Programmering 2 NTI Johanneberg

Uppgifter programmering 2

Jakob Widebrant
jakob.widebrant@ga.ntig.se

7 oktober 2020

Sammandrag

Här följer uppgifter som lämpar sig för kursen programmering 2 som har fokus på objekt orienterad programmering (OOP). Uppgifterna är skrivna för att passa i Ruby men kan troligen göras i många andra språk också. Det finns även ett facit till många av uppgifterna. Detta ligger i en separat PDF.

Skapad av:
NTI-johanneberg
Jakob Widebrant
Leo Lisinski

Innehåll

1 Intro objekt

Följande uppgifter är till för att skapa ett grundläggande förståelse för hur objekt fungerar. Uppgifterna ska öva på att sätta, hämta och förändra state i objekt.

1.1 En person

Skapa en class som är en person. Följande saker ska/bör finnas med:

- Personen ska ha ett förnamn och ett efternamn och en konstruktor/intilize som tar två parametar (first_name, last_name).
- Det ska gå att komma åt fist_name och last_name men inte kunna ändra på dem, med andra ord ska det finnas minst 2 getters.
- Lämpligt är att ha totalt tre getters, en first_namn, en last_name och en för hela namnet.
- Extra uppgift: lägga till så man kan ändra namnet.

Testa även i din testkod att skriva

Skapa en class som är ett fordon. Följande skaer

- Fordonet borde ha: en modell/märke (volvo, fiat, nissan, Koenigsegg m.m), typ (lastbil, buss, bil m.m.), antal hjul, färg.
- Skapa lämpliga getters.
- Skapa lämpliga setters. T.ex färgen kan ju ändras men modellen/märket kommer ju alltid förbli den samma.
- Extra uppgift: Vill vi kunna fördefineraa saker så att användaren eventuellt inte behöver ange dessa. Ändra din konstruktor så att användaren inte behöver ange t.ex färgen och/eller modellen. Bra syntax kan vara:
  - Vill man att detta ska fungera så kan man i Ruby definiera vad defult är. Detta gör man genom att skapa en konstruktor och sätta sina variabler till -"i sin konstruktor, då får de det värdet om inget annat skulle ges. Detta fungerar olika i olika programspråk.
  - t.ex: `initialize(modell = nil, antal_hjul, color = ''white")` här kommer användaren vara tvungen att ange antal hjul men varken färg eller modell behövs.
  - Vad händer när vi anropar en getter och vi har satt modellen till `nil`?
- (svår) Extra uppgift: inför en ny egenskap som håller koll på om bilen kör eller inte, håll räkningen på hur många bilar totalt som kör. Tänk på att bilarna ska kunna både stanna och starta.
- (svårare) Extra uppgift: Kan du hålla koll på vilka bilar som kör och står stilla? Med andra ord inte bara hålla koll på hur många utan även bilens namnöm den kör.

2 Arv och inkapsling, privata och publika funktioner

Här kommer uppgifter om arv och inkapsling, även lite om privata och publika funktioner.

Bra syntax kommer vara `class minClass < classSomJagÄrverFrån` och `super()`

2.1 Superklassisk uppgift kring arv: däggdjur, husdjur och hundras

Här ska vi jobba med djur och testa hur arv fungerar. Först skapa en klass som är ett "generellt" däggdjur. Hur generellt väljer du själv och detta är något som kommer vara svårt när vi jobbar med arv, man vill inte vara för specifik för då kan vi sluta upp med en orm som flyger ej eller för generell får då blir klassen rätt värdelös. Lämpligt är att ha någon initialize som t.ex tar in ålder, färg, antal_ben. Kanske ha någon "generell" (publik) funktion som typ är om djuret rör på sig och om det är dött eller lever.

2.1.1 Skapa ett husdjur

Skapa en class husdjur som ärver sina egenskapar från däggdjur, skapa lämpliga egenskaper och metoder. Lämpliga metoder och egenskaper är till exampel

- Ägare, kanske både i initialze, en getter och en setter om vi skulle vilja ändra ägare.
- Är djuret inomhus eller inte?
- Behövs bur? I såna fall är djuret i buren eller inte? Om du skapar en setter så kan det vara lämpligt att denna settern även ändrar på inomhus/utomhus, för troligen är djuret inomhus om den är i buren.

Tips: är att du inte helt behöver/ska skriva om initilaze metoden, använd `super(argument1, argumnet2, argument3, ovs...)` för att kalla på föräldrar klassens metod. Testa vad som händer när du kallar på föräldresklassens metoder.

2.1.2 Skapa en hund

Skapa en class som är hund. Denna ska ärva sina egenskaper av husdjur. Skapa lämpliga egenskaper och metoder. Testa vilka metoder som hunden har, har den både husdjur och däggdjur eller bara en av dem?

2.2 Varuautomat och innehåll

Du ska skapa en varuautomat som innehåller diverse godis/läsk m.m. Man ska kunna betala med olika valuta (typ dollar och svenska kronor) och få ut varorna. Automaten ska ha koll på hur mycket pengar den har men detta ska inte en användare av din automat kunna komma åt.

Ditt program ska minst ha:

- En class som är varuautomat
- Bra metoder och egenskaper skulle kunna vara:
  - Vart varuautomaten är placerad. Detta ska kanske kunna ändras men inte nödvändigtvis.
  - Hur mycket pengar den innehåller. Vi vill kanske ha privat en setter och en privat getter men tänk på vad en användare ska kunna komma åt och vad den inte ska kunna komma åt. Det kanske även ska finnas pengar i den från början.

  - En metod som kollar och ändrar valuta
  - En metod som presenterar innehållet
  - En metod låter användaren köpa något med olika valuta. användaren bör då få varan och eventuell växel tillbaka.
  - Lämpligt är ju om varuautomaten hanterar innehållet på formen som du skapar i classen innehåll
- En class som är ”innehållet”
- Egenskaper som innehållet har är kanske:
  - Namn (konstruktor och getter)
  - Pris (konstruktor och getter, eventuellt setter om vi vill kunna ändra priset)
  - Eventuellt hållbarhetsdatum
  - Eventuellt allergier. Om någon köper något med nötter i så kan det vara bra att varna kunden för det.
  - I princip alla getters kan vara publika för ”innehålls” klassen.

Bra extra grejer att försöka sig på:

- Kan du lösa så man vet hur många varor som finns? Kanske inte så användaren ser det men så automaten skickar ett medelande till någon/”admin” när den börjar få slut på någon vara. Om det blir lite varor kvar kan du skicka till en ”dummy” metod som egentligen borde skicka ett meddelanden men inte gör det nu. t.ex:
  - `if antal_varor >= 3`
  - `send_message_to_admin()`
  - `end`
  - `def send_messeage_to_admin()`
  - `#dummy`
  - `end`
- Kan du göra en underkategori av automat som kanske enbart har läsk? Man kanske inte får ut en läskflaska utan får fylla på i sitt glas. Hur gör man då för att kolla hur mycket som är kvar?

2.2.1 Samling av automater

Extra extra: Går det att göra en class för flera automater? Så har man koll på hela lagret. Kan även vara lämpligt att om det är slut på en vara att kunden får ett medelande om ”närmast”/andra automater som har varan. En lösning som jag tror är bra på uppgiften är att göra automater till en underklass (ärva) från samling av automater, men inte nödvändigtvis bäst. (obs: jag (Jakob) tror att detta är görbart men svårt, dock har jag inte testat det själv ännu så det kan både vara balubas svårt och super enkelt)

2.2.2 Inkludera webbserver programmering

OBS: Jakob kan nog inte hjälpa till med detta, men han är ju bra på att killgissa så man vet ju aldrig.
För er som är hype på webbserver programmering skulle man kunna implementera sin kod på en webbserver. En av de stora fördelarna med Ruby är att det ska vara väldigt lätt att använda på hemsidor m.m.. Dock som jag (jakob) förstår det så är det mer med att hantera mailadresser och proxy-servrar men jag tror även "vanlig" kod ska fungera. Då får man även tänka lite på hur det skulle se ut rent grafiskt. Bra länkar är `https://rubyonrails.org` och `https://stackoverflow.com/questions/34056486/linking-ruby-with-html-code`

3 Fortsättning: Arv och inkapsling, privata och publika funktioner

Här följer lite uppgifter kring viktiga funktionaliteter som vi använder oss av i OOP.

3.1 Bank

Skapa en bank, banken bör ha ett kassavalv som kan öppnas och stängas. Banken bör ha möjlighet för en användare att ta ut pengar, när användaren tar ut pengar bör det finnas någonting som kollar hur mycket pengar användaren har och om användaren får komma åt kontot. med andra ord bör det finnas en metod i klassen som ser ungefär ut som:

```
def make_withdrawal(amout)
    if accsess_allowed()
        open_safe()
        get_cash(amout)
        close_safe()
    end
end
```

Vilka saker bör vara privata och vilka bör vara publika? Denna uppgift är med vilje relativt oklar, ofta när man som programmerare ger sig in i en uppgift så är inte uppgiften bättre beskriven än ovan.

3.2 En tärning

Gör en klass som är en 6-sidig träning (även kallad D6). Man bör kunna rulla tärningen och få ut resultatet. (Ja den här uppgiften är oklar vad du ska ha för initialzie, getters och setter men det är lite tanken med den.)

3.2.1 Annat antal sidor tärningar

Gör en klass som ärver metoder från D6 men som kan vara ett annat antal sidor.

3.2.2 Flera tärningar

Gör en klass som har antingen D6 eller DX som föräldrar klass (du väljer själv vilken du tycker är lämpligast). Denna klassen ska kunna rulla flera stycken antal tärningar. Antalet tärningar

kan/bör hela tiden vara samma (med andra ord definieras i initialize) och alla tärningar bör rullas om varje gång.

extra: kan du göra så valfritt antal tärningar rullas om? Kan du skapa så att alla tärningar under ett visst värde rullas om ett antal gånger? Tänk t.ex yatzi där man kanske vill gå för 4 st 6:or och då vill man kanske rulla om alla tärningar som är 5 eller under, MEN man får bara rulla om 2 gånger.

3.3 Svår: Polynom

Uppgiften är kopierad men delvis modifierad. Original kommer från Chalmers. Ett polynom är (alltid) givet på formen

$$p(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + ... + a_n \cdot x^n$$

för några konstanter a0, a1, . . . , an. Där n är gradtalet av polynomet och varje a_i är kallade koefficienter. Skriv en klass som är Polynom. Ett obejkt av denna klass är alltså ett polynom. Klassen borde ha följande "egenskaper":

- Polynomest gradtal (förmodligen en integer), sätt i detta fall koefficienterna till "1"
- En array med koefficienterna [a1, a2, a3, ... , an]
- En Initialize som kan skapa ett polynom där bara gradtalet är angett. Den borde klaga om man ger ett tal som är 0 eller mindre.
  - initialize kanske ser ut så här:
  - `def initialize(degree=nil, array_with_coeffisients=nil)`
- En metod som tar adderar två polynom, syntax kan vara till exempel `polynom3 = polynom1.add(po`
  Tänk på vad din "add" funktion returnerar?
- (svårare) en metod som multiplicerar två polynom
- En metod som beräknar värdet av polynomet i en viss punkt

Om vi adderar två polynom (av samma längd) så går det till på följande sätt:

$$(a_0 + a_1 \cdot x + ... + a_n \cdot x^n) + (b_0 + b_1 \cdot x + ... + b_n \cdot x^n) = (a_0 + b_0) + (a_1 + b_1) \cdot x + ... + (a_n + b_n) \cdot x^n$$

Om vi adderar två polynom av olika längd kan vi sätta de "saknade" koefficienterna till noll

$$(a_0 + a_1 \cdot x + ... + a_m \cdot x_m) + (b_0 + b1 \cdot x + ... + b_n \cdot x^n) = (a_0 + b_0) + (a_1 + b_1)x + ... + (a_m + b_m)x^m + b_{m+1}x_{m+1} + ... + b_n \cdot x^n$$

m i det här fallet är mindre än n

Produkten av två polynom är given av

$$(a_0 + a_1 \cdot x + ... + a_m \cdot x^m)(b_0 + b_1 \cdot x + ... + b_n \cdot x^n) = c_0 + c_1 \cdot x + ... + c_{m+n} \cdot x^{m+n}$$

där

$$c_i = a_0 \cdot b_i + a_1 \cdot b_{i-1} + ... + a_i \cdot b$$

3.4 Svår: Pokemon battle

Uppgiften är kopierad men delvis modifierad. Original kommer från Chalmers. Obs: Svår uppgift. Även så blandar jag svenska och engelska i uppgifttexten (sorry för det)!

Pokemon are small monsters who fight each other in rock-paper-scissors-like battles. Every pokemon has exactly one type: fire, water, or plant. When two pokemon fight:

- If the two pokemon have the same type, then either can win.
- If a plant type pokemon fights a water type pokemon, the plant pokemon wins.
- If a fire type pokemon fights a plant type pokemon, the fire pokemon wins.
- If a water type pokemon fights a fire type pokemon, the water pokemon wins.
-

Din pokemon klass borde bland annat ha metoderna "getType()" som förmodligen retunerar en sträng och "getName()" som också förmodligen retunerar en sträng.

3.4.1 Del 1

Your task: write an class FighterPokemon which implements the Pokemon interface. The FighterPokemon class must have a single method canDefeat(Pokemon opponent) which returns true if the pokemon represented by this could possibly defeat the pokemon represented by opponent, and otherwise returns false. That is, the method should return true if and only if:

- The two pokemon have the same type (in which case either could win), or
- The this pokemon has a type which beats the type of the opponent pokemon.

3.4.2 Del 2

Some pokemon do not only have a type, but also a level. A pokemon's level represents its combat strength: if two pokemon of the same type fight, the one with the higher level wins. If two pokemon of different types fight, the rock-paper-scissors rule still applies (i.e. water beats fire, etc.). Your task: write an class LeveledPokemon, which extends FighterPokemon and has the following members:

- A constructor which takes the initial level of the pokemon.
- A method getLevel() which returns the level of the pokemon.
- A method canDefeat(LeveledPokemon opponent), which returns true if and only if the this pokemon could possibly defeat the opponent pokemon, taking the both the types and levels of the two pokemon into account.

For full marks, your solutions to tasks (a), (c) and (d) must observe the DRY (Don't Repeat Yourself) principle. You should also make all instance variables private (som dom är per defult i ruby).

4 Rekursiv programmering

För att ge er ett verktyg till har vi rekursivprogrammering. Det är att anropa sin egen funktion. Bra exempel och övningar hittar du här:
`https://csharpskolan.se/article/rekursiva-algoritmer/`

4.1 Input till 20

Skriv ett program som skriver ut talen 1 till 20. Använd ej en loop utan använd rekursivitet!

4.2 40 till 20

Skriv ett program som skriver ut talen 40 till 20 Använd ej en loop utan använd rekursivitet!

4.3 Fibonatcci

Räkna ut tal i Fibonatcci sekvensen. Fibonaccis talföljd är: 1, 1, 2, 3, 5, 8, 12, 20... Med andra ord är nästa tal summan av de två tidigare talen. Be en användare säga hur många tal du ska beräkna i Fibonaccis talföljd och som ger tillbaka högsta/sista talet

Wikipedia sidorna för både Fibonacci (och Gyllene snittet) är väldigt bra, läs gärna dom för mer information.

Scope och avgör kod Här finns uppgifter som bygger på att man ska kolla vad andras kod är och gör.

5 Inledande uppgifter

Vet inte hur många mer uppgifter än så här det blir men här är de första uppgifterna i alla fall.

5.1 Vad/vilka fel finns i koden

Titta på koden nedan och avgör vilka fel som finns, försök sätta namn på felet (t.ex är det syntax fel eller scope-fel eller logist fel). Exemplet är från lektionen 2020-09-30

```
$min_globala_variabel = 10
min_inte_globala_variabel = "vart syns denna?"

def synlighet()
    counter = 10
    while counter > 1
        puts "multimeter"
        counter -= 1
    end
    puts min_globala_variabel
    puts min_inte_globala_variabel
    return counter
end

synlighet()
puts hej
```

5.2 Globalclass variabel och global variabel

Sättet vi skriver globala variabler på i ruby är med hjälp av $, vi vet också att vi kan skriva classvariabler med hjälp av @@. Försök komma på exempel när man vill ha en global variabler och INTE en classvariabel och tvärtom (när vi vill ha en classvariabel men inte en globalvariabel).

5.3 Läsa annans kod

Titta på följande kod:

```
def p(s) {
    for i in 0..s.length {
        if s.[i] != s.[s.length()-i-1]
            return false
        end
    end
    return true
end
```

Förklara vad koden ovan för och föreslå ett bättre namn på metoden och variablerna

5.4 Vad gör koden?

Uppgiften är kopierad men delvis modifierad. Original kommer från Chalmers. Vad gör de tre sista raderna i koden? Diskutera och resonera, du kan testköra koden för att se om du hade rätt. Försök förklara vad som sker.

```
class Question1
    def go(task)
        if task == "task a"
            Bepa.new.doTheThing(100)
        end
        if task == "task b"
            Bepa.new.doTheThing("the thing")
        end
        if task ==  "task c"
            Cykel.new.printValuePlus(5)
        end
    end
end

class Apa
    @value = 10;
    def initialze(value=10)
        @value = value
    end
    def getValue()
        return @value
    end
    def doTheThing(arg)
```

```
        puts "arg =  #{arg}"
    end
end

class Bepa < Apa
    def initialze()
        super(42)
    end
    def getValue()
        return super()
    end
    def doTheThing(arg)
        if arg.class == String
            puts ("ignoring arg")
        elsif arg.class == Apa
            super(arg.to_s)
        else
            super(arg.to_s)
        end
    end
end

class Cykel < Bepa
    value = 1000;
    def printValuePlus(extra)
        puts getValue()# + extra
    end

    def getValue()
        puts "hej"
        return super()
    end
end

Question1.new.go("task a")
Question1.new.go("task b")
Question1.new.go("task c")
```

5.5 (Svår) Vad gör koden? funktion som skickar till funktion

Einstine och Max Planck har äntligen genom kvantmekanik lyckats uppfinna teleporten. Dock har de strulat till lite vart och hur saker skickar. De gör några test försök och ser var saker landar.

Hjälp dessa två duktiga fysiker och se vart saker tar vägen! Du kan givetvis copy-pasta koden för att få facit på om du hade rätt men försök tänka igenom och fundera själva först.

```
def teleport_main_gate(person)
    if person == "Jakob"
        teleport_b(person)
```

```
    elsif person == "Leo"
        teleport_c("Karin")
    elsif person.class == Integer
        teleport_nummer()
    else
        tillbaka()
    end
end

def teleport_b(person)
    puts person
    puts "Välkommen till Jamaica"
end

def teleport_c(person)
    puts person
    puts "Välkommen till underjorden"
    return
end

def tillbaka()
    text = "Du skickas tillbaka"
    return text
end

def teleport_nummer()
    puts "detta var ingen person"
    tillbaka()
end

teleport_main_gate("Jakob")
teleport_main_gate(152)
teleport_main_gate("Leo")
teleport_main_gate("leo")
```

Ändra i standard biblioteken Vi ska här testa ändra standard biblioteken för att ändra dem till det vi önskar, detta är något som vi gör relativt sällan men när vi väl gör det är det jättekraftfullt. Det skapar även väldigt bra förståelse för hur arv och föräldrar klasser fungerar.

6 Jakob testar overleaf syntax nedan

```
#this is a comment

a = 5
b = a * 5
puts b
puts b + 3
```

```ruby
class Test < Test::SomeClass
@test = 3
    def bar
        if foo
            puts "foo"
        else
            puts "bar"
        end
    end
end
```