Bakgrund

**Blockmullvad** (*Talpa laterculus*) är ett fantasidjur i familjen mullvadsdjur. Den är känd för sitt karaktäristiska kvadratiska utseende. Den lever mest ensam i sina underjordiska gångar som till skillnad från mullvadens (*Talpa europaea*) har helt raka väggar.

## Datorarbete

1. Du ska själv bygga upp ditt Java-program steg för steg i Eclipse men projektet finns förberett i det workspace som använts på tidigare laborationer.

   a) Skapa filen `Mole.java` genom att:

      a) Högerklicka på projektet *Lab03* i projektvyn,

      b) Expandera New i menyn och välj Class.

      c) Skriv namnet på klassen (`Mole`).

      d) Klicka på Finish.

   b) Öppna din nya klass (om det inte redan gjorts automatiskt). Lägg till en `main`-metod i klassen som skriver ut texten *Keep on digging!* med hjälp av `System.out.println`.

   c) Testa att köra ditt program. Om allt går bra ska texten du angivit skrivas ut i konsolfönstret i Eclipse.

   d) Nu har du skrivit ett Java-program som skriver ut en uppmaning till en mullvad att fortsätta gräva. Det programmet är inte så användbart, eftersom mullvadar inte kan inte läsa. Nästa steg är att skriva ett grafiskt program, snarare än ett textbaserat.

      Funktionen `println` som anropas i `main`-funktionen ingår i Javas standardbibliotek. Ett programbibliotek innehåller kod som kan användas av andra program, och för de flesta programspråk ingår ett standardbibliotek som alla program kan nyttja. Till grafiken i denna uppgift ska du, precis som i tidigare laborationer, använda den färdiga klassen `SimpleWindow`. Den finns i biblioteket *cs_pt* i ditt workspace.

   e) Låt `main`-metoden innehålla följande satser:

```
SimpleWindow w = new SimpleWindow(300, 500, "Digging");
w.moveTo(10, 10);
w.lineTo(10, 20);
w.lineTo(20, 20);
w.lineTo(20, 10);
w.lineTo(10, 10);
```

      Överst i filen måste du ha raden

```
import se.lth.cs.pt.window.SimpleWindow;
```

för att klassen `SimpleWindow` ska bli tillgänglig.

Koden känner du nog igen från tidigare i kursen. Den första raden skapar ett nytt `SimpleWindow` som ritar upp ett fönster som är 300 bildpunkter brett och 500 bildpunkter högt med titeln *Digging*. `SimpleWindow` har en *penna* som kan flyttas runt och rita linjer. Anropet `w.moveTo(10, 10)` flyttar pennan för fönstret `w` till position $(10, 10)$ utan att rita något, och anropet `w.lineTo(10, 20)` ritar en linje därifrån till position $(10, 20)$.

f) Kör ditt program. Du ska nu få upp ett fönster med en liten kvadrat utritad i övre vänstra hörnet.

2. Hela ditt program är för tillfället samlat i en och samma metod, vilket fungerar bra för väldigt små program. Nu ska vi strukturera om programmet så det blir lättare att utöka senare.

a) Skapa en ny klass med namnet `Graphics` (se punkt a i föregående uppgift om du glömt hur man skapar en klass) och flytta dit deklarationen av fönstret `w` så att det blir ett attribut i klassen. Skapa en ny metod i klassen `Graphics` med namnet `square`, och flytta dit koden som ritar kvadraten.

Filen `Graphics.java` ska se ut såhär:

```
import se.lth.cs.pt.window.SimpleWindow;

public class Graphics {
    private SimpleWindow w = new SimpleWindow(300, 500, "Digging");

    public void square(){
        // Fyll i koden för att rita en kvadrat här.
        // Observera att w är definerat ovan.
    }
}
```

Nu har vi beskrivit vad klassen `Graphics` ska innehålla och hur den ska fungera. Men för att faktiskt använda den måste vi göra något mer. Vi måste anropa metoden `square` i `main`-metoden. Metoden `square` finns i klassen `Graphics`, och därför måste ett `Graphics`-objekt skapas först. Därefter använder vi punktnotation `square`-metoden genom att med punktnotation ange att det är en metod inuti `Graphics`-objektet som anropas.

Filen `Mole.java` ska se ut såhär:

```
public class Mole {

    public static void main(String[] args) {
        Graphics g = new Graphics();
        g.square();
    }

}
```

b) Kör programmet, om allt fungerar ska programmet göra samma sak som i föregående uppgift.

c) Kommentera bort raden `g.square();` genom att sätta `//` framför. Kör programmet igen och se vad som händer. Varför är det så?

d) Ta bort kommentartecknen vid `g.square();` igen och kommentera nu istället bort raden `Graphics g = new Graphics();`. Vad händer då och varför? Diskutera med din labhandledare om det känns oklart. Ta bort kommentaren och kontrollera att programmet fungerar innan du går vidare.

3. Nu ska du skapa ett nytt koordinatsystem för `Graphics` som har *stora* bildpunkter. Vi kallar `Graphics` stora bildpunkter för *block* för att lättare skilja dem från `SimpleWindows` bildpunkter. Om blockstorleken är $b$, så ligger koordinaten $(x, y)$ i `Graphics` på koordinaten $(bx, by)$ i `SimpleWindow`.

a) Lägg till följande deklarationer som attribut överst i klassen `Graphics`.

```
private int width;
private int blockSize;
private int height;
```

b) Nu vill vi att våra attribut ska få startvärden. Ett heltalsattribut är noll om inget annat anges. Om man vill att ett attribut alltid ska få samma startvärde så går det att tilldela värdet direkt vid deklarationen (alltså t.ex. `private int width = 30;`). Men i denna laborationen ska vi öva på att ge attributen dess startvärden via konstruktorn. På så vis kan den som skapar objektet själv välja vilka värden attributen ska få.

En konstruktor har alltid samma namn som klassen. Skapa en konstruktor med heltalsparametrarna `w`, `h` samt `bs`. Låt värdet av `w` tilldelas till attributet `width`, värdet av `h` tilldelas till attributet `height` och värdet av `bs` tilldelas till attributet `blockSize`.

c) Ändra bredden på ditt `SimpleWindow` till `width * blockSize` och ändra höjden till `height * blockSize`. Detta betyder att fönstret måste skapas i konstruktorn, men det måste ändå deklareras som attribut. Nu ska din klass alltså ha följande struktur:

```
import se.lth.cs.pt.window.SimpleWindow;

public class Graphics {
    private int width;
    private int blockSize;
    private int height;

    private SimpleWindow w;

    public Graphics(int w, int h, int bs){

        // Lägg till satser för att initiera width, blocksize och height

        this.w = new SimpleWindow(width * blockSize,
                                  height * blockSize,
                                  "Digging");
    }

    // Metoden square som du skrivit tidigare
}
```

d) Nu har du ändrat i `Graphics`. Bland annat har du ändrat hur objekt av klassen skapas. Detta kräver att du gör motsvarande ändring i klassen `Mole`. Nu måste du skicka in värden på varje parameter (`w`, `h` och `bS`) – dessa värden kallas argument. Ändra därför så att `main` nu ser ut så här i stället:

```
public class Mole {
```

```
    public static void main(String[] args) {
        Graphics g = new Graphics(30,50,10);
        g.square();
    }
}
```

Provkör och kontrollera att programmet fortfarande fungerar som innan.

e) Innan vi gjorde en egen konstruktor kunde vi ändå skapa ett nytt `Graphics`-objekt, men då utan att ange några värden. Varför? (Hitta ledtrådar genom att googla ”Java default constructor”).

f) Skapa en ny metod i `Graphics` med namnet `block` och två parametrar `x` och `y` av typen `int` och returtypen `void`. Metodens *kropp* ska se ut såhär:

```
int left = x * blockSize;
int right = left + blockSize - 1;
int top = y * blockSize;
int bottom = top + blockSize - 1;
for(int row = top; row <= bottom; row ++){
    w.moveTo(left, row);
    w.lineTo(right, row);
}
```

g) Metoden `block` ritar ett antal linjer. Hur många linjer ritas ut? I vilken ordning ritas linjerna?

h) Anropa funktionen `block` några gånger i `main`-metoden så att några olika block ritas upp i fönstret när programmet körs. Kör ditt program och kontrollera resultatet.

4. Det finns många sätt att beskriva färger. I naturligt språk har vi olika namn på färgerna, till exempel *vitt*, *rosa* och *magenta*. I datorn är det vanligt att beskriva färgerna som en blandning av *rött*, *grönt* och *blått* i det så kallade RGB-systemet. `SimpleWindow` använder typen `java.awt.Color` för att beskriva färger och `java.awt.Color` bygger på RGB. Det finns några fördefinierade färger i `java.awt.Color`, till exempel `java.awt.Color.BLACK` för svart och `java.awt.Color.GREEN` för grönt. Andra färger kan skapas genom att ange mängden rött, grönt och blått.

a) Skapa en ny klass med namnet `Colors` och lägg in följande definitioner:

```
public static final Color MOLE = new Color(51, 51, 0);
public static final Color SOIL = new Color(153, 102, 51);
public static final Color TUNNEL = new Color(204, 153, 102);
```

För att använda klassen `java.awt.Color` är det enklast att importera den genom att skriva `import java.awt.Color;` överst i filen.

Den tre parametrarna till `new Color(r, g, b)` anger hur mycket *rött*, *grönt* respektive *blått* som färgen ska innehålla, och mängderna ska vara i intervallet 0–255. Färgen $(153, 102, 51)$ innebär ganska mycket rött, lite mindre grönt och ännu mindre blått och det upplevs som brunt. Klassen `Colors` fungerar här som en färgpalett, men vi har inte ritat något med färg ännu. Kompilera och kör ditt program ändå, för att se så programmet fungerar likadant som sist.

b) Lägg till en parameter till metoden `block` i klassen `Graphics`. Skriv dit den sist i parameterlistan med namnet `color` och typen `java.awt.Color`. För att ändra färgen

på blocket, till den färg som angivits som parameter, ska du byta linjefärg innan du ritar. Lägg till följande rad i början av metoden `block`:

```
w.setLineColor(color)
```

c) Ändra i `main` och lägg till en av färgerna från klassen `Colors` som tredje argument i dina anrop till `block`. Eftersom de färger du har definerat i ditt program är publika (`public`) och statiska (`static`) i klassen Colors behöver vi denna gången inte skapa något objekt. Ett anrop till metoden `block` kan därför se ut så här: `g.block(10,2,Colors.MOLE);` Kompilera och kör ditt program och upplev världen i färg.

*Ytterligare förklaring:* Normalt sett har varje objekt sina egna värden på varje attribut (som i fallet med `width`, `height`, `blockSize` och `w` i klassen `Graphics`) men `static` betyder att det bara finns ett värde i hela programmet (det är alltså inte unikt per objekt). `public` betyder att det går att komma åt värdena även från andra klasser och final betyder att det aldrig kan ändras under programmets gång. Detta betyder att vi har definierat färgerna som tre globala konstanter. Konventionen i Java är att konstanter namnges med enbart versaler. Punktnotation används alltid för att komma åt saker i en annan klass eller ett annat objekt, och därför skriver vi `Colors.MOLE` för att komma åt färgen.

*Tips:* undvik att ange attribut som `public` eller `static` om du inte är riktigt säker på din sak. De flesta attribut framöver i kursen är privata och icke-statiska.

d) Eftersom attributen i klassen `Graphics` är privata kan vi inte komma åt dessa från klassen `Mole`. Men ofta finns det behov av att i efterhand komma åt attributens värden. I nästa uppgift kommer vi behöva komma åt bredden och höjden därför ska vi nu implementera get-metoder för dessa attribut.

En get-metod har ofta samma namn som attributet men med `get` framför. Då en metod består av flera ord låter man första ordet börja med liten bokstav (eftersom metoder alltid ska börja med liten bokstav) och alla efterföljande ord börja med stor bokstav (för att det ska bli lättare att läsa). Vidare brukar varje get-metod enbart returnera attributets värde, det betyder att get-metodernas returtyp ska vara samma som attributets typ. Det leder oss fram till att vi behöver följande i klassen `Graphics`:

```
public int getWidth() {
    return width;
}

public int getHeight() {
    return height;
}
```

Fråga din handledare om du behöver ytterligare förklaring.

5. Nu ska du skriva en funktion för att rita en rektangel och rektangeln ska ritas med hjälp av funktionen `block`. Sen ska du rita upp mullvadens underjordiska värld med hjälp av denna funktion.

a) Lägg till en metod i klassen `Graphics` med namnet `rectangle`. Metoden ska ta fem parametrar: `x`, `y`, `width` och `height` av typen `int`, samt `c` av typen `Color`.

Parametrarna `x` och `y` anger `Graphics`-koordinaten för rektangelns övre vänstra hörn och `width` och `height` anger bredden respektive höjden på rektangeln. Använd följande `for`-satser för att rita ut rektangeln.

```
for (int yy = y; yy < y + height; yy++){
    for(int xx = x; xx < x + width; xx++){
        block(xx, yy, c);
    }
}
```

b) I vilken ordning ritas blocken ut?

c) Skriv en metod i klassen `Mole` med namnet `drawWorld` som ritar ut mullvadens värld, det vill säga en massa jord där den kan gräva sina tunnlar. `drawWorld` ska inte ha några parametrar, returtypen ska vara `void`, och metoden ska anropa `rectangle` för att rita en rektangel med färgen `Colors.SOIL` som precis täcker fönstret. Anropa `drawWorld` i `main`-metoden och testa så att det fungerar genom att köra programmet.

*Ledning:* Eftersom funktionen `rectangle` finns i klassen `Graphics` så måste vi i klassen `Mole` ha tillgång till det `Graphics`-objekt som ska användas. Vi lägger det därför som attribut i klassen Mole. Eftersom attributet och metoden `drawWorld` inte är statiska måste vi då skapa ett objekt av typen `Mole` för att sen kunna använda metoderna. (Exekveringen startar ju i `main`, som är statisk, därför finns inget `Mole`-objekt från början, trots att vi är inuti klassen.) `Mole` måste alltså att se ut så här:

```
public class Mole {
    private Graphics g = new Graphics(30, 50, 10);

    public static void main(String[] args) {
        Mole m = new Mole();
        m.drawWorld();
    }

    public void drawWorld(){
        // Kod som ritar upp världen med hjälp av g.rectangle...
    }
}
```

Eftersom vi inte definerat någon konstruktor i klassen `Mole` är det återigen default-konstruktorn som anropas (vilket är tillräckligt för denna uppgift eftersom inga speciella värden måste tilldelas i konstruktorn).

6. I `SimpleWindow` finns en metod för att känna av tangenttryckningar. Du ska använda denna för att styra en liten blockmullvad.

a) Lägg till följande metod i klassen `Graphics`:

```
public char waitForKey() {
    return w.waitForKey();
}
```

I `Graphics`-klassen finns alltså nu en metod för att invänta en tangenttryckning. Den har vi implementerat genom att anropa metoden med samma namn i vårt `SimpleWindow`-objekt `w`.

Man kan säga att klassen `Graphics` *delegerar* uppgiften till `SimpleWindow`. Ser du varför det är en passande term?

7. Nu är det dags att få mullvaden att gräva på riktigt.

   a) Lägg till en funktion i klassen `Mole` med namnet `dig`, utan parametrar och med returtypen `void`. Funktionens kropp ska se ut såhär (fast utan `/* TODO */`):

   ```
   int x = g.getWidth() / 2;    // För att börja på mitten
   int y = g.getHeight() / 2;
   while (true) {
       g.block(x, y, Colors.MOLE);
       char key = g.waitForKey();

       if (key == 'w') { /* TODO */ }
       else if (key == 'a') { /* TODO */ }
       else if (key == 's') { /* TODO */ }
       else if (key == 'd') { /* TODO */ }
   }
   ```

   Byt ut i alla `/* TODO */` mot Java-satser så att `'w'` styr mullvaden ett steg uppåt, `'a'` ett steg åt vänster, `'s'` ett steg nedåt och `'d'` ett steg åt höger.

   b) Ändra `main` så att den anropar både `drawWorld` och `dig`. Kompilera och kör ditt program för att se om programmet reagerar på knapptryck på w, a, s och d.

   c) Om programmet fungerar kommer det bli många mullvadsfärgade block som tillsammans bildar en lång mask, och det är ju lite underligt. Lägg till ett anrop i `dig` som ritar ut en bit tunnel på position $(x, y)$ efter anropet till `waitForKey` men innan `if`-satserna. Kompilera och kör ditt program för att gräva tunnlar med din blockmullvad.

## Frivilliga extrauppgifter

8. Mullvaden kan för tillfället gräva sig utanför fönstret. Lägg till några `if`-satser i början av `while`-satsen som upptäcker om `x` eller `y` ligger utanför fönstrets kant och flyttar i så fall tillbaka mullvaden precis innanför kanten.

9. Mullvadar är inte så intresserade av livet ovanför jord, men det kan vara trevligt att se hur långt ner mullvaden grävt sig. Lägg till en himmelsfärg och en gräsfärg i objektet `Colors` och rita ut himmel och gräs i `drawWorld`. Justera också det du gjorde i föregående uppgift, så mullvaden håller sig under jord. (*Tips:* Den andra parametern till `Color` reglerar mängden grönt och den tredje parametern reglerar mängden blått.)

10. Ändra så att mullvaden kan springa uppe på gräset också, men se till så att ingen tunnel ritas ut där.

## Checklista

I den här laborationen har du övat på att

- Bygga upp ett program steg för steg och testa ofta
- Strukturera vårt program med klasser och metoder
- Skriva egna klasser och metoder
- Skapa och anropa en egen konstruktor
- Anropa default-konstruktorn

- Definera och använda statiska konstanter
- Skriva get-metoder