Inlämningsuppgift 4: Simulering av köer på en väg med svängfil

Moment: Centrala begrepp som abstraktioner med klasser, objekt, metoder, attribut. Även arrayer, viss simulering och dokumentation med Javadoc.

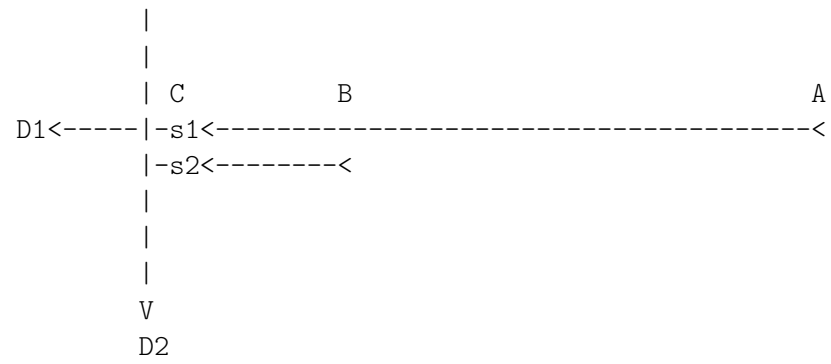
Lämplig start: 2012-10-28.

Redovisningen skall innehålla programkod i enlighet med den angivna stilguiden för Java (se kurswebben), klasser och metoder dokumenterade med javadoc, statistikutskrifter samt utskrifter som illustrerar hur trafiksituationen utvecklas. Illustrationen skall göras med hjälp av `toString`-metoderna i respektive klasser. Som frivillig utvidgning kan man göra illustationen med hjälp av grafik.

Eran redovisning skall även inehålla relevanta enhetstester med JUnit som vid redovisningstillfället både skall kunna visas upp samt köras. För enklare tester räcker det med att ni använder `assertTrue(boolean)` som testar om dess argument är true eller false och därmed avgörs om testet lyckats eller inte. Några av funktionerna kan kräva lite mer avancerade tester, och i en del fall mer än ett test per funktion. I dessa fall skall ni ange ett felmeddelande med `fail(String)`, vilket kommer att synas i den lista med fel som JUnit ger. Kom ihåg att era tester skall ge den information som behövs om något blir fel.

Problembeskrivning

Gatukontoret i Uppsala vill studera vad som händer på en väg som får en svängfil framför en gatukorsning. Betrakta följande schematiska bild:



Bilar anländer vid punkten A. Bilar kan antingen vara på väg rakt fram mot D_1 eller, med en vänstersväng, mot D_2. Bilar som skall till punkten D_2 skall ta svängfilen som börjar vid B och går fram till korsningen C. Vid korsningen C finns två ljussignaler s_1 och s_2 som kontrollerar filen rakt fram mot D_1 respektive svängfilen mot D_2. Signalerna har alltid samma period men de kan ha olika långa gröntider.

Man vill kunna studera vad som händer för olika grönperioder, ankomstintensiteter och destinationer (dvs andelen bilar som skall till D_1 repektive D_2).

Obs! *I implementationen representerar vi för enkelhets skull de två filerna som tre objekt – ett för avståndet A–B, ett för avståndet B–C samt ytterligare ett för svängfilen samma avstånd.*

Exempel i praktiken Dag Hammarskjölds väg från rondellen utanför Polacksbacken (A) söderut till korsningen med Vårdsätravägen Kungsängsleden. Sträckan tar cirka cirka 40 sekunder att köra. Den 1 befintliga svängfilen rymmer ca 10 bilar. (Denna vägsträcka har ytterligare en fil men den glömmer vi bort.)

Genomförande

Simuleringen görs genom att tiden indelas i ett antal diskreta steg. Ett tidssteg är av storleksordningen en sekund. Vid varje tidssteg kan en eller flera av följande saker inträffa:

- trafiksignaler kan byta färg,
- en bil kan passera korsningen C,
- en bil kan avancera ett stycke på vägen,
- en bil kan byta till svängfilen,
- en bil kan anlända till systemet (position A),

En oönskad situation är att kön framför någon av signalerna blir så lång att den hindrar bilar som skall till den andra destinationen (t ex att kön från svängfilen växer in i A–B). Frågan är alltså hur lång svängfilen skall göras och långa grönperioder man behöver för att detta inte skall inträffa ”alltför ofta”.

Man vill också få statistik på hur lång tid det i genomsnitt och maximalt tar för en bil att ta sig igenom systemet.

Eftersom en matematisk analys av denna situation inte är helt enkel och praktiska försök med olika längd på svängfilen är dyra och svåra att genomföra så är detta ett problem som väl lämpar sig för att studera i en datormodell.

Ni skall följa den design som finns bifogad. Innan ni läser den är det dock en god idé att själv fundera igenom vilka klasser som behövs och vad de skall innehålla.

Tips

1. Börja med att kopiera design-dokumentet och splittra det så att varje klass kommer på en egen fil.
2. Förse klasserna med `toString`-metoder från början. En lämplig stil för `toString`-metoder är

   ```
   public string toString() {
     return "ClassName(var1=" + this.var1 + ",
                    var2=" + this.var2 + ",
                    ...
                    varN=" + this.varN + ")";
   }
   ```

   Detta gör det lättare att läsa spårutskrifter.
3. Implementera och testa de enkla klasserna `Car` och `Light` först. Skriv sedan klassen `Lane` och testa den med en `toString`-metod som illustrerar i vilka positioner det finns bilar i varje tidssteg.

4. Genom att implementera tester för metoder och klasser redan innan dessa är skriva kan ni få omedelbar återkoppling på om eran kod fungerar.
5. Bygg och testa sedan succesivt upp mer och mer komplicerade system. Avslutningsvis läggs statistiken in.
6. Inför varje simulering så är det ett antal parametrar som skall sättas (ankomstintensitet, period, grönperiod, väglängder . . .). Ett enkelt och trevligt sätt att hantera detta är utnyttja klassen Properties som kan läsa in sådana värden från en fil och lagra dem i en hashtabell. Se javadokumentationen!