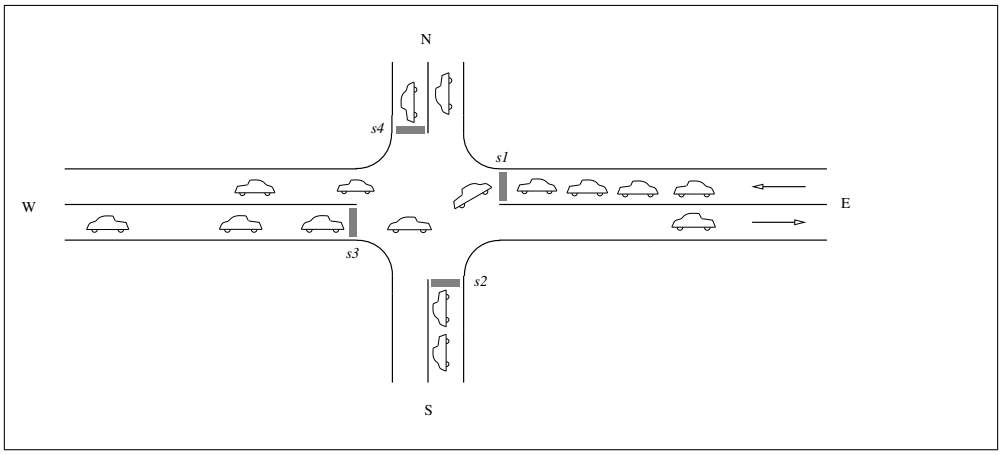
Obligatorisk uppgift: Simulering av köer i ett trafiksystem

Moment: Centrala begrepp som klasser, objekt, metoder, attribut.

Problembeskrivning

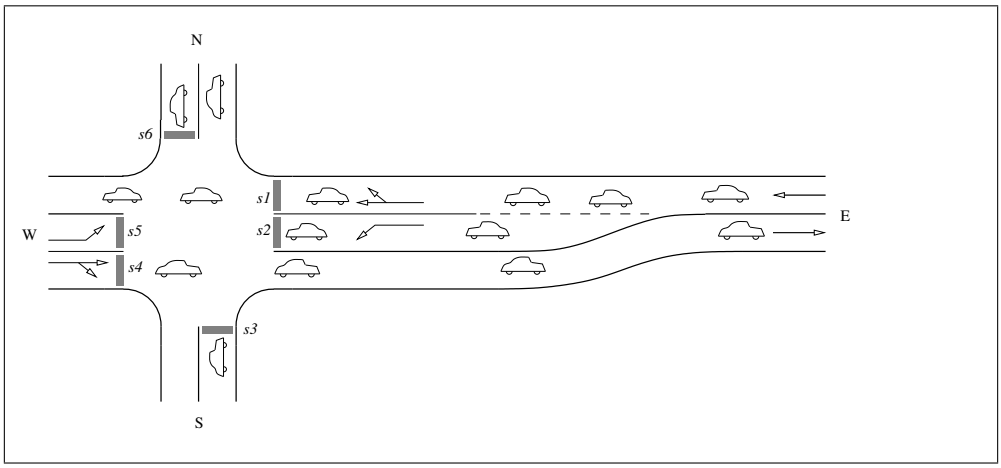
OBS: Uppgifterna kommer att diskuteras på föreläsningstid.

Gatukontoret i Uppsala har konstaterat att det blir mycket köer i en korsning av följande typ:



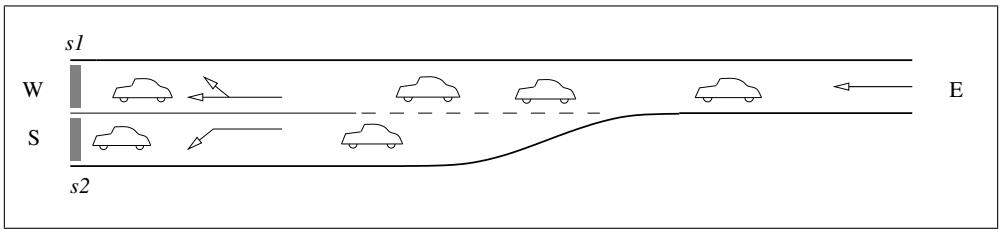
Korsningen kontrolleras alltså av de fyra ljussignaler *s1*, *s2*, *s3* och *s4*.

En betydande mängd av trafiken som kommer från `E` till höger i figuren vill svänga vänster mot `S` i korsningen och deras framart är ofta blockerad av fordon som kommer från `W`. För att lösa problemet avser man att bygga svängfiler för fordon som kommer från `E` respektive `W` och som skall vänster i korsningen dvs man vill ha en korsning av detta utseende:



Denna korsning har alltså svängfiler kontrollerade av signalerna *s2* och *s5*.

För att avgöra hur långa svängfilerna skall göras vill gatukontoret ha ett program som simulerar vad som händer i korsningen vid olika längder på svängfilen, olika trafikintensitet och olika ljusintervall på signalerna. För detta ändamål räcker det med att studera trafikflödet i en riktning dvs i ett system med följande utseende:



Eftersom trafik från `E` både mot `N` och `W` kontrolleras av samma signal (*s1*) behöver vi inte särskilja dessa utan betrakta denna trafik som gående mot `W`. Vi behöver inte heller följa fordonen när de har passerat ljussignalerna.

Exempel i praktiken

Dag Hammarskjölds väg från rondellen utanför Polacksbacken (`E`) söderut till korsningen med Vårdsätravägen – Kungsängsleden. Sträckan tar cirka cirka 40 sekunder att köra. Den befintliga svängfilen rymmer ca 10 bilar. Det finns ytterligare en fil i verkligheten men den ”abstraherar vi bort”. (Väderstrecken stämmer inte heller.)

Datormodell

Programmet skall ha följande klasser:

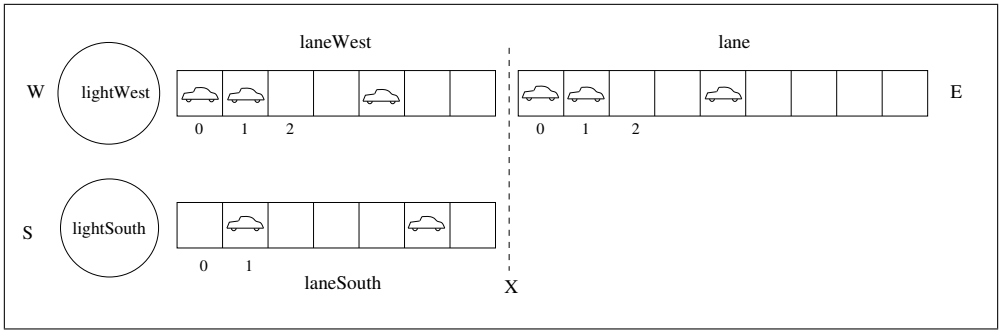
- `Vehicle` Ett fordon
- `Light` En trafiksignal
- `Lane` En fil
- `TrafficSystem` Ett trafiksystem med filer och signaler
- `Simulation` Innehåller en globalt tillgänglig klocka (`Simulation.getTime()`) och en `main`-metod som driver simuleringen.
- `VehicleGenerator` Producerar fordon enligt givna sannolikheter.

Klasserna `Simulation` och `VehicleGenerator` är givna och skall användas.

Klasserna `Vehicle`, `Light` och `Lane` skall vara utformade så att de kan sättas samman till andra trafiksystem än de som beskrivs i denna uppgift. Se specifikationen av klasserna.

Använd en *array* för att representera ett `Lane`-objekt. Varje plats antingen är tom eller rymmer ett fordon (alla fordon betraktas alltså som lika stora).

Trafiksystemet ska bestå av tre filer (`lane`, `laneWest` och `laneSouth`) och ljussignalerna `lightWest` och `lightSouth` (tidigare kallade *s1* och *s2*):



Filerna har en fix längd dvs de rymmer högst ett angivet antal fordon. Vid punkten `E` finns det en *kö* (ej utritad) som lämpligen implementeras med hjälp av en `ArrayList`. Se kursens minilektion om ArrayList och/eller Javadoc om klassen ArrayList. Längden på `lane` skall dock vara så stor att kön för det mesta är tom. Om kön växer i längd betyder det att systemet överhuvud taget inte kan svälja den angivna trafikvolymen.

Vid ett tidssteg *kan* en eller flera av följande saker hända:

- ett fordon passerar en signal (om den är grön),
- ett fordon avancerar ett steg i en fil (om platsen framför är ledig)
- ett fordon omedelbart framför `X` (dvs i position `0` på filen `lane`) flyttas till `laneWest` eller `laneSouth` beroende på dess destination (`W` eller `S`),
- ett fordon anländer till systemet vid punkten `E` och ställs i kön,
- om sista platsen i `lane` är ledig och det finns fordon i kön tas första från kön till sista platsen i `lane`,
- den ena eller båda signalerna skiftar färg.

En simulering består alltså av en tidsstegning där ovanstående saker händer. Det är lämpligt att momenten, precis som i verkligheten, utförs i den ordning de står i ovan.

Ljussignalernas funktion

En signal är grön eller röd (ingen gul färg behövs). Signaler karakteriseras av två parametar:

- en *period* dvs antalet tidssteg det är från början av ett grönt intervall till början av nästa gröna intervall och
- en *grönperiod* dvs antalet tidssteg som den är grön.

Dessa parametrar ges när signalen byggs (dvs som parametrar till konstruktorn).

En signal behöver också en intern *klocka* som tickas upp av en *step*-metod. Det är lämpligt (men inte nödvändigt) att låta klockan gå cirkulärt dvs när den kommit till sista klockslaget i perioden börjar den om från 0.

Det behövs också en metod som avgör om signalen är grön eller ej.

De båda signalerna skall ha *samma* period och starta som gröna.

Exempel: Om perioden är 7 och grönperioden för `lightWest` är 3 och för `lightSouth` är 2

så visar nedanstående tabell hur den interna klockan skall ticka och vilken färg de skall ha

Tidssteg:	0	1	2	3	4	5	6	7	8	9	10	...
Intern klocka:	0	1	2	3	4	5	6	0	1	2	3	...
Färg `lightWest`:	G	G	G	R	R	R	R	G	G	G	R	...
Färg `lightSouth`:	G	G	R	R	R	R	R	G	G	R	R	...

Indata till en simulering

Programmet styrs av följande indata:

- längderna på filerna,
- ljussignalernas karakteristik (period och grönperiod),
- ankomstintensitet dvs sannolikheten att ett fordon dyker upp vi `E` vid ett tidssteg och
- sannolikheten att ett skapat fordon skall svänga dvs ha `S` som destination.

De två sista värdena (ankomstintensitet och sannolikhet att svänga) varierar med tiden. Detta hanteras hanteras av den givna klassen `VehicleGenerator` som läser informationen från en fil vars namn ges som parameter till konstruktorn. Exempel på filinnehåll:

```
30   0.2   0.3   Night
10   0.8   0.8   Morning rush rush
30   0.5   0.5   Day
10   0.7   0.6   Afternoon rush
20   0.6   0.4   Evening
```

Se dokumentationen för klassen `VehicleGenerator` för betydelsen! Eftersom klassen är given behöver ni inte själva skriva koden som läser filen men läs igenom konstruktorn för att se hur den gör det!

De två första punkterna, dvs längden på filerna och ljussignalernas karakteristik, definieras också på en fil enligt följande exempel:

```
laneLength     : 20
laneWSLength   :  8
lightPeriod    : 14
lightWestGreen :  6
lightSouthGreen:  4
```

Dessa rader sätter längden av den första filen (20), längden av filerna framför signalerna (8), signalernas period (14) och signalernas gröntid (6 respektive 4).

Observera att raderna kan ligga i godtycklig ordning!

För att läsa denna fil ska Javaklassen `Properties` användas där metoden `load` gör uppgiften enkel. Se minilektionen om den klassen och/eller javadokumentationen

Resultat av en körning

1. Initial utskrift av simuleringsparametrarna genom metoden `printSetup`. Exempel:

```
Simulation parameters:
     laneLength     : 20
     laneWSLength   :  8
     lightPeriod    : 14
     lightSouthGreen:  4
     lightWestGreen :  6

Traffic periods and probabilities:
     30   0.2   0.3   Night
     10   0.8   0.8   Morning rush rush
     30   0.5   0.5   Day
     10   0.7   0.6   Afternoon rush
     20   0.6   0.4   Evening
```

2. Statistik innehållande

   a) genomsnittliga och maximala tider (antal tidssteg) för fordon att passera ljussignalen `lightWest` respektive `lightSouth`,

   b) andel tidssteg som kön framför vardera signalen varit längre än längden på `laneWest` och `laneSouth` dvs den tid som fildelningen vid `X` varit blockerad av kö samt

   c) andel tidssteg som det funnits fordon i kön vid entrypunkten (`E`).

   Samla tiderna för fordon som lämnar systemet i två `Measurements`-objekt (nätlektion 7), ett för vardera utgång, och använd dessa för att ta fram statistiken till punkt a). (Komplettera klassen `Measurements` med en metod för att returnera maxvärdet om du inte redan har gjort det.)

   Statistiken skall skrivas ut av metoden `printStatistics` i `TrafficSystem`. Metoden skall kunna anropas när som helst under simuleringen.

   Exempel på utskrift:

```
Statistics after 1000 time steps

  Exit west
     Number:  219
     Mean   :   26.6
     Max    :   47

  Exit south
     Number:  244
     Mean   :   34.3
     Max    :   64

  Percent time step with block: 11.0
  Percent time step with queue:  5.5
```

3. En enkel ögonblicksbild av systemet vid ett visst tidssteg. Exempel:

```
(G) <WW  W  W    W> <SSW   S    WWS S W S    S SW  S>  Queue: []
(R) <SSSS   S   S >
```

   Denna utskrift skall göras av metoden `print` som anropas varje tidssteg från `Simulation`.

   Ögonblicksbilder av systemet vid de tre följande tidsstegen. Exempel:

```
(G) <W  W  W    W > <SW   S    WWS S W S    S SW  S >  Queue: []
(R) <SSSS  S   S S>
```

```
(R) <  W  W    W  > <W   S    WWS S W S    S SW  S  >  Queue: []
(R) <SSSS S   S SS>

(R) < W  W    W  W> <   S    WWS S W S    S SW  S  W>  Queue: []
(R) <SSSSS   S SS >
```

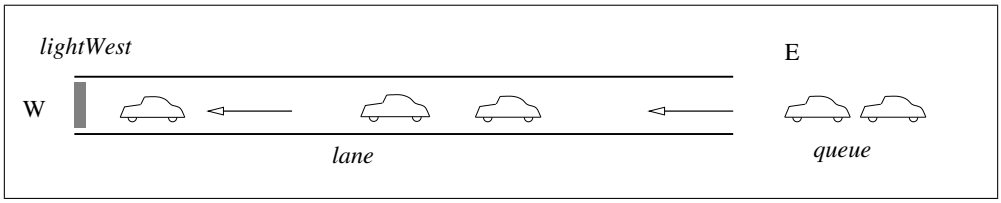
Med indata, t.ex. ljussignalernas karakteristik, så att det köar fordon längst till höger. Exempel:

```
(R) <W  WWWW WWWWW> <SWWWWS SSSWWS S W S    S SW  SW>  Queue: [SWWWS]
(R) <SSSS  SSSSSSS>
```

Lämplig arbetsgång

Implementera klasserna `Vehicle`, `Light` och `Lane`. Förse var och en av dessa med en `main`-metod som testar och demonstrerar att klasserna fungerar.

Börja med att i klassen `TrafficSystem` implementera ett enklare system bestående av EN ljussignal, EN fil och EN kö:



Använd den nedladdade klassen `VehicleGenerator` för att skapa fordon. Den nedladdade klassen `Simulation` innehåller bl a en `main`-metod som sätter upp trafiksystemet, driver simuleringen och anropar `print`-metoder. Dessa två klasser ska användas i oförändrat skick (se dock frivillig modifiering nedan).

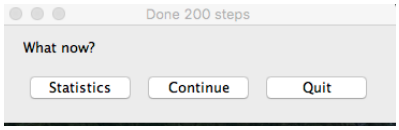
Implementera sedan systemet med en svängfil. Endast klassen `TrafficSystem` ska behöva modifieras.

För att bli godkänd på uppgiften måste

1. programmet fungera enligt specifikation i detta dokument och Javadoc specifikationen för klasserna `Light`, `Lane` och `Vehicle`,
2. koden uppfylla kodstandarden,
3. du kunna förklara hur programmet fungerar och
4. du kunna rita en figur som beskriver hur objekten hänger samman i en konkret situation.

Frivillig modifiering

Simuleringen tidstegas av `main`-metoden i den nedladdade klassen `Simulation`. Vart hundrade tidsteg skriver metoden ut statistik och frågar användaren om den ska fortsätta. Snyggare vore att i stället använda en dialogruta:



Just denna ruta är producerad av koden

```
int ans = JOptionPane.showOptionDialog(null,
                                       "What now?",
                                       "Done " + time + " steps",
                                        JOptionPane.DEFAULT_OPTION,
                                        JOptionPane.PLAIN_MESSAGE,
                                        null,
                                        alternatives,
                                        alternatives[2]);
```

där `alternatives` är definierad som

```
String[] alternatives = {"Quit", "Continue", "Statistics"};
```

Vaiabeln `ans` kommer innehålla index för valt alternativ.