Opgaver uge 3

Formål med opgaverne

Efter disse øvelser skal du kunne opbygge simple Java-programmer. Du skal forstå hvordan opdeling i klasser fremmer modularisering, og du skal kunne bruge objektkonstruktion (`new`), metodekald, logiske operatorer (`&&`, `||`), betingede sætninger (`if-else`) og strengsammensætning.

Godkendelsesopgaverne A.1, A.2 og A3 skal afleveres til øvelserne tirsdag (uge 4) til instruktorerne; sammen med de andre godkendelsesopgaver B, C, D og E er deres godkendelse en forudsætning for at gå til eksamen i kurset. I bedes derfor være sikre på at i aflevere til tiden.

I må gerne arbejde sammen om godkendelsesopgaverne, men skal aflevere individuelt og personligt kunne stå inde for besvarelserne. Se på LearnIT for formalia i forbindelse med afleverings opgaver. Der vil i også kunne få status over jeres opgaver efter de er blevet rettet.

Aflevér en samlet besvarelse for A.1 og en samlet besvarelse for A.2 og en for A.3 med **navn** og **email-adresse** på alle besvarelser. Brug programkommentarer til at svare på evt. spørgsmål i delopgaverne.

Opgave 3.1

Disse opgaver omhandler ur-eksemplet: Lav B&K opgave
3.10, 3.11, 3.12, 3.17, 3.27

Opgave 3.2 – tjekopgaver

Tjekopgaver om logiske udtryk: B&K 3.13, 3.14, 3.15, 3.16 — vink: sidstnævnte opgave kan løses med logisk "ikke" (`!`) og logisk "eller" (`||`).

Tjekopgaver om modulo-operatoren: Lav B&K 3.21, 3.22,3.23 3.24, 3.26.

Vink 1: Ved besvarelse af 3.23 og 3.24 kan du gå ud fra at `n` og `m` er positive tal, altså større end nul. (Opgaven er mere subtil hvis de også kan være nul eller negative).

Vink 2: En nem måde at eksperimentere med Java-udtryk er at gå ind i BlueJ og vælge VIEW || SHOW CODE PAD. Derved fremkommer et vindue nederst til højre i BlueJ hvor man direkte kan skrive Java-udtryk såsom `8%3` og få dem udregnet når man trykker Enter. (Det går lidt langsomt på grund af den måde BlueJ er lavet).

Godkendelsesopgave GA.1

Denne opgave går ud på at modellere en (ret lille) skov med tre træer. Et træ har en alder i år og højde i meter. Hvert år øger træet sin højde med et bestemt antal procent, afhængig af træart.

(i) Lav en klasse `Tree` i BlueJ med tre felter:

- `age` af type `int` som er træets alder.
- `height` af type `double` som er træets højde.
- `growthPct` af type `double` som er træets tilvækst i procent per år.

Klassen `Tree` skal også have en konstruktor `Tree(double growthPct)` som sætter `age` til 1, sætter `height` til 0.25 og sætter feltet `this.growthPct` til `growthPct`.

(ii) Lav en metode `void growOneYear()` som forøger `age` med 1 og forøger `height` med `growthPct` procent. Det kan udtrykkes som `height * (1 + growthPct/100)`.

(iii) Lav en metode `void show()` som udskriver træets alder og højde, fx i dette format: `alder = 17 år og højde = 3.66 meter`.

(iv) Lav en klasse `Forest` der har tre felter `tree1`, `tree2`, `tree3` af type `Tree`. Klassen skal have en konstruktor `Forest()` der initialiserer de tre felter med hvert sit `Tree`-objekt; disse `Tree`-objekter skal have forskellige `growthPct`-parametre, fx 10.0, 25.0, og 40.0 procent.

(v) Lav en metode `show()` i klassen `Forest`, der udskriver skovens tilstand ved at kalde de tre træers `show`-metoder samt udskrive en blank linje.

(vi) Lav en metode `growOneYear()` i klassen `Forest`, der lader skoven vokse et år (ved at kalde de tre træers `growOneYear()`-metoder) og derefter udskriver skovens tilstand.

(vii) Modificer metoden `growOneYear()` i `Tree`-klassen så et træ holder op med at vokse når det er blevet 20 meter eller højere. (I en mere realistisk simulering vil træet selvfølgelig også dø når det er blevet tilstrækkelig gammelt. Hvis du har lyst kan du indføre et felt `alive` af type `boolean` som er `true` så længe træet er levende, og sættes til `false` når træet er blevet mere end 120 år. Husk i så fald at tilpasse `show()`-metoden også).

Godkendelsesopgave GA.2

Denne opgave handler om andengradspolynomier af formen:

$$p(x) = ax^2 + bx + c$$

hvor a, b og c er (reelle) koefficienter.

(i) Lav en klasse `Quadratic` med en konstruktor der tager tre `double` parametre `a`, `b` og `c`, nemlig polynomiets koefficienter.

(ii) Lav en metode `void show()` som udskriver polynomiet. For eksempel, hvis `a`, `b`, `c` er 3, 9 og -30, så skal der skrives `3.0x^2 + 9.0x - 30.0` eller `3.0x^2 + 9.0x + -30.0` eller lignende.

(iii) Lav en metode `double compute(double x)` som beregner værdien af $ax^2 + bx + c$ for det givne x. Med eksempelkoefficienterne ovenfor skal `Compute(3)` give $3 \cdot 3^2 + 9 \cdot 3 - 30 = 24$.

(iv) Lav en metode `void solve()` som løser andengradsligningen, dvs. finder de x for hvilke $ax^2 + bx + c = 0$. Din metode skal undersøge om der er nul, én eller to løsninger, ud fra om $d = b^2 - 4ac$ er negativ, nul, eller positiv, og skal udskrive en besked om antallet af løsninger samt selve løsningerne (x-værdierne). Husk på at når der er én eller to løsninger x er de givet ved

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \text{og} \quad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

Med eksempelkoefficienterne ovenfor skal der være to løsninger $x_1 = 2$ og $x_2 = -5$. Vink: I Java kan kvadratet x^2 enten udtrykkes som `x*x` eller som `Math.pow(x, 2)`; og kvadratroden $\sqrt{x}$ kan udtrykkes som `Math.sqrt(x)`.

Husk at afprøve metoderne i `Quadratic` klassen med forskellige koefficientsæt. Især skal du prøve `solve()` med tre forskellige koefficientsæt, som give ingen, én eller to løsninger.

(v) Prøv at lave et `Quadratic` objekt hvor $a = 0$ og $b = 1$ og $c = -5$, og kald `solve()`. Resultaterne ser formentlig ret meningsløse ud. Hvad er der gået galt i din metode? Sagen er at når $a = 0$ er det slet ikke en andengradsligning men en førstegradsligning $1 \cdot x - 5 = 0$, og så duer formlerne ovenfor ikke: `solve()` forsøger at dividere med $2a$ som er nul.

Lav en ny metode `superSolve()` som undersøger om $a = 0$ og i så fald udskriver løsningen $x_1 = -c/b$, og ellers kalder `solve()`.

(vi) Overvej om ikke også `superSolve()` kan risikere at dividere med nul. Hvordan? Lav en metode `robustSolve()` som undgår også dette problem, og derfor virker for alle værdier af a, b og c.

Godkendelsesopgave GA.3 – en mere fleksibel skov

Opgave GA.1 simulerede i en skov med præcis 3 træer, hvilket både er urealistisk og besværligt. Her skal du — med den samme `Tree`-klasse — lave en ny `BigForest`-klasse der kan rumme vilkårligt mange træer.

(i) Klassen `BigForest` skal have et felt `trees` af type `ArrayList<Tree>`. Konstruktoren skal initialisere feltet til en tom arrayliste.

(ii) Lav metoder `void show()` og `void growOneYear()` i klassen `BigForest` med samme virkning som i klasse `Forest`. Metoderne kan hensigtsmæssigt bruge en foreach-løkke til at gennemløbe arraylisten og sørge for at alle træerne behandles.

(iii) Lav en metode `void addTree(Tree t)` i klassen `BigForest` som tilføjer det givne træ til arraylisten.

(iv) Lav en metode `void growManyYears(int n)` i klasse `BigForest` som simulerer at skoven vokser i `n` år. For hvert år vokser alle eksisterende træer, og derefter føjes der et nyt træ til skoven; det nye træ skabes med udtrykket `new Tree(10.0)`.

(v) Lav en metode `double averageHeight()` i klasse `BigForest` som beregner og returnerer den gennemsnitlige højde af skovens træer.