Afleveringsopgave 3

Afleveres senest:
Søndag d. 25. oktober kl. 23.59

Jørgen Villadsen

Se afleveringsopgave 1 for 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): 92233720368547758O7
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 er 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: `http://eprint.iacr.org/2010/006.pdf`

Opgave 2

En 2-dimensionel *random walk* simulerer den tilfældige opførsel af en partikel, som bevæger sig i en plan — eller bevægelsen af en turist i Manhattan, som ved hvert vejkryds vælger en tilfældig vej at gå. I hvert skridt i en random walk vælger man at gå enten mod nord, syd, øst eller vest, alle med lige stor sandsynlighed.

Normalt foregår en random walk i et *grid*, dvs. et udsnit at planen delt op i felter som på et skakbrædt. Dette er illustreret på figur 1.

Felterne i griddet angives ved deres heltalskoordinater som på figuren.

Skriv et program `RandomWalk` som tager et heltal n som input fra brugeren og udfører en random walk på et grid med koordinaterne $(-n,-n)$ i nederste venstre hjørne og koordinaterne (n,n) i øverste højre hjørne. Figuren ovenfor svarer til et grid med $n = 2$.

Random walken skal starte i koordinatsættet $(0,0)$. I hvert skridt skal man enten bevæge sig ét felt op (mod nord), ét felt ned (mod syd), ét felt til højre (mod øst) eller ét felt til venstre (mod vest). Random walken skal fortsætte, indtil den første gang bevæger sig uden for det angivne grid.

Program skal som output give de enkelte koordinater som random walken passerer, samt til slut hvor mange skridt random walken har foretaget i alt.

(-2,2)	(-1,2)	(0,2)	(1,2)	(2,2)
(-2,1)	(-1,1)	(0,1)	(1,1)	(2,1)
(-2,0)	(-1,0)	(0,0)	(1,0)	(2,0)
(-2,-1)	(-1,-1)	(0,-1)	(1,-1)	(2,-1)
(-2,-2)	(-1,-2)	(0,-2)	(1,-2)	(2,-2)

Figur 1: Eksempel på et grid.

Eksempel:

```
Enter size of grid: 4

Position = (0,0)
Position = (0,1)
Position = (1,1)
Position = (1,2)
Position = (2,2)
Position = (1,2)
Position = (1,3)
Position = (2,3)
Position = (2,2)
Position = (3,2)
Position = (4,2)
Position = (4,1)
Position = (4,2)
Position = (4,3)
Position = (5,3)

Total number of steps = 14
```

I skal herefter udbygge programmet så det printer en random walk i et vindue på skærmen. Til dette formål får I brug for et grafik-bibliotek. Biblioteket ligger i filen `StdDraw.java` som I har fået en meddelelse om. Filen indeholder et simpelt grafik-bibliotek til Java. Når I har downloadet filen skal I efterfølgende sørge for at kopiere den ind i det aktuelle Java-projekt under Eclipse, så biblioteket bliver tilgængeligt. Det gøres på følgende måde. Først åbnes menupunktet

`File -> Import`. Derefter vælges 'File System' under 'General' og der klikkes efterfølgende på 'Next'. Der skal nu navigeres til det katalog som indeholder filen `StdDraw.java`. Klik dernæst 'OK'. I det fremkomne vindue kan man nu markere filen `StdDraw.java` og klikke 'Finish'. Sørg for at der ud for 'Into folder' står `src`-kataloget for det aktuelle Java-projekt.

Biblioteket `StdDraw` indeholder en række simple metoder, hvoraf I får brug for følgende:

- `StdDraw.setXscale(x0,x1)`, som sætter vinduets x-koordinater til at gå fra `x0` til `x1`. Både `x0` og `x1` har typen `double`.
- `StdDraw.setYscale(y0,y1)`, som sætter vinduets y-koordinater til at gå fra `y0` til `y1`. Både `y0` og `y1` har typen `double`.
- `StdDraw.setPenRadius(r)`, som sætter punktstørrelsen til at være `r`. Værdien `r` har typen `double`.
- `StdDraw.point(x,y)`, som placerer et punkt i koordinaterne (x,y) i vinduet. Både `x` og `y` har typen `double`.

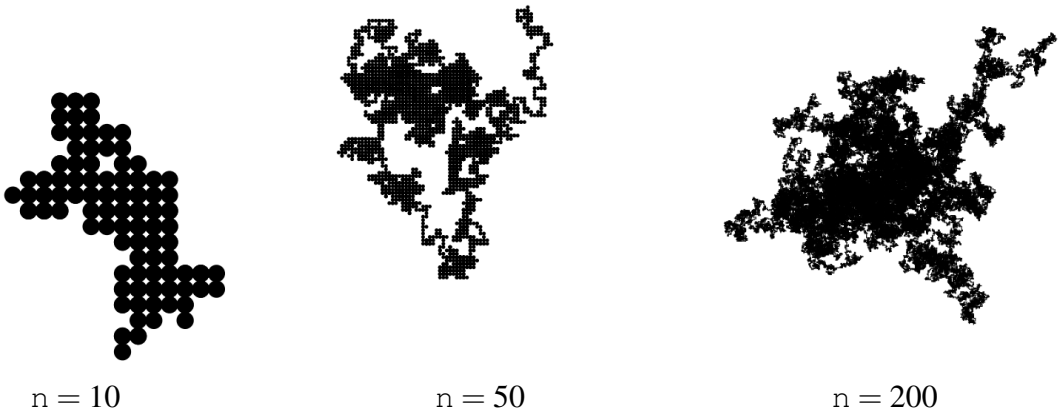
Her er et simpelt eksempel på et program som anvender `StdDraw` (som I har fået en meddelelse om):

```
public class TestStdDraw {
  public static void main(String[] args) {
    StdDraw.setXscale(-1, 1);
    StdDraw.setYscale(-1, 1);
    StdDraw.setPenRadius(2.0/1000);
    int n = 100;
    for (int i = 0; i < n*2*Math.PI; i++) {
      StdDraw.point(Math.cos(i/(double) n), Math.sin(i/(double) n));
    }
  }
}
```

Læs ovenstående kode igennem og prøv at forstå hvad koden gør. Afprøv dernæst programmet. Hvorfor står der `2.0/1000` i stedet for blot `2/1000`? Hvad sker der hvis man glemmer at skrive `(double)` foran `n`? Svarene på spørgsmålene skal ikke med i rapporten.

I skal nu benytte biblioteket `StdDraw` til at lave det modificerede program som printer en random walk i et vindue. Figur 2 giver et par eksempler på hvordan resultatet kan se ud.

Man kan gemme indholdet af et `StdDraw`-vindue i en jpg-fil ved at vælge `File -> Save` fra vinduets menu. Som filtype kan man vælge `.png` og `.jpg`. Filen kan dernæst inkluderes i rapporten. Afprøv jeres program på et par velvalgte input og medtag det grafiske output i jeres rapport.



Figur 2: Random walks illustreret ved brug af `StdDraw`.

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
  ```

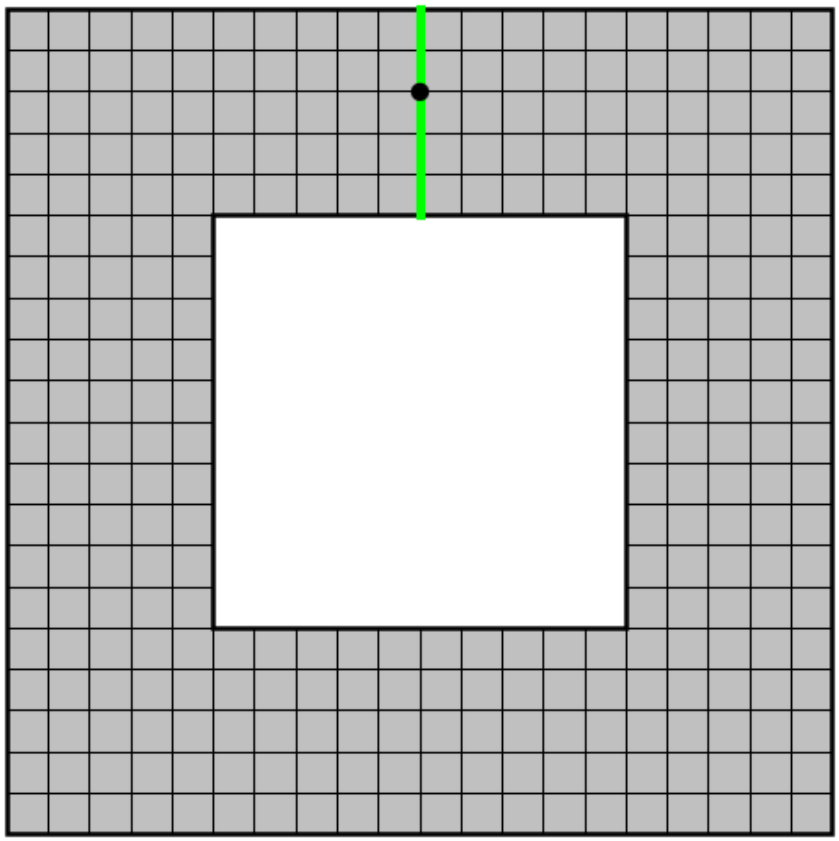
  Man kan altså for eksempel give kommandoen:

  ```
  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 3.



Figur 3: 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 4.

7 ↖	8 ↑	9 ↗
4 ←	5	6 →
1 ↙	2 ↓	3 ↘

Figur 4: 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.

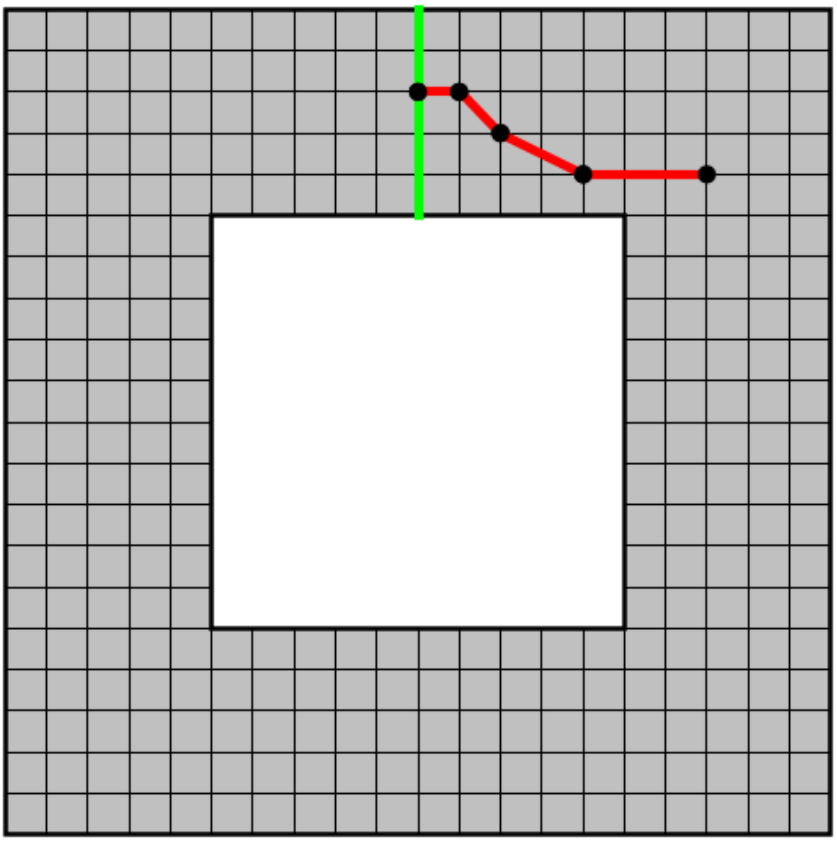
Hver gang brugeren har indtastet et tal skal det vises med en linje i vinduet hvordan racerbilen har flyttet sig. Figur 5 viser et eksempel på hvordan vinduet kan se ud efter at brugeren har tastet sekvensen 6, 2, 6, 9.

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.

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.



Figur 5: Et igangværende Racetrack-spil.