02100 Indledende Programmering og Softwareteknologi

Gruppeaflevering 3

Afleveres senest:
Søndag d. 2. november kl. 23.59

Jørgen Villadsen og Carsten Witt

Se afleveringsopgave 1 for yderligere generel information. Bemærk at Java-filerne skal være i en ZIP-fil, men der må ikke være andre filer eller kataloger i denne (I skal ikke aflevere filer, som I har fået udleveret).

Opgave 1

Skriv et program `PrimeFactors` som tager et heltal som input fra brugeren og printer dets primfaktorisering. Et tal n's primfaktorisering er den entydige følge af primtal hvis produkt er lig n. Eksempelvis er primfaktoriseringen af 12 lig $2,2,3$ idet 2 og 3 begge er primtal og $2 \cdot 2 \cdot 3 = 12$. Primfaktoriseringen af 30030 er $2,3,5,7,11,13$ og primfaktoriseringen af 3757208 er $2,2,2,7,13,13,397$.

I jeres program bør I benytte heltalstypen `long` i stedet for `int`, da det muliggør at I kan indtaste større tal. Værdierne af en `long` går fra -2^{63} (omtrent -10^{19}) til $2^{63}-1$ (omtrent 10^{19}). I bør skrive jeres program således at man efter hver primfaktorisering bliver spurgt om man ønsker at indtaste et nyt tal.

Eksempel:

```
Enter integer greater than 1 (0 to terminate): 24
List of prime factors: 2, 2, 2, 3

Enter integer greater than 1 (0 to terminate): 1111111111111111
List of prime factors: 1111111111111111

Enter integer greater than 1 (0 to terminate): 9201111169755555649
List of prime factors: 3033333343, 3033333343

Enter integer greater than 1 (0 to terminate): 9223372036854775783
```

```
List of prime factors: 9223372036854775783

Enter integer greater than 1 (0 to terminate): 9223372036854775807
List of prime factors: 7, 7, 73, 127, 337, 92737, 649657

Enter integer greater than 1 (0 to terminate): 0
```

Start med at få programmet til at virke med `int` for tal op til 2147483647 og ret dernæst til `long` således at I kan få de samme resultater som i eksemplet ovenfor. De midterste tre primfaktoriseringer kan tage nogle minutter...

Primfaktorisering af store tal er eksempelvis centralt i forbindelse med kryptering af data som sendes over internettet. I det udbredte RSA-krypteringssystem benyttes produkter af store primtal. For at knække koden skal man være i stand til at primfaktorisere produktet. Som I kan se i forbindelse med jeres program er det forholdsvis uproblematisk at primfaktorisere tal op til størrelsen 2^{63}. I krypteringssystemer benytter man derfor meget større tal, typisk i størrelsesordenen af 2^{1000} eller endnu større.

Man kender ikke algoritmer (programmer) som kan primfaktorisere så store tal indenfor en realistisk tid, og derfor opfattes sådanne krypteringssystemer i dag som sikre. Den længste kode som indtil 2019 var blevet knækket var et produkt af to primtal af størrelsesordenen 2^{768} (kendt som RSA-768). Dette tal har 232 cifre og primfaktorerne blev fundet 12. december 2009 (den oprindelige computer, der dannede tallet, var med vilje blevet destrueret). Det tog hvad der svarer til næsten 2000 år på en 2.2 GHz PC at primfaktorisere tallet!

Se evt. mere her: `https://eprint.iacr.org/2010/006.pdf`

I 2019 og 2020 var der nye faktoriseringsrekorder, jf. `https://en.wikipedia.org/wiki/Integer_factorization_records`, med henholdsvis 240 og 250 cifre.

Opgave 2

Formålet med opgaven er at skrive et program, der kan håndtere artikler og deres referencer.

Skriv en klasse `Forlag`, hvis instanser repræsenterer forskellige forlag. Et forlag er beskrevet ved navnet på forlaget (f.eks. "University Press") og stedet, hvor forlaget ligger (f.eks. "Denmark"). Klassen skal således have to felter: `navn` og `sted`. Klassen skal også indeholde en konstruktør til at oprette et nyt forlag med forlagets navn og sted angivet som parametre.

Skriv en klasse `Tidsskrift`, hvis instanser repræsenterer tidsskrifter. Et tidsskrift er beskrevet ved en titel, et forlag og et ISSN-nummer. Klassen skal således have tre felter: `titel`, `forlag` og `issn`. ISSN-nummeret er en entydig kode, som alle tidsskrifter har (en streng). Forlaget skal have typen `Forlag`. Klassen skal også indeholde en konstruktør til at oprette et nyt tidsskrift med dets titel angivet som parameter. Tidsskriftets ISSN-nummer og forlag skal kunne sættes efterfølgende med to metoder `setIssn` og `setForlag`.

Skriv en klasse `Artikel`, hvis instanser repræsenterer artikler i tidsskrifter. En artikel er beskrevet ved en liste (et array) af forfattere, en titel, en angivelse af tidsskriftet, den er pu-

bliceret i, og en reference-liste. Reference-listen indeholder listen (arrayet) over de andre artikler, som den pågældende artikel refererer til. Klassen skal således have følgende felter: `forfattere`, `titel`, `tidsskrift` og `referenceliste`. Klassen skal også indeholde en konstruktør til at oprette en ny artikel. Konstruktøren skal tage forfattere, titel og tidsskrift som parametre, idet elementerne i referencelisten senere skal kunne sættes med en metode `setReferenceliste`, som også skal skrives.

Hvis ikke andet er specificeret, så kan strenge benyttes som felters type.

Klasserne skal testes grundigt. Specielt skal der skrives en klasse `ArtikelTest` med en `main`-metode, som opretter følgende:

- Forlaget *University Press*, *Denmark*.
- Tidsskrifterne *Journal of Logic* og *Brain*. Disse to tidsskrifter kommer begge fra *University Press*. ISSN-numrene kendes ikke.
- Følgende to artikler:
  - A. Abe & A. Turing: "A". Journal of Logic.
  - B. Bim: "B". Journal of Logic.

  Den første af disse artikler har en reference til den anden.

Benyt `toString`-metoder i klasserne (dog ikke i `ArtikelTest`-klassen).

Opgave 3

Denne opgave er mere fri end de foregående, og I opfordres til at være kreative i jeres løsning af opgaven. Opgaven er også lidt vanskeligere og mere omfattende end de foregående. Opgaven handler om spillet *Racetrack* (også kaldet *vektor-rally* på dansk). Racetrack spilles normalt med papir og blyant, men i denne opgave går det ud på at implementere spillet i Java. Spillet er beskrevet på følgende Wikipedia-side, som I bør læse medmindre I allerede kender spillet i forvejen:

`http://en.wikipedia.org/wiki/Racetrack_(game)`

Jeres opgave er at lave et program `RaceTrack` som implementerer spillet i Java, og som benytter biblioteket `StdDraw` til at vise resultatet. Spillet skal foregå på en bane som først skal tegnes op. Til dette formål får I brug for følgende metode fra `StdDraw`:

- `StdDraw.line(x0,y0,x1,y1)`, som tegner en linje fra punktet med koordinaterne $(x0, y0)$ til punktet med koordinaterne $(x1, y1)$.

I kan måske også få brug for følgende metoder:

- `StdDraw.setPenColor(c)`, som ændrer farven af de efterfølgende punkter og linjer til `c`. Værdien `c` er af typen `Color`. I biblioteket `StdDraw` findes der følgende konstanter af typen `Color`:

  ```
  BLACK, BLUE, CYAN, DARK_GRAY, GRAY, GREEN, LIGHT_GRAY,
  MAGENTA, ORANGE, PINK, RED, WHITE, YELLOW
  ```

  Man kan altså for eksempel give kommandoen:

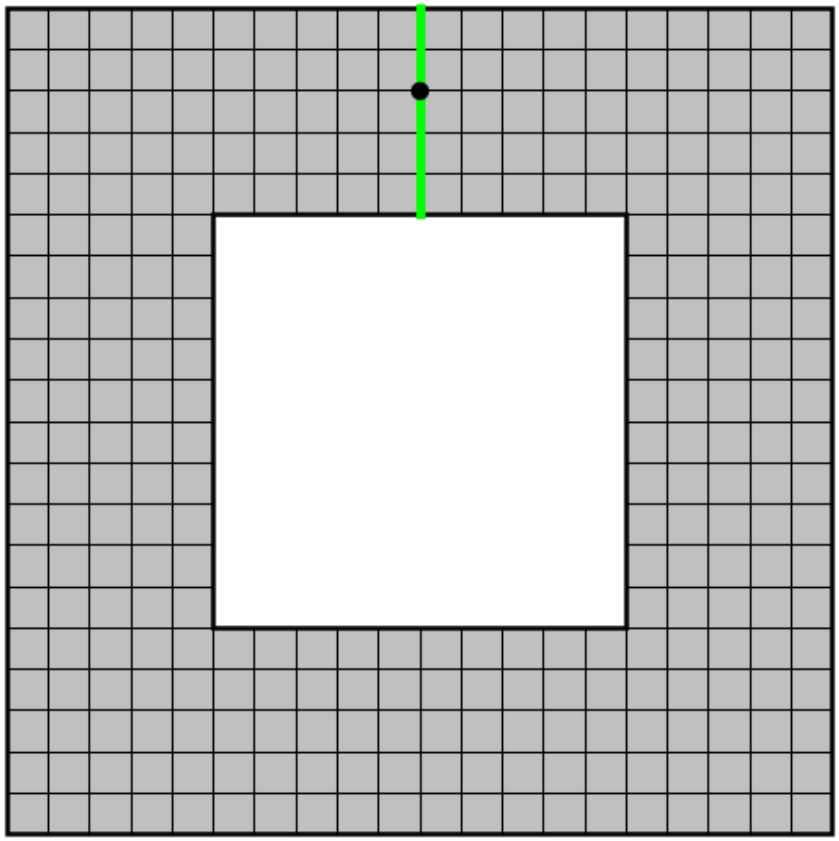
  ```
  StdDraw.setPenColor(StdDraw.RED);
  ```

  hvorefter alt bliver tegnet med rød farve.

- `StdDraw.square(x,y,r)`, som tegner et kvadrat af længde $2r$ med centrum i koordinaterne (x,y).

- `StdDraw.filledSquare(x,y,r)`, som tegner et udfyldt kvadrat af længde $2r$ med centrum i koordinaterne (x,y).

I begyndelsen opfordres I til blot at lave en meget simpel bane som ser ud i stil med figur 1.



Figur 1: Simpel Racetrack-bane.

På figuren udgør det grå område banen, og den lodrette grønne streg er start- og mål-linjen. Hvis I ønsker mere udfordring kan I altid senere eksperimentere med mere komplekse baner. I bør dog allerede fra starten muliggøre, at banen kan have forskellig størrelse (men samme form).

I spillet er brugeren føreren af en racerbil. Racerbilen starter ved start- og mål-linjen og skal køre rundt på banen uden at havne udenfor banen. Hvis man havner udenfor banen har man tabt, og spillet skal afsluttes med en besked til konsollen.

Brugeren styrer racerbilen ved hjælp af de numeriske taster 1–9. Hvis brugeren taster 5 efterfulgt af enter skal racerbilen flytte til det efterfølgende principielle punkt (se beskrivelsen af spillet på ovennævnte Wikipedia-side). Hvis en af de andre numeriske taster benyttes skal racerbilen flytte til nabopunktet af det principielle punkt som svarer til placeringen af den numeriske tast på computeren, se illustrationen på figur 2.

7 ↖	8 ↑	9 ↗
4 ←	5	6 →
1 ↙	2 ↓	3 ↘

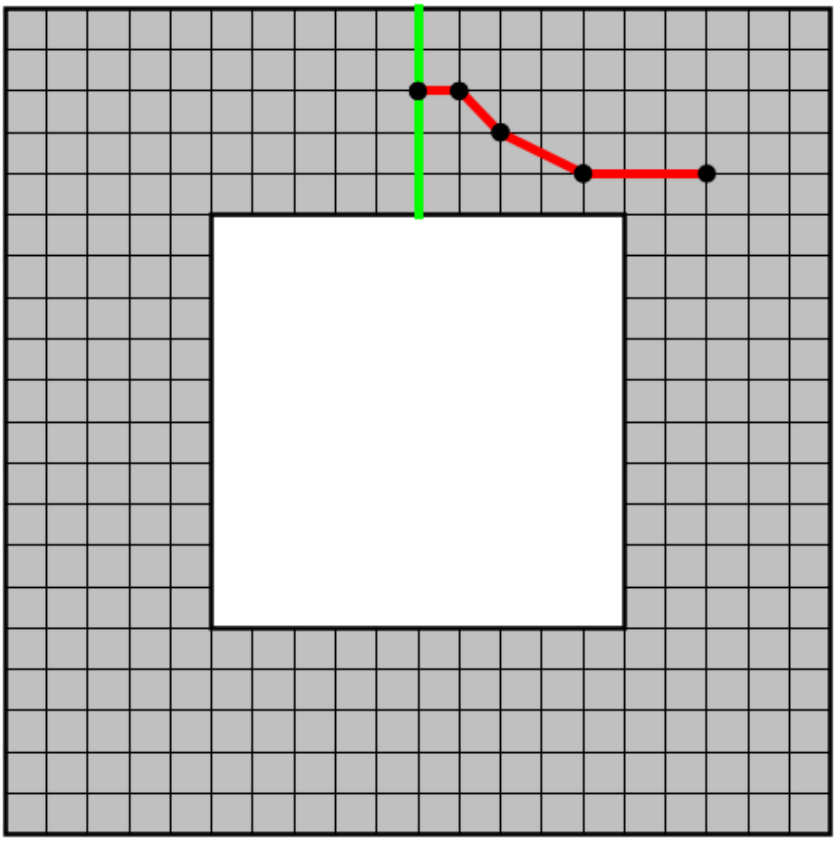
Figur 2: Illustration af taster til styring.

Hvis brugeren eksempelvis taster 7 efterfulgt af enter skal racerbilen flytte til punktet som ligger skråt oppe til højre (nordvest) for det principielle punkt.

Hver gang brugeren har indtastet et tal skal det vises med en linje i vinduet hvordan racerbilen har flyttet sig. Figur 3 viser et eksempel på hvordan vinduet kan se ud efter at brugeren har tastet sekvensen 6, 2, 6, 9.

Når det basale program er på plads kan I overveje hvilke udvidelser I vil indføre. Der er mange muligheder, for eksempel:

- Lav kode som registrerer når racerbilen passerer målstregen, og derefter angiver hvor mange træk bilen har foretaget. Spillet går ud på at gennemføre banen i så få træk som muligt.
- Lav kode som registrerer hvis racerbilen kører den forkerte vej.
- Lav kode som muliggører at spillet kan spilles med to (eller eventuelt flere) spillere. Tænk på hvad programmet skal gøre hvis to spillere havner på samme felt.
- Lav kode som muliggør at man kan spille mod computeren. Dette er en vanskelig opgave og anbefales primært til de som allerede har programmeringserfaring fra tidligere.
- Lav alternative baner som spillet kan spilles på.



Figur 3: Et igangværende Racetrack-spil.

I skal implementere mindst én af ovenstående udvidelser eller en anden udvidelse, som I selv finder på.

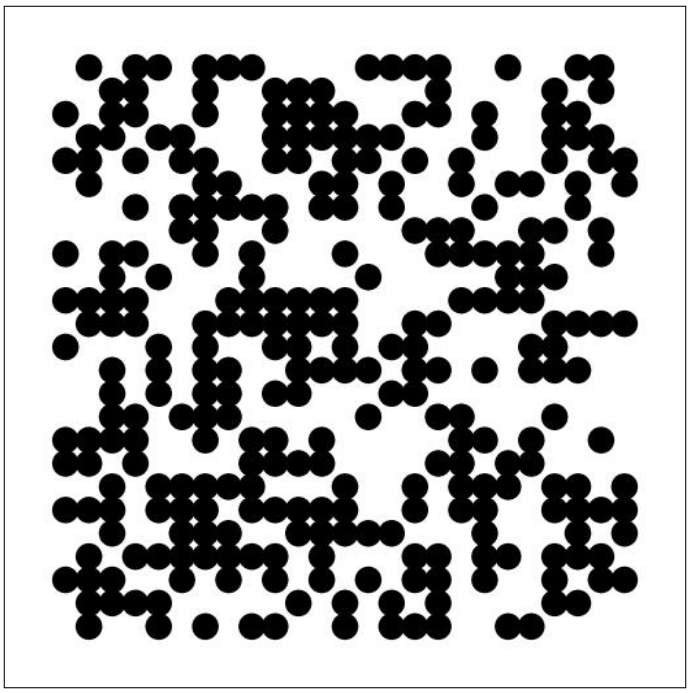
Sørg for at dokumentere jeres løsning: Hvordan anvendes programmet, hvordan virker det, hvilke udvidelser har I foretaget, hvilke begrænsninger har programmet, osv.

Opgave 4

Denne opgave omhandler Conway's *Game of Life*. Game of Life er en simpel grafisk simulator, hvor brugeren angiver simulationens starttilstand, og derefter kan følge hvordan tilstanden udvikler sig på ofte forundrende og uforudsete måder. Game of Life er beskrevet på følgende Wikipedia-side, men de grundlæggende elementer vil blive også gennemgået i opgaveteksten.

`https://en.wikipedia.org/wiki/Conway's_Game_of_Life`

I Game of Life består en tilstand af et n gange n grid af felter kaldet *celler*, hvor hver celle enten er død eller levende. For at visualisere en tilstand tegnes en sort prik på koordinat (x,y) hvis cellen på plads (x,y) er levende (en død celle markeres ikke).



Figur 4: Eksempel på mulig tilstand i Game of Life.

Hver celle har otte naboer således som illustreret på figur 5(a) (hver af de otte naboer til den midterste celle er givet et nummer).

På baggrund af en given tilstand kan den næste tilstand findes ud fra følgende regler:

(1) En levende celle med færre end to naboer er død af ensomhed i næste tilstand.

(2) En levende celle med flere end tre naboer er død af pladsmangel i næste tilstand.

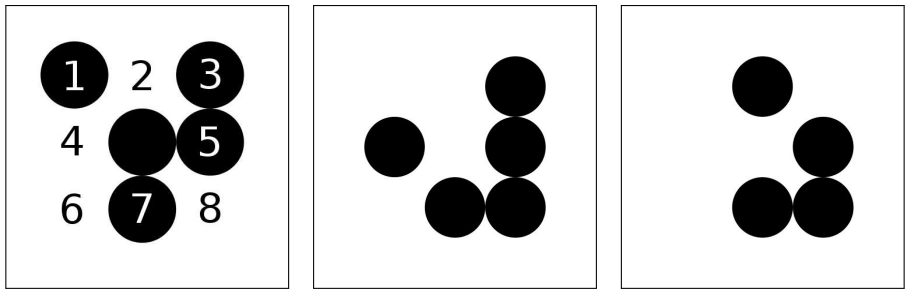
(3) En levende celle med to eller tre levende naboer lever uforandret videre i næste tilstand.

(4) En død celle med præcis tre levende naboer bliver vækket til live i næste tilstand.

Figur 5 viser hvordan en tilstand ændrer sig trin for trin. I trin 2 er den midterste celle fra trin 1 død af pladsmangel, da den har fire levende naboer; den øverste venstre celle er død af ensomhed, fordi den kun har én levende nabo, mens den nederste højre celle er vækket til live, da den har præcis tre levende naboer, osv.

1. Implementér Conway's Game of Life, hvor `StdDraw` benyttes til at visualisere hvordan en given tilstand ændrer sig — det bliver altså en slags lille animation. I bør implementere jeres løsning som mindst to klasser: `GameOfLife` og `GameOfLifeMain`.

   Klassen `GameOfLife` skal definere `GameOfLife`-objekter der repræsenterer en tilstand i en Game of Life-simulation. Klassen skal desuden tilbyde metoder til at manipulere denne tilstand, f.eks. at ændre værdien af en celle, simulere ét trin frem og lignende. Internt i `GameOfLife` foreslås det, at man repræsenterer tilstanden ved et 2-dimensionelt array af heltal (`int[][]`), hvor et 1-tal angiver en levende celle og 0 en død. Husk at



(a) Trin 1. Den midterste celle interagerer med sine otte naboer.

(b) Trin 2

(c) Trin 3

(d) Trin 4

(e) Trin 5

(f) Trin 6

Figur 5: Eksempel på udviklingen for en given starttilstand.

et 2-dimensionelt array af heltal svarer til en matrix i matematisk forstand. Tilstanden på figur 5(a) vil således være repræsenteret ved følgende matrix (2-dimensionelle array):

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Jeres `GameOfLife`-klasse skal mindst have følgende to konstruktører:

- `public GameOfLife(int n)`. En konstruktør til at generere et Game of Life med starttilstanden værende et `n` gange `n` grid af tilfældigt initialiserede celler.
- `public GameOfLife(int[][] initialState)`. En konstruktør til at at generere et Game of Life med starttilstanden repræsenteret i arrayet `initialState`.

I klassen vil I også få brug for en række metoder på `GameOfLife`-objekter. Eksempelvis bør der være en metode `nextState()` som genererer den efterfølgende tilstand i det aktuelle `GameOfLife`. I forbindelse med at skrive metoden `nextState()` kan det være en fordel at have defineret en privat hjælpemetode `liveNeighbours(int x, int y)`, som tæller hvor mange levende naboer cellen `(x,y)` har i det aktuelle `GameOfLife`.

Klassen `GameOfLifeMain` skal indeholde klientkoden, herunder `main`-metoden, som instantierer et `GameOfLife`-objekt og udnytter de metoder som objektet tilbyder.

For at sikre at jeres animation kører flydende, skal I benytte metoden `StdDraw.show(n)`, hvor `n` er et heltal (`int`). Denne metode tegner billedet og venter derefter `n` millisekunder. Efterfølgende kald af tegne-metoder vil ikke blive vist med det samme, men først ved næste kald til `StdDraw.show(n)`.

Metoden `StdDraw.clear()` kan benyttes til at slette det tegnede.

`StdDraw` benytter som standard en figur-størrelse på 512 gange 512 pixels, men hvis I eksempelvis ønsker en figur på 1000 gange 1000 pixels, så skal I blot kalde metoden `StdDraw.setCanvasSize(1000,1000)`.

Test jeres løsning på et par forskellige `GameOfLife`-instanser.

2. Programmet `GameOfLifeMain` skal nu udvides så det kan indlæse starttilstanden fra en fil. Formatet af filen er som en matrix af 0'er og 1-taller, f.eks. følgende som svarer til tilstanden på figur 5(a):

```
1 0 1
0 1 1
0 1 0
```

I filen `gol.zip` findes nogle eksempelfiler med forskellige interessante starttilstande. De ligger i filerne `toad.gol`, `pulsar.gol`, `pentadecathlon.gol`, `glider_gun.gol` og `acorn.gol`. Afprøv jeres program på disse filer.

Herunder er der nogle eksempler på mulige udvidelser af programmet. Det er ikke påkrævet at I laver nogle af disse udvidelser. *Det er kun hvis I har ekstra tid og ønsker ekstra udfordring.*

- Modificér programmet så det holder øje med om simuleringen på et tidspunkt bliver periodisk, altså om en bestemt sekvens af tilstande begynder at gentage sig igen og igen. Få i dette tilfælde programmet til at printe perioden, dvs. antallet af tilstande som forløber imellem to gentagelser.
- Lad banen være en *torus* således at øverste kant er forbundet med nederste kant, og venstre kant med højre (ligesom i computerspillet Pac-Man).
- Introducér farver på cellerne og lav forskellige regler for overlevelse som afhænger af disse farver.
- Find eventuelt mere inspiration på Wikipedia-siden!