Eksamen IN1010 vår 2021

Les hele oppgaven før du starter å besvare den; det kan være nyttig informasjon om en deloppgave i senere tekst.

Svar på oppgave 1 og 2 skal sammen utgjøre et fullstendig program med main og alle nødvendige klasser. Du kan levere programmet samlet i en .java-fil, eller som én .java-fil per klasse. Du skal ikke levere .class-filer. Tegninger kan være håndtegning fotografert med mobil eller generert av et tegneprogram. Filer med tegning skal ha navn etter oppgaven de svarer på, og være på .png, .jpg eller .pdf-format – for eksempel **1a.jpg**. Alle filer som inngår i svar på oppgave 1 og 2 leveres samlet i én zip-fil.

Oppgave 3 og 4 leveres separat som hver sin .zip-fil.

Merk: Det er ikke et krav at programmene du leverer er kjørbare. Du blir ikke trukket for ubetydelige feil, selv om de gjør at programmet ikke kompilerer eller virker helt som det skal.

Du skal programmere etter gode objekt-orienterte prinsipper slik det er undervist i IN1010. Ikke bruk klasser fra annet enn java.lang-biblioteket til Java (som du ikke trenger å importere), unntatt der det er eksplisitt skrevet at det er lov.

Hvis du trenger en verdi som ikke er definert i oppgavesettet, kan du definere en konstant med en passende verdi.

I dette oppgavesettet skal du gjøre mange tilfeldige valg ved hjelp av klassen Trekk med en statisk metode **int trekkInt(int min, int max)** som returnerer et heltall x,
der min <= x <= max. Denne kan du bruke selv om du ikke har skrevet klassen i oppgave 1d.

En skog med stier og turgåere

I dette oppgavesettet skal du skrive et program som simulerer hvordan turgåere går i en skog med stier.

Oppgave 1. Skogen, stiene og stikryssene 30%

I skogen er det mange stier. I denne oppgaven sier vi at noen stier er kjerreveier, andre er naturstier. Alle stier i skogen er enten kjerreveier eller naturstier.
På noen stier er det veldig god utsikt, og det finnes både kjerreveier og naturstier med god utsikt. Den ekstra egenskapen at en sti har god utsikt skal du modellere som et interface.

I oppgave 1 går det meste ut på å bygge opp skogen med stier og stikryss. I oppgave 2 dreier det meste seg om å simulere (late som, etterape) turgåere som går i skogen.

Oppgave 1a 3%

Tegn klassehierarkiet for stiene i skogsområdet slik det er beskrevet over.
Slutt 1a

Skogen består av stier som møtes i stikryss. Du skal følgelig skrive klassene Skog, Kryss og alle klassene og interface-et du beskrev i oppgave 1a. Du har stor frihet i hvordan du programmerer klassene, bare de følger disse kravene:

Klassen Kryss skal ha en datastruktur som angir alle stiene som møtes i krysset. I klassen Kryss kan du bruke alle klasser fra hele Javas API. Når vi senere skal simulere at det går personer i skogen, må vi finne ut hvilken sti en person skal gå videre på når vedkommende står i et kryss. Skriv derfor en metode i klassen Kryss som returnerer en tilfeldig sti ut fra krysset. I denne metoden er det OK om det er samme vei som personen kom. Klassen Kryss trenger videre en boolsk metode som returnerer true om krysset er isolert, det vil si at ingen stier fører til eller fra dette krysset (for da er det for eksempel ikke gunstig å starte en tur her).

Klassen Sti skal ha en konstruktør som setter stiens lengde (i meter) og kryssene stien går mellom – dette er parametere til konstruktøren. En sti må inneholde referanser til to kryss, ett i hver ende av stien. En sti har ingen retning og det er OK at disse endene er det samme krysset, da blir stien bare en rundtur i skogen tilbake til samme kryss. Når vi senere skal simulere at det går personer i skogen, må vi finne ut hvor personen ender når vedkommende starter på en sti som går ut fra et krysset. Skriv derfor en metode finnAndreEnde i klassen Sti som, gitt det ene endepunktet (krysset), returnerer det andre endepunktet av stien. Videre skal klassen Sti ha en metode int beregnGaaTid(int fart) som regner ut hvor lang tid det tar å gå stien med den oppgitte farten og returnerer resultatet. Farten angis i meter per minutt, og gåtiden returneres i hele minutter. Om du avrunder eller fjerner sifre bak komma når du beregner resultatet spiller ingen rolle.

I en sti som har god utsikt er det et tall utsiktsVerdi mellom (og inklusive) 1 og 6 som forteller hvor fin utsikten er. Dette tallet skal også være en parameter til klassenes konstruktør. Konstruktøren sjekker at tallet er i området 1 til 6 og ellers kaster et unntak (subklasse av klassen RuntimeException) som du selv lager navnet på og definerer. Det er ingen forskjell på klassene for kjerreveier og naturstier annet enn navnet. For stier med utsikt skal det være mulig å lese av hvor fin utsikten er.

Klassen Skog skal ha en konstruktør som bygger opp datastrukturen av stier og kryss. Det skal være ANTSTIER stier og ANTKRYSS kryss og disse konstantene skal settes av klassens konstruktør. Klassen Skog skal inneholde en array som peker på alle kryssene. Konstruktøren i Skog bygger opp datastrukturen som følger: Opprett alle kryssene, foreløpig uten stier, og legg disse i arrayen for alle kryss. Opprette deretter alle stiene. For hver sti som opprettes trekkes to tilfeldige kryss stien skal gå mellom, type sti og lengde.

For hver sti som lages skal det være omtrent lik sjanse (25%) for at det blir en kjerrevei, en natursti, en kjerrevei med utsikt eller en natursti med utsikt. Når programmet lager en sti med utsikt, skal det være omtrent lik sjanse for at stien får hver av de seks karakterene. Stiens lengde trekkes tilfeldig mellom 220 og 2500 meter.

Legg merke til at det tilfeldige stisystemet som lages på denne måten, ikke er realistisk, men det bryr vi oss ikke om i oppgave 1-3. Det er for eksempel greit om det finnes kryss som ikke har noen stier (eller bare én).

Klassen Skog skal inneholde en metode hentTilfeldigKryss() som returnerer et tilfeldig kryss i skogen, og en annen metode hentTilfeldigStart() som garanterer at krysset som returneres har minst én sti slik at krysset er egnet som startpunkt for turgåere. Dere kan alltid regne med at minst ett slikt kryss finnes i skogen.

Oppgave 1b. Vekt 3%

Tegn en datastruktur med tre objekter av klassen Kryss og to objekter av en subklasse av klassen Sti. Du trenger ikke ta med metoder.
Slutt 1b

Oppgave 1c. 23%

Skriv klassene Skog, Kryss og alle klassene og interfacet du beskrev i oppgave 1a.
Slutt 1c

Oppgave 1d. 1%

Skriv klassen Trekk med en statisk metode int trekkInt(int min, int max) som beskrevet i innledningen. Metoden returnerer et tilfeldig heltall N der 0<=min<=N<=max. En måte å trekke et tilfeldig heltall fra og med 0 til og med N-1 er å skrive (int)(Math.random()*N).
Slutt 1d

Oppgave 2. Simulering av turgåere i skogen 45%

Basert på de programdelene du har laget i oppgavene over skal du nå lage et fullstendig program som simulerer mange turgåere i skogen. Du skal ikke bruke tråder noe sted i oppgave 1 eller 2.

I oppgave 2 skal du lage komponentene til en generell simulator, og deretter simulere turgåere ved hjelp av denne simulatoren. Det er sterkt anbefalt å lese hele oppgave 2 grundig, inkludert hvordan turgåere skal simuleres, før du starter implementeringen.

En enkel generell simulator
Du skal implementere en enkel, generell simulator ved hjelp av klassene Simulator, Aktivitet og PrioKo. Det er bare ett objekt av klassen Simulator, og det har en instansvariabel int globaltid som starter på 0. Klassen Aktivitet modellerer en generell *aktivitet*. Alle aktiviteter har en *handling* som utføres av en instansmetode void handling(). I tillegg har alle aktiviteter en instansvariabel int tid som angir når denne handlingen skal utføres neste gang. Det er derfor en invariant at tid >= globaltid for alle aktiviteter.
I hvert skritt i simuleringen skal handlingen i aktiviteten med lavest verdi av variabelen tid utføres. Aktiviteter skal derfor kunne legges i en *prioritetskø*, der den aktiviteten som har lavest tid hentes ut først. Prioritetskøen er implementert som en klasse PrioKo. Klassene Simulator, PrioKo og Aktivitet er designet for å jobbe tett sammen (de er tett koblet) og instansvariablene i klassen Aktivitet skal kunne aksesseres fra andre klasser i samme pakke (klasser som ligger i samme mappe). Du skal derfor ikke beskytte instansvariablene i Aktivitet ved å angi private (eller protected) tilgangsnivå på disse.

Klassen Aktivitet skal ha:

- to instansvariabler som refererer objektet med henholdsvis lavere og høyere verdi av instansvariabelen tid i prioritetskøen (brukes som neste- og forrige-pekere)
- en instansvariabel tid (se over).
- en abstrakt metode handling (se over).

Videre skal klassen implementere interface Comparable for å kunne sammenligne verdiene av instansvariabelen tid i to Aktivitets-objekter

Oppgave 2a Vekt 4%

Skriv klassen Aktivitet
Slutt 2a

Prioritetskø for aktiviteter
Uten bruk av noen klasser fra Java-biblioteket, lag en klasse PrioKo som implementerer en toveis prioritetskø av Aktivitet-objekter. Klassen har to metoder: settInn(a) og hentUt().

hentUt() tar ut og returnerer det første objektet i køen (det med minst tid).
settInn(a) leter bakfra (dvs starter med å sammenligne objektet som settes inn med det som har størst tid) og setter inn det nye objektet på riktig plass i prioritetskøen.

Oppgave 2b Vekt 10%

Skriv klassen PrioKo
Slutt 2b

Simulator
Du skal nå skrive klassen Simulator. Simulatoren skal inneholde en prioritetskø. Konstruktøren til Simulator skal ta som parameter en array med referanser til alle aktivitetene som skal simuleres og legge dem inn i prioritetskøen. Disse aktivitene vil kanskje ha tid lik 0 og det må være OK at prioritetskøen inneholder bare like tider.

Neste tidspunkt det skal skje noe er på tidspunktet bestemt av instansvariabelen tid i aktiviteten med lavest tid (det første objektet i prioritetskøen). Denne instansvariabelen oppdateres i metoden handling() i subklasser av Aktivitet.

Simulatoren skal inneholde en metode void simuler(int t) som simulerer aktivitetene; t er antall minutter simuleringen skal vare. Metoden simuler(int t) bør derfor inneholde en while-løkke slik: while (globaltid < t) { . . . }

Innmaten i while-løkken skal gjøre dette:

- Hente aktiviteten med lavest tid ut av prioritetskøen.
- Sette den globale tiden til denne tiden.
- Kalle metoden handling() i aktiviteten (siden denne metoden er abstrakt i Aktivitet må den defineres av en subklasse som for eksempel Turgaaer, se nedenfor). Som en avslutning på sin handling antar vi at denne metoden øker den lokale tiden til det tidspunktet i fremtiden der denne aktiviteten igjen skal utføres av simulatoren.
- Sette denne aktiviteten inn i prioritetskøen igjen (med den nye tiden).

Oppgave 2c vekt 10%

Skriv den fullstendige klassen Simulator
Slutt 2c

Simulere turgåere
Du skal nå bruke denne simulatoren til å simulere turgåere. Klassen Turgaaer er derfor en subklasse av Aktivitet.

En turgåer har en hastighet (i meter/minutt) og et sted (en referanse til et Kryss) der vedkommende enten er, eller er på vei til. Hastigheten og krysset der turgåeren starter er parameter til Turgaaer-s konstruktør. Alle turgåere starter med tid lik 0.

Når en turgåer forlater et kryss, skal programmet sette tiden i aktiviteten til tidspunktet turgåeren kommer frem til neste kryss (se lenger nede). Metoden handling() i klassen Turgaaer blir derfor kalt av simulatoren når en turgåer er kommet frem til et kryss. Metoden må velge hvor turgåeren skal gå videre ved å trekker en tilfeldig sti ut fra krysset der den er (og det kan være samme veien den kom). Det neste krysset turgåeren kommer til, er den andre enden av denne stien. Så kaller turgåeren metode beregnGaaTid i stien som skal gås, med sin egen hastighet som parameter. Basert på denne gåtiden beregner turgåeren når den er fremme i den andre enden (gammel verdi av tid (som er 0 når turgåeren først starter) + tiden det tar å gå til den andre enden av stien). Det siste metoden handling() gjør, er å sette tidspunktet når den er fremme i den andre enden av stien tilbake i instansvariabelen **tid**.

Oppgave 2d Vekt 10%

Skriv klassen Turgaaer.
Slutt 2d

Hovedprogram i en klasse TestSimulator
Skriv en klasse TestSimulator med en main()-metode slik at det hele virker sammen. Denne metoden skal lage skogen, lage alle turgåerne og lage simulatoren med alle turgåerne. La en konstant angi hvor mange turgåere som simuleres. For hver turgåer som lages trekkes det et tilfeldig startkryss der turgåeren skal starte. Det trekkes også hvor raskt turgåeren beveger seg; hastigheten er et tilfeldig tall i intervallet 20-200. Til slutt kalles metoden simuler(t) i simulatoren for å utføre simuleringen; parameteren er et heltall trukket tilfeldig i intervallet 30-480.

Oppgave 2e vekt 9%

Skriv klassen TestSimulator med en main()-metode som beskrevet over.
Slutt 2e

Oppgave 2f vekt 2%

I klassen Turgaaer kan en uheldig programmerer ved en feil prøve å sette ny tid til en tid mindre enn gammel tid. Dette vil ødelegge simuleringen. Hvordan burde klassen Aktivitet vært programmert for å lage klassen mer robust og hindre at tiden går bakover eller står stille? Skriv svaret ditt i en ny Java-fil kalt Aktivitet2.java. Legg gjerne inn noen kommentarer i filen som forklarer hva du foreslår. Aktivitet2 trenger ikke virke sammen med noen av de klassene du alt har skrevet i oppgavene 1 og 2.

Slutt 2f

Levering av oppgave 1 og 2: Alle programfiler og filer med tegninger samles i en zip-fil med navn Oppgave1_2.zip

Oppgave 3: Turgåere som tråder 15%

Vi ønsker å illustrere omtrent hvordan turgåere beveger seg i skogen på en annen måte ved å la hver turgåer være en tråd, og så kan tiden det tar å gå en sti «simuleres» med Thread.sleep().
Ta en kopi av løsningen din fra oppgave 1 og 2. Ta vekk simulatoren og klassene PrioKo og Aktivitet. Skogen med kryss og stier bygges opp akkurat som i oppgave 1. Det du hovedsakelig skal modifisere, er klassen Turgaaer. Turgåeren som før var en Aktivitet, skal nå bli en tråd. I tillegg skal du legge til blant annet to metoder i klassen Kryss. Du skal også skrive en modifisert main()-metode i en ny klasse TestTradSimulator som setter det hele i gang på riktig måte, men main() behøver ikke vente på at trådene blir ferdige.

For hele oppgave 3 gjelder: Det er meningen at sluttresultatet skal kunne være kjørbart, men igjen er dette ikke forventet. Du kan bruke java.concurrent-biblioteket.

I den versjonen av programmet som du nå skal lage, skal en turgåer sette seg ned og hvile litt i hvert kryss; hvilen er 1, 2, 3, 4 eller 5 minutter valgt tilfeldig. I et kryss er det et konstant antall sitteplasser (for eksempel 4, definert av en static konstant ANTALL_SETER i Kryss) og antall ledige sitteplasser modelleres bare med et antall. Når turgåere nå kommer til et kryss, må du sørge for at bare én om gangen får sjekke om det er en ledig sitteplass. Lag en ny metode i Kryss som heter turgaaerKommer(), hvor turgåeren reserverer en sitteplass for å hvile. Er det ingen ledige sitteplasser, må turgåeren vente i metoden.
Når turgåeren har fått seg en sitteplass, returnerer metoden til run()-metoden og turgåeren setter seg til å hvile med Thread.sleep().
Når hvileperioden er over, kaller turgåeren på nytt en metode i krysset for å gi fra seg sitteplassen igjen. Så bestemmer den (på samme måte som i oppgave 1 og 2) hvilken sti den skal ta videre og hvilket kryss den da kommer til. Tråden (dvs turgåeren) utfører så Thread.sleep() igjen og sover proporsjonalt med så lenge det tar å gå til neste kryss. Hvis tiden det tar å gå til neste kryss for eksempel er 20 minutter, kan programmet si Thread.sleep(20 * 1000).

Tråden avsluttes når den simulerte turen har vart i 2 timer.

Oppgave 3 15%

Skriv det fullstendige programmet som kjører turgåerene som tråder.
Slutt oppgave 3

Levering av oppgave 3: Lever en zip-fil med samtlige filer som kreves for at main-metoden i TestTradSimulator kan fungere. Zip-filen skal hete Oppgave3.zip.

Oppgave 4: Unngå at skogen inneholder rundturer 10%

Ta utgangspunkt i klassene du skrev i oppgave 1 (lag en kopi). Du skal nå unngå at strukturen du bygger opp av stier og kryss inneholder kombinasjoner av stier som kan ta en turgåer tilbake til et tidligere kryss, dvs at det skal ikke være mulig å gå rundturer i skogen. Dette skal gjøres ved hjelp av en rekursiv metode som sjekker om det allerede finnes en vei mellom to kryss når disse er trukket for en ny sti.

Du skal skrive en rekursiv metode i klassen Kryss: boolean finnesVeiTil(Kryss kryss2). Som før skal du i konstruktøren til Skog for hver sti som opprettes trekke to tilfeldige kryss: kryss1 og kryss2. Så kaller du finnesVeiTil() i det ene krysset med det andre krysset som parameter. Dersom det allerede finnes en vei av stier fra kryss1 til kryss2, skal du ikke legge inn den nye stien mellom disse kryssene (for da ville du fått en rundtur), men i stedet skal du trekke et nytt par av kryss og prøver disse. Om du ikke har funnet to tilfredsstillende kryss etter 100 forsøk, gir du opp og avslutter hele programmet med en feilmelding.

De to kryssene som er i hver sin ende av en sti, kaller vi *nabokryss*. Om kryss1 kaller den Boolske metoden finnesVeiTil(kryss2) i seg selv (this), skal metoden returnere true om kryss2 er det samme krysset som kryss1, eller om det for noen av nabokryssene til kryss1 finnes en vei til kryss2. Ellers returnerer den false.

Pass på at et kryss ikke spør den naboen som spørsmålet kom fra, for da vil rekursjonen ikke stoppe. Du kan for eksempel ha enda en parameter til i metoden finnesVeiTil() for å unngå dette.

Merk at hvis du kjører en simulering med en slik skog uten rundturer er det allikevel OK at en turgåer går ut fra et kryss den samme veien vedkommende kom fra.

Oppgave 4 vekt 10%

Skriv metoden finnesVeiTil() i klassen Kryss.
Gjør endringer som beskrevet i konstruktøren til Skog
Slutt oppgave 4

Levering av oppgave 4: Lever en zip-fil med de modifiserte klassene Skog og Kryss. Om du har gjort endringer i andre klasser for å løse oppgave 4, lever disse også, med kommentarer som viser hva som er endret. Zip-filen skal hete Oppgave4.zip.