Processing

Bok 9

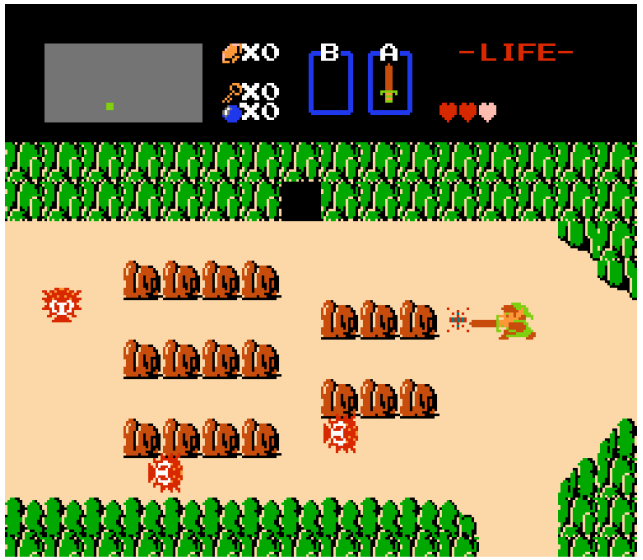

Figure 1: Bok 9

#	Beskriving
33	Funktioner 1
34	Uppräkningar
35	Klasser 1
36	Funktioner 2

Contents

Förord

Detta är en bok om Processing för ungdomar. Processing är ett programmeringsspråk. Denna bok lär dig det programmeringsspråket.

Om den här boken

Det är fortfarande en lite slarvig bok. Det finns stafvel och la*youten är* in`te allt`**id vack**er. Eftersom den här boken finns på en webbplats kan alla som tycker att den här boken är för slarvig göra den mindre slarvig.

```
void draw()
{
  ellipse(x, 100, 100, 100);
  x = x + 1;
  if (x > width + 50)
  {
    x = -50;
  }
  if (x > width + 50) show_error("x is too big");
  if (x < -50) show_error("x is too small");
}
```

Den här stil är kallad 'att programmera defensivt': du, som programmerare, vet att du är felaktig, och du hjälper dig själva med dina fel.

36.5. Slutuppgift

Tar en spel av dig själva och lägg till funktionen `show_error` och användt den på ett nyttigt sät åtminstone fem gånger.

33. Funktioner 1

Funktioner hjälper en programmerare att uttrycka sina tankar bättre: istället av att skriva en massa rad med kod, kan man skriva, t.ex. `move_enemies()`, eller `draw_menu()`.

Nu är det dags att vår kod blir skriven i engelska

33.1. Funktionen måste göra vad dem säger

Kolla på en minimalt Processing program:

```
void setup() {}

void draw() {}
```

Den här program använder två funktioner: `setup` och `draw`. Namnet av `setup` funktionen är bra: den gör vad den säger. Namnet av `draw` funktionen blev från bra till dåligt: i våra första program bara ritade vi saker. Men nudagens reagerar vi också på tangenbordet, kankse spelar musik, kankse skriva en fil.

En bra funktion gör exakt vad den säger

Här har vi ett mer komplett exempel:

```
float x = 0;
float y = 0;

void setup()
{
  size(200,200);
  x = width / 2;
  y = height / 2;
}

void draw()
{
  if (keyPressed)
  {
    if (key == 'w') y = y - 1;
    if (key == 'd') x = x + 1;
    if (key == 's') y = y + 1;
    if (key == 'a') x = x - 1;
  }
  if (x < 0) y = width;
  if (x > width) x = 0;
  if (y < 0) y = height;
  if (y > height) y = 0;
  point(x,y);
}
```

Tror du att `setup` och `draw` funktionerna gör vad den säger? Varför ja eller nej?

36.4. Svar

`show_error` blir:

```
void show_error(final String e)
{
  println("ERROR: " + e);
  exit();
}
```

Du kan ser att `exit` stänger av programmet. I vår fall säger programmet en felmeldning och stänger av programmet. Det ska hjälper oss med felsökning!

36.5. Felsökning

Här har vi ett bekant program, nu med felsökning:

```
float x = 50;

void show_error(final String e)
{
  println("ERROR: " + e);
  exit();
}

void setup()
{
  size(320, 200);
}
```

36.2. Svar

Här är ett möjligt svar:

```
void show_error(final String e)
{
  println("ERROR: " + e);
}

void setup()
{
  show_error("You should never start this program");
}

void draw() { }
```

Här, `show_error` kallar `String`en `e`. Du får kallar den hur som helst. Man kan säga att `e` är för kort, och man föredra `error_message` istället. Också `text` eller `s` (‘en `String`’) funkar bra som namn av funktionsargumentet.

36.3. `exit`

Vär funktion är inte helt nyttigt för felsökningen: den bör stänga av programmet när något fel händer.

Lägg till den här rad inom funktionskroppen av `show_error`:

```
exit();
```

Vad tror du att `exit` gör? Vad händer?

33.1. Svar

`setup`funktionen gör vad den säger: den gör saker för att förbera spelet.

`draw`funktion gör mer än vad den säger:

- först reagerer den på tangentborden
- efter detta, hållar den spelaren i fönstret
- sist ritar den spelaren

33.2. Att lägga till kommentar

Den första steg att förenkla din kod är med kommentarer som sammanfattar vad några rader gör, t.ex:

```
float x = 0;
float y = 0;

void setup()
{
  size(200,200);
  x = width / 2;
  y = height / 2;
}
```

```
void draw()
{
  // Respond to keyboard
  if (keyPressed)
  {
    if (key == 'w') y = y - 1;
    if (key == 'd') x = x + 1;
    if (key == 's') y = y + 1;
    if (key == 'a') x = x - 1;
  }

  // Keep player on the screen
  if (x < 0) y = width;
  if (x > width) x = 0;
  if (y < 0) y = height;
  if (y > height) y = 0;

  // Draw the player
  point(x,y);
}
```

En kommentar är inte läst av din dator

Inne i funktion lever en lille gubbe som gör saker. Gubben har ett namn för saker som kommer in i ingångshålet, `text` i den här fall. Gubben vet var utgångshålen är, men i den här fall blir den inte användt.

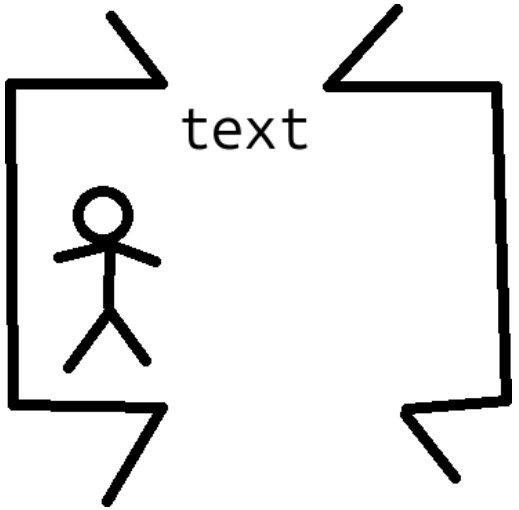

Figure 9: En funktion inifrån

Detta är hela gubbens värld. Tekniskts heter detta att funktionsargumenter har ett lokalt omfång (engelska: 'local scope'): dem bara är kända som detta inom funktionen.

Skapa ett nytt funktion som heter `show_error`. Om du ger funktion en `String`, visar den `ERROR:` och värdet av `String`en. Test funktion med att använda den i din kod.

Det är hjälpsams att ser en funktion som den här tva cartoonerna här nere. Först, en funktion utifrån är likadant en låda med ett namn (i den här fall `show`), ett ingångshål från ovan sida, och ett utgångshål på nere sida:

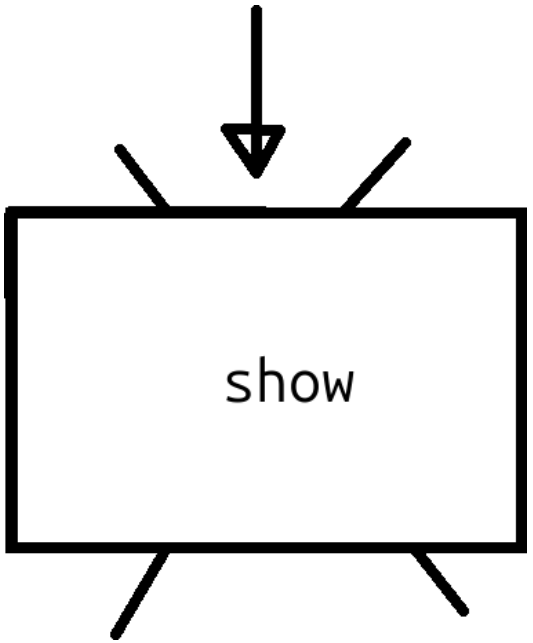

Figure 8: En funktion utifrån

Kolla på den kod här nere: vilka kommentar kann du skriva här? Om du tror det är bättre, du får ändra radföljden!

```
float cannon_angle = 0;

void setup()
{
  size(300, 200);
  strokeWeight(5);
}

void draw()
{
  background(255, 255, 255);
  final float x_mid = width / 2;
  final float y_mid = height / 2;
  final float x_cannon = x_mid + (cos(cannon_angle) * 20);
  final float y_cannon = y_mid - (sin(cannon_angle) * 20);
  line(x_mid, y_mid, x_cannon, y_cannon);
  ellipse(x_mid, y_mid, 20, 20);
}
```

33.2. Svar

Bara den `draw` funktion har saker:

```
void draw()
{
  // Clear the background
  background(255, 255, 255);

  // Calculate where to draw the cannon
  final float x_mid = width / 2;
  final float y_mid = height / 2;
  final float x_cannon = x_mid + (cos(cannon_angle) * 20);
  final float y_cannon = y_mid - (sin(cannon_angle) * 20);

  // Draw the cannon
  line(x_mid, y_mid, x_cannon, y_cannon);
  ellipse(x_mid, y_mid, 20, 20);
}
```

33.3. Att skapa en procedyr

En procedyr en type av funktion som har inga argument (vad det än är) och ger inget värde tillbaka (vad det än är). Du känner redan två: `setup` och `draw`.

Att skapa en procedyr är lätt:

```
void name_of_procedure()
{
  // Code in procedure
}
```

Att använda en procedyr i din kod är lätt också:

```
name_of_procedure();
```


36.2. Att visa fel

Här skapar vi en funktion själv, som visar text:

```
void show(final String text)
{
  println(text);
}
```

Här ser vi några saker:

- Funktionen heter `show`
- Mellan rundparenteser (`(` och `)`) finns **argumenter** som går inne i funktionen. I den här fall är det ett argument, av datatyp `String`. På grund av `final` kan värdet inte blir ändrat.
- Mellan måsvingar (`{` och `}`) finns **funktionskroppen** och där blir inmatningen `text` använt: den blir skickad till `println`.

36.1. Svar

Här ser vi några saker:

- `draw` är en funktion som gör inget. Dät är fint: vi vill bara göra något en gång.
- Nyckelordet `final` dyker upp. `final` betyder att värdet av variabeln kan inte andras efter variablen fick den värde. Om du kan använda `final`, så göt detta!
- Vi använder tre `String`. Värdet av en `String` är mellan dubbla citattecken (`"`).
- Vi använda ett plus tecken (`+`) att koppla ihop fler `String`s, så att dem blir ett ord
- Vi använder en funktion som heter `println`, som skriver i konsoln ('Console', längst nere) av Processing

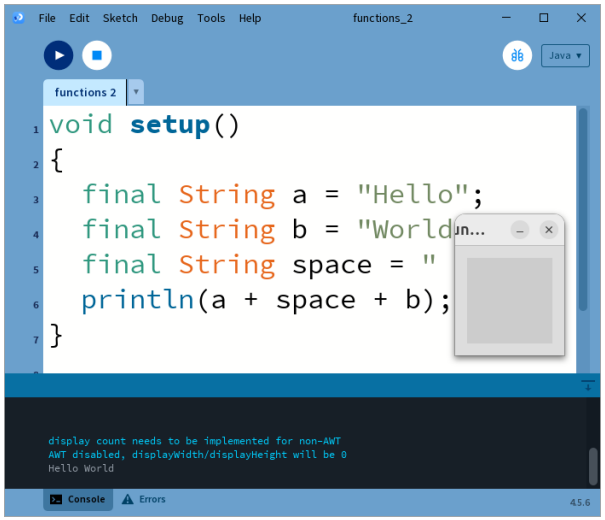

Figure 7: `println` skrivar i konsoln av Processing

Nu använder vi den sista exempel, och visar hur det ser ut:

```
void clear_background()
{
  background(255, 255, 255);
}

void draw_cannon()
{
  // Calculate where to draw the cannon
  final float x_mid = width / 2;
  final float y_mid = height / 2;
  final float x_cannon = x_mid + (cos(cannon_angle) * 20);
  final float y_cannon = y_mid - (sin(cannon_angle) * 20);

  line(x_mid, y_mid, x_cannon, y_cannon);
  ellipse(x_mid, y_mid, 20, 20);
}

void draw()
{
  clear_background()
  draw_cannon();
}
```

Så har vi just skapad en `draw` funktion (som är en procedyr) som är väldigt lätt att läsa: den är precis engelska, utan konstigheter!

En bra programmerare skriver kod som är mestadels lätt att läsa

Vi har också ersätt kommentarer med namn av funktioner. Det är klokt, för det behövs inte att säga samma sak två gånger.

En bra programmerare föredra att använda klara funktionnamn över att skriva en kommentar

Ordningen av funktionerna har inget betydelse: du får ordnar det som du tror är bäst.

33.4. Slutuppgift

I ditt eget program, använder åtminstone 5 procedyr. Varje procedyr måste göra precis vad den säger. Du får försvara dina val, men om en lärare är inte övertigad, räknas en procedyr inte.

36. Funktioner 2

Funktioner hjälper en programmerare att uttrycka sina tankar bättre: istället av göra samma sak med felmeldningen överallt, kan man göra det med en funktion.

36.1. En `String`

En `String` är en datatype för text, dvz ingen, ett, eller fler karaktär.

Kolla på och kör följande kod. Kan du förstå vad allt betyder?

```
void setup()
{
  final String a = "Hello";
  final String b = "World";
  final String space = " ";
  println(a + space + b);
}

void draw() { }
```

35.5 Slutuppgift

Här har vi koden av en boll som studsar horisontellt:

```
float x = 300;
float hastighet = 2;

void setup()
{
  size(600, 100);
}

void draw()
{
  ellipse(x, 50, 100, 100);
  x = x + hastighet;
  if (x > 550)
  {
    hastighet = -hastighet;
  }
  if (x < 50)
  {
    hastighet = -hastighet;
  }
}
```

Använder en klass för positionen kallad `Position`. En position har en x-koordinat kallad `x`'. Klassen är i en fil kallad `position.pde`.

Använder en klass för hastigheten kallad `Velocity`. En hastighet har en ändring i x-riktningen kallad `dx`, läsas som (svenska) 'di-ex', eller (engelska) 'delta `x`'. Klassen är i en fil kallad `velocity.pde`.

Får koden att funkar med dina klass.

34. `enum` och spelstatus

Du har redan använt **datatypen** `float`. En datatyp är arten av data, till exempel, en kommavärde. Men ibland vill man säga något annat, till exampel, i vilken tillstånd en spel är. Med en **enum** kan du göra detta.

34.1

Här har vi kod som visar vår problem:

```
float game_state = 0;

void setup()
{
  size(320, 200);
  textSize(32);
}

void draw_menu()
{
  text("Menu", 0, 32);
}

void draw_game()
{
  text("Game", 0, 32);
}

void draw()
{
  if (game_state == 0) draw_menu();
  if (game_state == 1) draw_game();
}
```

Problemet är att det är onaturligt att använda en noll eller ett för en speltillstånd. Istället ska vi envända en uppräkning (på engelska: ‘enumeration’) med nyckelordet `enum`.

Här ser en uppräkning ut:

```
enum GameState {
  MENU, GAME;
}
```

Namnet av en uppräkning börja med en stor bokstav och har kamelfall (engelsa: ‘Camel case’), dvs stora bokstäver för varje nästa ord. Namn av uppräkningselementer är i skrikfall (engelse: ‘Scream case’), dvs alla bokstäver är stora.

Med den här uppräkning i din kod, kann du översätta den oklara kod här:

```
float game_state = 0;
```

till den här klarare kod:

```
GameState game_state = GameState.MENU;
```

Också på andra ställe kan ändra `0` till `GameState.MENU` och `1` till `GameState.GAME`.

Får koden att funkar med en uppräkning.

I den huvud tab stannar kvar den här kod:

```
Position ball_position;

void setup()
{
  size(320, 200);
  ball_position = new Position(160, 100);

}
void draw()
{
  ellipse(ball_position.x, ball_position.y, 10, 10);
}
```

35.4. Svar

Koden blir, i `position.pde`:

```
class Position
{
  float x;
  float y;
  Position(float any_x, float any_y) {
    x = any_x;
    y = any_y;
  }
};
```


34.1. Svar

```
enum GameState {
  MENU, GAME;
}

GameState game_state = GameState.MENU;

void setup()
{
  size(320, 200);
  textSize(32);
}

void draw_menu()
{
  text("Menu", 0, 32);
}

void draw_game()
{
  text("Game", 0, 32);
}

void draw()
{
  if (game_state == GameState.MENU) draw_menu();
  if (game_state == GameState.GAME) draw_game();
}
```

34.2. Slutuppgift

Lägg till en uppräkningsvärde: `GAME_WON` (man kan inte dö i ditt spel ännu). Använder ett nytt funktion kallad `draw_game_won` för att visar detta.

I menyn, om du trycker på en tangent, blir tillståndet av spelet till `GAME`. I spelet, om du trycker på en tangent, blir tillståndet `GAME_WON`. När speleran har vunnit, om hen trycker på en tangent, blir tillståndet `MENU` igen.

Flytta koden av klassen `Position` hit.

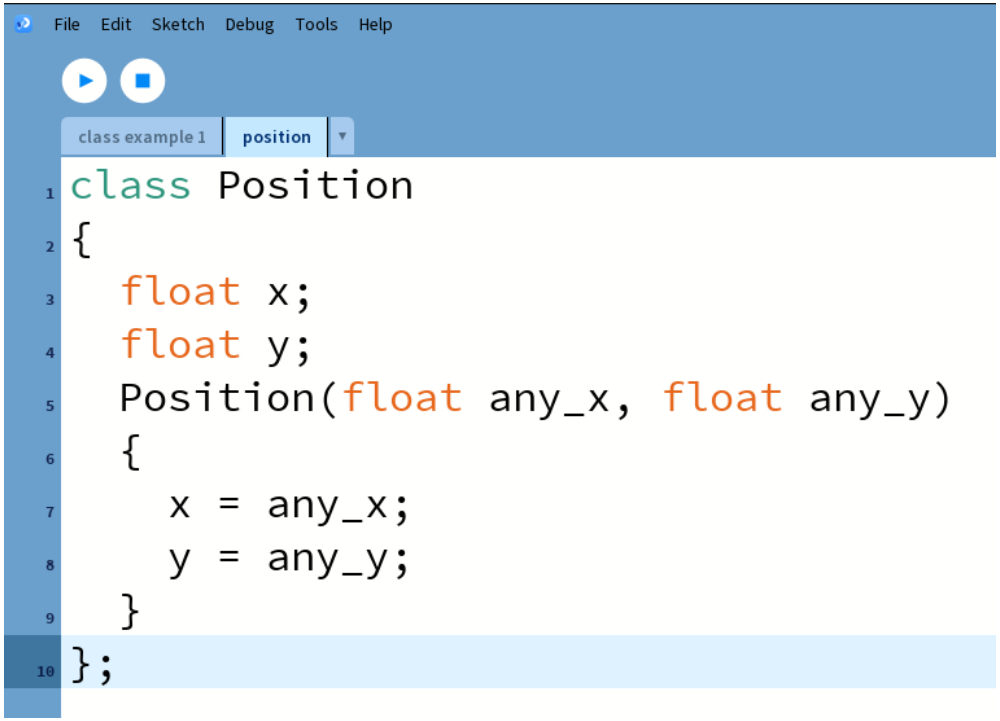

Figure 6: Flytta koden av klassen `Position` hit

Nu kan du köra koden som vanligt.

Flytta koden av klassen `Position` till sin egen fil.

Processing visar att `position.pde` är ännu tomt.

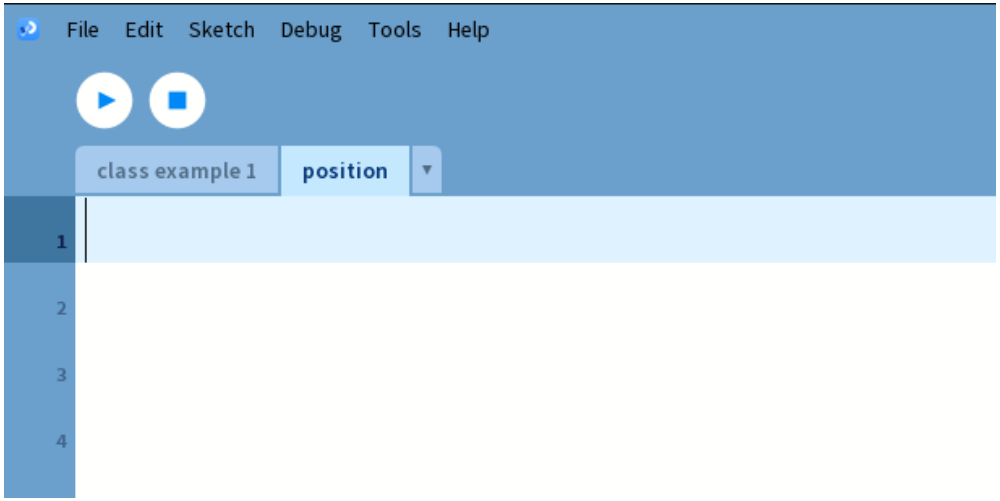

Figure 5: Filen `position.pde` är ännu tomt

35. Klasser 1

Klasser är ett sätt att bunta data. Under den här lektion skapar vi en sådant klass.

35.1. Att använder en `Position` klass

Sedan du har programmerat en bol som studsar snett, har du skrivit följande:

```
float x = 160;
float y = 100;

void setup()
{
  size(320, 200);
}
void draw()
{
  ellipse(x, y, 10, 10);
}
```

Du har skapat två variabler, även om den här två variabler hör ihop: tillsammans är dem **positionen** av bollen.

Så kan vi uttrycka att en position har ett x och ett y koordinat:

```
class Position
{
  float x;
  float y;
};
```

Ett klassnamn startar med ett stort bokstav (i den här fall en `P`). Klassen `Position` (läs den här på engelska, så 'på-sis-jön') har två **medlemsvariabler** kallad `x` ('ex') och `y` ('vaj'). En klass alltid slutar med en semikolon (`;`).

Man läser den här kod som: 'En `Position` **har** ett `x` och ett `y`'. Nyckelordet är 'har': en mening så här måste vara rimligt svenska/engelska. Till exempel, meningen 'en `Position` har en hastighet' är nonsens. Istället kan man säga 'en `Player` har en `Position` och en hastighet'.

Du kann ser att `position.pde` är kallad sådant i en filutforskare:

Figure 4: `position.pde` är kallad sådant

Ger ett namn för din nya fil, i den här fall `position` (med lite bokstav).

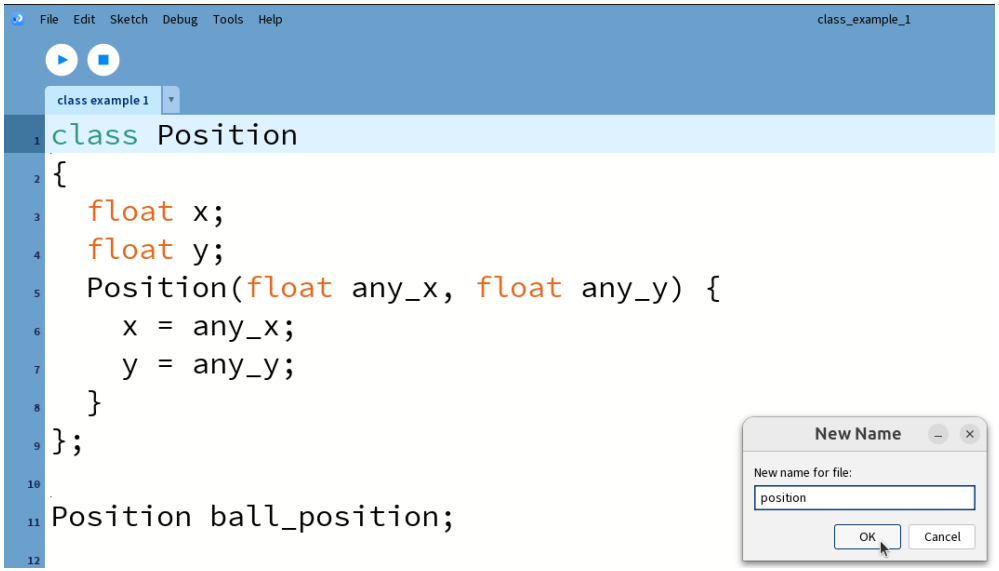

Figure 3: Ger ett namn för din nya fil

Du har nu skapad en fil kallad `position.pde`.

Nu kan vi skapar en variable som heter `ball_position` så här:

```
Position ball_position;
```

Igen, namnet av en klass börjar med ett stort bokstav, namn av ett variabel med ett lite bokstav.

Inte glöm att initialisera detta i `setup` funktionen, så här:

```
ball_position = new Position();
```

För att skriva x-värdet av positionen, gör man:

```
ball_position.x = 160;
```

Den period (`.`) läser man som 'av', i den här fall: '(över)skriv `x`-värdet av `ball_position` med `160`'

Att läsa x-värdet av position går identiskt:

```
ellipse(ball_position.x, 100, 10, 10);
```

Får den hela koden att funkar med en `Position` klass.

35.1. Svar

```
class Position
{
  float x;
  float y;
};

Position ball_position;

void setup()
{
  size(320, 200);
  ball_position = new Position();
  ball_position.x = 160;
  ball_position.y = 100;

}
void draw()
{
  ellipse(ball_position.x, ball_position.y, 10, 10);
}
```


Klicka på ‘New tab’.

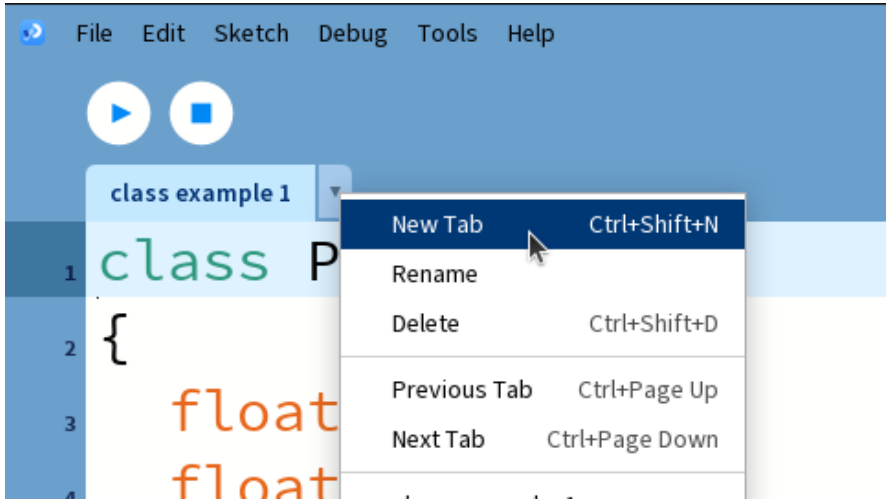

Figure 2: Klicka på ‘New tab’

35.3. Svar

```
class Position
{
  float x;
  float y;
  Position(float any_x, float any_y) {
    x = any_x;
    y = any_y;
  }
};

Position ball_position;

void setup()
{
  size(320, 200);
  ball_position = new Position(160, 100);

}
void draw()
{
  ellipse(ball_position.x, ball_position.y, 10, 10);
}
```

35.4. En egen fil för en klass

Det är vanligt att en klass har sin egen fil.

35.2. En kontructor som saknas

I koden finns något som är lite mycket skrivning:

```
ball_position = new Position();
ball_position.x = 160;
ball_position.y = 100;
```

Ändra den här koden till:

```
ball_position = new Position(160, 100);
```

Vilket felmeldning får du?

35.2. Svar

Du får felmeldningen likadant:

```
The constructor sketch_260806a.Position(int, int) is undefined
```

Processing frågar oss snäll (som vanligt) att skriva en konstruktor.

35.3. Att skriva en konstruktor

Ändra koden av `Position` till den här:

```
class Position
{
  float x;
  float y;
  Position(float any_x, float any_y) {
    x = any_x;
    y = any_y;
  }
};
```


Vi har just skapat en så-kallade 'constructor'. Detta är ett engelskt ord för svenskt 'konstruktor', som betyder 'detta som skapar'. Man säger 'med den konstruktor av `Position` kan du skapa en `Position`'

Den här konstruktor har två **argument**: `any_x` och `any_y`. Man kann inte kalla dem `x` och `y`, för att i nästa två rader blir Processing förvirrad (`x = x;` är väl lite konsigt). I en konstruktor läser man `x = any_x;` som: 'sätt **medlemsvariabeln** `x` till värdet av `any_x`.

Lägg till konstruktorn i din kod.