Laboration 5 – Snabbköpsoptimering
Version 1
2018–03–06

1. Kort sammanfattning

Uppgiften går ut på att skapa

1. Javaklasser som bildar en generell, diskret och händelsestyrd simulator,

2. Javaklasser som kan användas för att simulera snabbköp och optimera kundflödet samt

3. Ett Javaprogram som simulerar snabbköp med hjälp av 1 och 2 ovan dels a) för att visa på hur en sådan simulering med givna parametrar sker i steg dels b) själv optimerar parametrar till en snabbköpssimulering inom givna gränser.

Arbetet ska bedrivas i grupper, där alla bidrar lika mycket även om det är på olika sätt. Att så är fallet intygar gruppen genom att i form av en javadoc-kommentar först i samtliga källkodsfiler ange alla gruppens medlemmar. Allas namn radas upp i alla filer och då även filer de inte direkt varit med att skriva.

I ett första steg ska en objektorienterad design i UML göras där det framgår vilka delar som implementationen kommer att innehålla och (i stort) hur dessa delar är tänkta att samverka för att lösa deluppgifterna. Efter att ha presenterats för lärare/labhandledare och blivit godkänd (vilket kan kräva mer än en presentation) ska designen sen implementeras. Observera att det inte är tillåtet att börja programmera innan designen är godkänd.

Implementationen ska göras i Java och vara baserad på de principer för programkonstruktion som lärs ut i kursen. Detta innebär t ex att alla klasser ligger i paket, att tillstånd och inre kod (hjälpmetoder) är dolda, och att all kod dokumenterats.

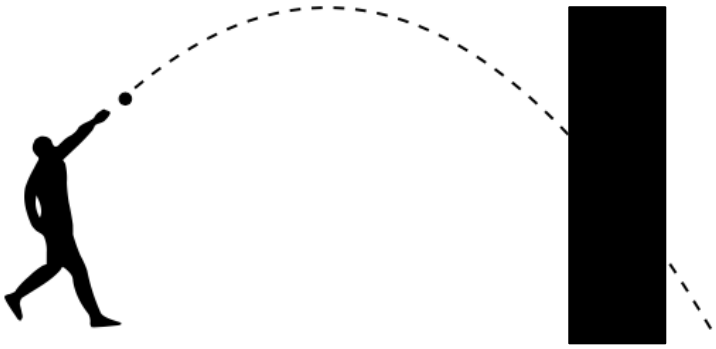
Tider och närmare anvisningar för presentationen av designen och redovisningen av den färdiga implementationen kommer att publiceras närmare slutdatum.

2. En generell, diskret och händelsestyrd simulator

Generellt uttryckt är en diskret händelsestyrd simulator ett program som gör det möjligt att följa hur ett tillstånd, utsatt för förändringar orsakade av händelser, förändras över tid. Sådana simuleringar används för att studera komplicerade förlopp som är svåra eller rent av omöjliga att analysera matematiskt.

Simuleringen startar med ett initialtillstånd och en mängd första händelser som säkert kommer att inträffa. Beroende på vad som ”händer då händelserna inträffar” kan också ytterligare, framtida, händelser identifieras. Om en händelse inträffar som t ex innebär

att en boll kastas i riktning mot en vägg så skulle en framtida – just nu identifierad – händelse kunna vara att bollen studsar mot väggen och ändrar riktning. Exakt vad studsen då skulle få för konsekvens är något som bara kan avgöras när studsen sker. Detta är innebörden i beteckningen *händelsestyrd*: Det sätt som simuleringen fortskrider på bestäms huvudsakligen under simuleringen, inte innan. Händelser ger, när de inträffar, upphov till framtida händelser.



Att simuleringen kallas *diskret* beror på att händelserna inträffar på vissa givna tidpunkter, och endast då, och att det vanligtvis kontinuerliga förloppet som simuleras därför är diskretiserat. En utmaning med att simulera med en diskret och händelsestyrd simulator är att lyckas hitta alla väsentliga nyckelhändelser som har betydelse och som får simuleringen att efterlikna det verkliga förlopp som ska studeras. Det är, om vi återvänder till vårt exempel, sammanhanget som avgör om bollens studs behöver följas upp eller ens om händelsen att bollen kastas är väsentlig.

En annan viktig skillnad mellan simuleringen och det förlopp som simuleras är att (även) *tiden är simulerad* under simuleringen. Den beräknas med en formel och med hänsyn till det aktuella tillståndet. Det ögonblick i simuleringen då bollen i exemplet ovan kastas är ett beräknat tal. Det är inte riktigt tid. Vi kallar det ”tid” bara för att det används som tid och gör det möjligt att ordna händelser inbördes. Genom att formulera bollbanan som en kastparabel (kanske med hänsyn till vindstyrka, riktning, hastighet och detaljer ur tillståndet samt kanske också slumpen) kan vi beräkna vilken ”tid” studsen sker, vilket, eftersom det sker efter kastet, är ett tal större än motsvarande tal beräknat för kasthändelsen.

Att tiden endast är tal för inbördes ordning av händelser gör också att simuleringen kan skrida fram oberoende av riktig tid. Den kan omedelbart gå från en ”tidpunkt” till en annan utan att vänta in att datorns verkliga klocka verkligen tickar på från ena tidpunkten till nästa. Med andra ord kan simuleringen av en lång process löpa på blixtsnabbt. I simuleringen av bollkastet behöver inte simulatorn invänta att bollen når väggen utan utsätter tillståndet för effekten av studsen omedelbart efter att kasthändelsen hanterats, om nu inte nåt annat först ska hända förstås.

3. Simulatordelarna i deluppgift 1

En implementering av en generell simulator består av delar som arbetar ihop och beror av varandra under en simulering. Poängen med att göra generella delar är att dessa sen kan utökas – med arv – till specifika simuleringar, som blir enklare att skriva då de kan återanvända de generella delarnas kod.

Vår generella simulator behöver till att börja med en klass `State` för ett generellt tillstånd. Ett dynamiskt objekt av denna klass används för att hålla reda på generella detaljer alla möjliga simuleringar har gemensamma t ex aktuell uträknad tid [dit simuleringen så att säga hunnit] och en nödbroms i form av en boolesk flagga som när den sätts till sann stoppar simuleringen. Tanken är att specifika simuleringar ska utöka denna tillståndsklass med arv och lägga till alla detaljer som är väsentliga för just den simuleringen. Tillståndet är observerbart och ska kunna trigga anrop till observatörer varje gång en händelse inträffar.

Vi behöver också en klass `Event` för en generell händelse som är tänkt att ärvas av specifika händelser i specifika simuleringar. En händelse har en effekt och en uträknad tid den inträffar. Effekten är kodad som en metod som behöver tillgång till tillståndet (för att kunna påverka det). En händelse bär alltså med sig den effekt den har som en metod vi skulle kunna kalla ”utför mig”. Observera skillnaden mellan att skapa en händelse (ett dynamiskt objekt) och att trigga händelsens effekt på tillståndet genom att anropa metoden i objektet och därmed be händelsen att ”utföra sig” och ändra tillståndet. Av generella händelser i en simulering finns egentligen bara två: Att starta och avsluta simuleringen, vilket i det senare fallet görs genom att använda nödbromsen.

Framtida händelser lagras i en händelsekö implementerad som klassen `EventQueue`. Denna kö ordnar instoppade händelser efter beräknad tid och ger, på uppmaning, ut den händelse som står på tur. Notera att olika sorters händelser inte säkert ska inträffa i den ordning händelseobjekten skapas och stoppas in. Kön måste aktivt ordna händelserna. Eftersom en händelse kan ge upphov till nya händelser, måste klassen Event ha en referens till kön så dessa nya, framtida, händelser kan lagras med övriga framtida händelser.

Den klass som ser till att simuleringen fortskrider är `Simulator`. Denna innehåller i princip bara en metod `run` med en loop som plockar ut den händelse som står på tur ur händelsekön och ser till att händelsen inträffar genom att anropa händelsens metod ”utför mig”. Själva kärnan i simuleringen finns här: Varje utplockad händelse påverkar, när den inträffar, tillståndet men kan också ge upphov till framtida händelser som dess ”utför mig”-metod då stoppar in i händelsekön. Det är denna generering av nya händelser som håller igång simuleringen. Observera att nya händelser bara kan skapas i ”utför mig”-metoderna i händelser som inträffar. Loopen i simulatorns metod slutar när nödbromsen aktiverats.

Den sista delen är en klass `View` som är en generell observatör av `State`. Som observatör har den en metod update som dock är tom. Poängen är att View ska ärvas till en specifik vy som kan följa simuleringen i steg och synliggöra vad som händer med tillståndet.

4. En speciell simulering: Snabbköpet

Baserat på den generella simulators delar ska vi nu skapa en simulator för ett snabbköp. En sådan simulering ska visa vad som händer i snabbköpet givet vissa bestämda omständigheter och därmed ge beslutsunderlag till ledningen. Intressanta frågor kan t

ex vara:

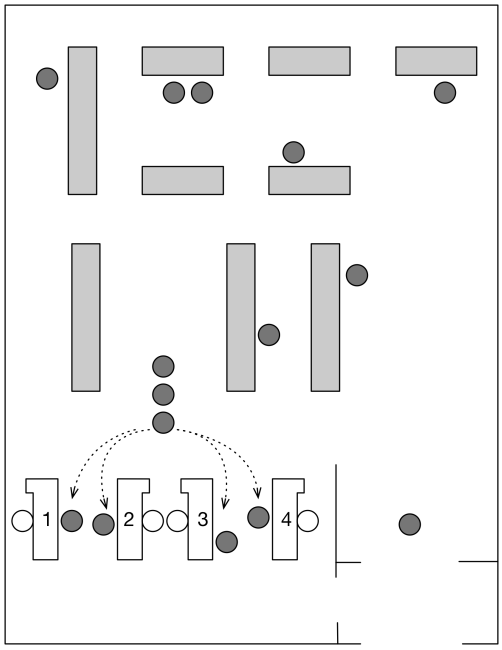
- Exakt hur påverkar t ex antal öppna kassor kundflödet genom snabbköpet? [Fler öppna kassor ökar förstås, allt annat lika, kundgenomströmningen men hur?]

- Exakt hur påverkas den genomsnittliga kötiden av antal kassor och hur skicklig kassapersonalen är?

- Hur ska bemanningen av kassorna vara för att maximera dagsförtjänsten?

I stort gör en kund följande: Anländer och plockar ihop sina varor, får varorna scannade och betalar (eventuellt efter att först ha behövt köa en stund) och lämnar snabbköpet med sina varor. Dessa aktiviteter kommer, som vi snart ska bli varse, att bilda grunden för den mängd händelser som styr simuleringen.

Bilden nedan illustrerar ett snabbköp med 4 kassor, alla upptagna, och 15 kunder (grå cirkelskivor) varav 4 betalar, 3 köar och övriga 8 plockar varor. En av de 15 har just kommit in i snabbköpet. Det streckade pilarna visar var den förste i den gemensamma kön kan komma att få gå för att betala så snart en kassa bli ledig.



Centralt för simuleringen är hur vi modellerar utrustning i snabbköpet och beteenden hos kunder och personal. I vårt fall definieras simuleringen av följande parametrar.

1. Antal kassor *N* som är öppna. Varje öppen kassa är antingen verksam, för att en kund håller på att få varorna scannade vid den och betalar, eller overksam.

2. Maximalt antal kunder *M* som samtidigt får vara inne i snabbköpet. Av brandskäl och av rent praktiska skäl finns alltid en övre gräns.

3. Ankomsttider för kunderna. Kunder anses anlända slumpmässigt och oberoende av varandra. Ett lämpligt antagande[1] är att skillnaden mellan ankomsttiderna är exponentialfördelade med parameter λ. Parametern anger det snittantal kunder som anländer per tidsenhet. Om t ex λ=4 kunder anländer per timme (tidsenhet), är den förväntade tiden mellan på varandra följande kunder ¼ timmar (15 minuter).

4. Tid det tar att få varor scannade och sen betala. Egentligen borde denna tid bero av antal (och typ av) varor samt hur effektiv kassapersonalen är, men vi förenklar och betraktar istället tiden det tar för en genomsnittlig kund. Kassatiderna antas vara likformigt fördelade inom ett intervall $[K_{\min} .. K_{\max}]$ så den förväntade kassatiden är ~~$(K_{\min} - K_{\max})/2$~~ $(K_{\min} + K_{\max})/2$.

5. Tid det tar för en kund att plocka ihop varorna. Även denna tid varierar slumpmässigt med bl a antal och typ på varorna men återigen förenklar vi och betraktar en genomsnittlig kund. Plocktiderna antas vara likformigt fördelade inom ett intervall $[P_{\min} .. P_{\max}]$ så den förväntade plocktiden är ~~$(P_{\min} - P_{\max})/2$~~ $(P_{\min} + P_{\max})/2$.

6. Snabbköpets öppettid. Snabbköpet öppnar vid beräknad tid 0 och stänger vid beräknad tid S.

7. En heltalskonstant f som är fröet (startvärdet) till alla slumptalsgeneratorer. Genom att f är en parameter kan vi köra om samma simulering flera gånger och få samma resultat.

Tillståndet under simuleringen ska bland annat innehålla allt vars utveckling över tiden vi vill studera. Detta inkluderar:

- Antal kunder som genomfört köp av varor. *Detta vill man maximera för att öka på förtjänsten.*

- Antal missade kunder som anlänt men inte kommit in, pga att det redan fanns maximalt med kunder i snabbköpet, och som därför avvikit (åkt till ett konkurrerande snabbköp). *Bäst är förstås om alla som tar sig till snabbköpet också handlar där, vilket kräver att kundflödet genom och ut matchar tillflödet.*

- Tid som kassorna stått overksamma. *Detta motsvarar en ren kostnad man i regel vill minimera.*

- Tid som kunder behövt stå i kassakö. *Långa kötider gör kunder missnöjda och leder på sikt till färre kunder.*

5. Specifika delar

[1] Våra tidsmodeller är medvetet förenklade. Att ta fram verklighetstrogna modeller och simuleringar kräver djup kunskap om statistik och köteori.

Till vår snabbköpssimulator behöver vi klasser för tillstånd, händelser och vy. Själva simulatorns grundfunktionalitet – att lagra händelser och se till att de inträffar i tidsordning – finns i den generella simulatorn. De specifika delarna ärver, men hålls annars helt skilt från, de generella. Faktum är att det räcker med att vi skapar händelserna, tillståndet och vyn för att vi sedan ska kunna skriva ett huvudprogram som sätter upp, startar och följer simuleringar.

5.1. Händelser

Nedan listas de händelsesorter som finns och vilken deras effekt är. Varje specifik händelseklass ärver rimligen direkt, eller indirekt (om det finns gemensamma delar i underklasserna som kan flyttas upp i en mellanklass), `Event`.

A) Simuleringen startar (*starthändelse*)

Denna klass får ärva den generella starthändelsen, sättas till att inträffa tiden 0 och ha som enda effekt att en första ankomsthändelse för en ny kund läggs till händelsekön.

B) En kund ankommer till snabbköpet (*ankomsthändelse*)

En kund C som anländer släpps in endast om snabbköpet ännu är öppet och det finns plats. Släpps C in är det en till kund inne i snabbköpet som kommer att handa varor.

Om det är öppet men fullt så avviker C istället utan att handla och har vi missat en kund. En kund som anländer efter stängning ska däremot inte räknas som en missad kund.

Först ska C plocka ihop sina varor. Vi skapar en framtida plockhändelse för C (se nedan) som inträffar när C plockat klart och gått bort till kassorna. Denna läggs till händelsekön.

Under förutsättning att snabbköpet har öppet ska slutligen också en (1) ny framtida ankomsthändelse (för en ny kund) skapas och lagras i händelsekön.

Att varje ankomsthändelse genererar nästa gör att vi dels vet att det alltid finns en ankomsthändelse i kön dels minskar behovet av minne som kön behöver. Sämre vore att generera alla framtida ankomsthändelser innan simuleringen startar, något som alltså **inte** ska göras.

C) Kunden har plockat sina varor (*plockhändelse*)

Denna händelsesort motsvarar att en kund C plockat alla sina varor, gått bort till kassorna och är klar att betala.

Om det finns lediga kassor så går C omedelbart till en av dem och scanning/betalning inleds. Antalet lediga kassor minskar med 1 och vi genererar en framtida betalningshändelse för C som läggs till händelsekön.

Om alla kassor däremot är upptagna ställs C istället i kassakö. Det finns endast en gemensam kassakö för alla kassor (inte en per kassa).

D) Kunden har betalat sina varor och lämnat snabbköpet (*betalningshändelse*).

När detta inträffar minskar antalet kunder i snabbköpet med 1 och vi kan anteckna att ytterligare en kund har handlat. Samtidigt som kunden går blir en kassa ledig.

Om kassakön inte är tom får den kund C' som stått längst i kö (den som i normala fall står först) gå till den lediga kassan som därefter omedelbart åter blir upptagen. Vi genererar sen en betalningshändelse för C'.

Om kassakön istället är tom ökar vi på antalet lediga kassor med 1.

E) Snabbköpet stänger för dagen (*stängningshändelse*)

Denna händelse involverar inte någon speciell kund. Den innebär att kunder som anländer till snabbköpet senare inte släpps in. Observera att sådana, sena, kunder inte ska räknas som missade.

Däremot betjänas alla kunder som redan är i snabbköpet klart, trots att snabbköpet alltså stängt ytterdörrarna och inte släpper in fler kunder.

All aktivitet i snabbköpet avbryts när den sista kunden betalat och lämnat snabbköpet. Observera att som simuleringen görs kan en kund ankomma efter stängning och efter att alla kunder betalat och lämnat snabbköpet. I detta fall ska tiden från sist betalande till denna sista ankomst **inte** räknas med då summa ledig kassatid beräknas. Det bifogade simuleringsexempel 2 visar hur tiden ska räknas i detta fall (se näst sista raden i förloppsavsnittet).

F) Simuleringen avslutas (stophändelse)

Denna händelse ärver den generella stophändelsen som avslutar simuleringen. Om den generella stopphändelsen inte är abstrakt går det också bra att skippa denna händelse.

5.2. Tillstånd

I beskrivningarna av händelserna finns detaljer och fakta som förändras under simuleringen och som därför ska sparas i ett specifikt tillstånd, som implementeras som en klass som ärver det generella tillståndet. T ex kan man notera att eftersom alla kassor är lika varandra så räcker det med heltal för att hålla reda på hur många de är och hur många som är lediga. Vi behöver inga objekt för kassor.

Dessutom passar det att tillståndsobjektet håller reda på statistik som t ex antal kunder som handlat, antal som behövt stå i kö, hur många kunder som missats samt summa tid kassor varit lediga och kunder fått vänta i kassakö. Notera att tiden kassor varit lediga slutar att räknas upp efter att sista kunden betalat och lämnat snabbköpet även om det senare ankommer en kund som inte släpps in. Vi slutar räkna upp tiden efter att sista kunden betalat.

Till simuleringen behövs även tidsberäkningar, en kassakö och möjlighet att skapa kunder.

A) Tidsberäkningar

3 olika slags tider behöver beräknas för ankomsthändelser, plockhändelser respektive betalningshändelser. Skapa klasser för det och ha ett objekt, en slags källa ur vilken tider kan hämtas, av varje sort i tillståndet. Varje tidskälla ger absoluta tider, dvs nuvarande tid plus ett delta så det blir en framtida tid. Sen använder händelseklasserna dessa tidskällor när nya händelser skapas.

Till denna laboration finns Javaklasser givna som genererar slumptal, både likformigt fördelade och exponentialfördelade, och som ska användas för beräkningarna av tid. De ska kapslas in i egna klasser.

B) Kassakö

Denna fungerar som en FIFO-kö (se lab 3) och sparas även den lämpligen i tillståndet.

C) Kunder

Kunder identifieras unikt av heltal 0, 1, 2, 3, … som utgör deras *kundnummer* men de är inte heltal. I tillståndet ska finnas en kundkälla från vilket nya kunder vid behov kan hämtas. Numreringen ska ske automatiskt.

5.3. Vy

En specifik vy över simuleringen skriver ut en rad med detaljer ur tillståndet *varje gång* en händelse inträffar men *före* händelsen haft effekt på tillståndet. Utskriften består av tre avsnitt med 1) parametrarna, 2) förloppet en händelse i taget och 3) resultat. Exempel som visar hur utskrifterna ska se ut när den första typen av huvudprogram (RunSim.java) körs för några olika val av parametrar finns bifogade sist. Följande gäller generellt.

1. Det första som händer är en starthändelse. I ett första avsnitt skrivs alla parametrar till simuleringen ut. Sen skrivs också rubriken på andra avsnittet, kolumnrubrikerna på andra avsnittet samt tiden 0 och namnet på starthändelsen.

2. Förloppsavsnittet består av rader. För varje händelse annan än starthändelsen, stängningshändelsen och stophändelsen skrivs, i denna ordning, en rad med följande ut i konsolen: 1) tid, 2) händelsenamn, 3) kundnummer, ~~4) summa tid kassor varit lediga, 5) summa tid kunder stått i kundkön, 6) antal kunder inne i snabbköpet, 7) antal kunder som handlat (dvs kommit in, plockat, betalat och lämnat), 8) antal missade kunder, 9) antal kunder som köat, 10) kassaköns längd och 11) hela kassakön, där den för varje par av kunder gäller att den som står till vänster har stått längre i kön än den som står till höger.~~ 4) om det är öppet eller stängt, 5) antal lediga kassor, 6) summa tid kassor varit lediga, 7) antal kunder inne i snabbköpet, 8) antal kunder som handlat (dvs kommit in, plockat, betalat och lämnat), 9) antal missade kunder, 10) antal kunder som köat, 11) summa tid kunder stått i kundkön, 12) kassaköns längd och 13) hela kassakön, där den för varje par av kunder gäller att den som står till vänster har stått längre i kön än

den som står till höger.

Notera att en ankomst efter stängning och som är den sista händelsen innan stophändelsen inte ska påverka summa tid kassor varit lediga.

3. För en stängningshändelse skrivs samma sak ut som i 2 ovan förutom kundnummer. Stängningen av snabbköpet involverar ju ingen speciell kund. Istället för kundnummer skrivs bara ett bindestreck ut på kundnumrets plats.

4. För en stophändelse skrivs först bara 1) tid och 2) händelsenamn ut. Sen följer en utskrift i tre delar med ~~av~~
    a. Hur många kunder som anlänt innan stängning och hur många av dem som missats,
    b. total och genomsnittligtid kassor varit lediga och
    c. total och genomsnittlig tid kunder köat.

En vy skapas som en klass som är en observatör och som observerar tillståndet.

5.4. Hjälp med implementeringen

I detta avsnitt finns samlat en del extra hjälp som underlättar implementeringen.

5.4a. Undvika static

Eftersom allt static är gemensamt medan simuleringar ska vara oberoende av varandra så inser man att inget, utom main-metoder, får vara static. Ett sätt att komma bort från static-variabler är att istället skapa "fabriker", dynamiska objekt som har en dynamisk variabel som används som static-variablerna.

I lab 5 har man utmaningen att sätta unika nummer 0,1,2,3... på kunderna. En lösningen med static skulle spara senast tilldelade nummer i en statisk variabel som ökades på för varje nytt nummer. Men har man då flera simuleringar igång samtidigt kommer numreringen att blir gemensam för alla(!) Det skulle bli svårförutsägbara hopp i numreringen i de enskilda simuleringarna och vi skulle kunna förledas tro att det är fel i simuleringarna. Om man istället ger varje simulering en egen "fabrik", som i praktiken använder en dold dynamisk variabel som den statiska används, blir numreringarna oberoende. "Fabriken" har en metod som levererar nästa nummer baserat på senast tilldelade nummer.

6. Huvudprogram

Själva simulatorn sätts samman i ett huvudprogram. Vi ska göra två slags huvudprogram i denna laborationsuppgift.

6.1. Programmet RunSim.java

Den första sorten genomför en (1) simulering med givna parametrar och visualiserar steg för steg hur simuleringen framskrider med utskrifter enligt avsnitt 5.3.

Programmet skapar ett starttillstånd, definierat av parametrarna, och en händelsekö i vilken läggs in en starthändelse, en stängningshändelse och en stophändelse. En vy skapas och läggs till som observatör på tillståndsobjektet. Sen skapas ett simulatorobjekt, med hjälp av tillståndet och kön, varpå dess metod `run` anropas och loopen börjar arbeta.

Testkör med samma parametrar som i exemplen och kontrollera att utskrifterna överensstämmer.

6.2. Programmet Optimize.java

Man kan själv optimera driften i snabbköpet genom att upprepade gånger manuellt köra den första sortens huvudprogram i A), variera parametrarna och jämföra resultaten. Den andra typen av huvudprogram automatiserar denna manuella hantering och hittar således optimala parametrar genom att själv variera parametrar och köra simuleringar. Här ska inga utskrifter göras under simuleringarna utan endast i slutet och av de funna optimala parametrarna. Någon vy behövs alltså inte.

Vi ska begränsa oss till att, givet alla andra parametrar, hitta det minsta antal kassor N_{opt} som ***troligen*** behövs för att antalet missade kunder ska bli så litet som möjligt. Det kan vara så att det finns ett antal kassor som gör att inga missas men det är alltså inte säkert.

Några saker som kan hjälpa med implementeringen:

- Varför ”troligen” och vad innebär det för vårt program? Jo, resultatet av en simulering [då vi fixerat alla parametrarna] beror av vilket frö slumptalsgeneratorerna får och vi kan därför inte blint lita på resultatet från en enstaka simulering. Det kan ju vara så att just det frö vi råkat välja ger ett icke representativt simuleringsresultat. För att våga lita på simuleringsresultaten måste vi därför implementera huvudprogrammet så att det kör om samma simulering många gånger med olika (förhoppningsvis orelaterade) frön, och sen gör en sammanvägd bedömning av resultaten.

  Svåra frågor att ge bestämda svar på är hur många gånger simuleringarna ska köras, och med vilka frön, samt hur ska sammanvägningen ska göras. I vårt fall tar vi det största (sämsta) antalet missade någon av simuleringarna ger. Förhoppningen är att denna pessimism i alla fall inte ska ge överoptimistiska (orealistiska) resultat.

- Vi inser att N_{opt} inte behöver vara större än M, det maximala antalet kunder i snabbköpet, för då skulle alltid minst en kassa inte användas. N_{opt} är alltså ett tal mellan 1 och M.

- En annan insikt är att om vi lägger till ännu en kassa så kan inte antalet missade kunder öka. Tvärtom, så klarar snabbköpet av att ta betalt av fler kunder per tidsenhet, vilket bara kan öka kundgenomströmningen, vilket i sin tur betyder att fler av de anländande kunderna kan släppas in. Antal missade är alltså en

avtagande funktion i antalet kassor. Detta kan användas för att göra programmet effektivare (vilket dock inte krävs).

Så här kan man göra implementeringen:

Det ska ju inte vara några utskrifter under simuleringarna så ta bort dem. OBS! Radera inte bort utskrifterna utan addera bara inte någon vy på simuleringen(!) Här märks fördelen med designmönstret Observer omedelbart (om man använt det rätt).

Skriv en metod (”metod 1”) som kör en simulering där alla parametrarna är fixerade/givna. Om vi hade kvar utskrifterna är detta i princip huvudprogrammet av den första typen (*RunSim.java*). Gör så metoden returnerar sluttillståndet.

Skriv en metod (”metod 2”) som givet alla parametrar FÖRUTOM ANTAL KASSOR, hittar minsta antal kassor. Denna metod använder upprepade gånger ”metod 1” och skickar varje gång med det antal kassor som ska användas. Alltså, dess interna funktion styrs av hur antal missade kunder i sluttillstånden förändras mellan anropen till metoden i 2. Det finns två huvudsätt att implementera jakten på det minsta antalet kassor på som båda baseras på de två insikter jag gett i specen. Metoden returnerar minsta antalet kassor.

Skriv en metod (”metod 3”) som kör ”metod 2” flera gånger med givna parametrar MEN OLIKA FRÖN VARJE GÅNG. Metoden tar alltså ett frö FRÖ som argument, har en variabel av typen *Random* som får referera till slumptalskällan *new Random(FRÖ)*, och kör en loop i vilken:

- ”Metod 2” körs med nästa slumptal ur slumptalskällan.
- Det undersöks om det minsta antal som returneras är högre än tidigare och i så fall sparas det som nytt högsta minsta antal.

Ju fler gånger loopen körs desto mer troligt att det inte finns nåt högre antal kassor som behövs för att inte missa några kunder. Vi låter därför loppandet fortgå tills högsta minsta antalet inte förändrats på "länge". Detta påminner om hur en del numeriska metoder över reella tal t ex Newton-Raphsons metod vidare, inte ett visst antal gånger, utan tills skillnaderna mellan på varandra följande beräkningar skiljer mindre än ett givet epsilon.

Istället för att välja det högsta antalet i loopen skulle man kunna räkna ut ett genomsnitt av alla returnerade antal. Att jag har valt högsta är för att det är lättast att implementera och är i vart fall inte någon glädjekalkyl.

En annan observation som kan underlätta implementeringen är att *för ett givet frö* ger lika många kassor som det maximalt ryms kunder i snabbköpet det minsta antal kunder som missas. Bättre går inte att åstadkomma (för just det fröet). Samtidigt missas maximalt med kunder om man bara har en (1) kassa (för just det fröet). Sämre går inte att åstadkomma (för just det fröet). Antal missade är därför en avtagande funktion i antalet kassor.

Men notera att denna avtagande funktion kan se olika ut för *olika frön*, allt annat lika. Det betyder att för vissa frön kan det gå att helt undvika att missa kunder medan andra frön obönhörligt medför missade kunder. Och bland de senare fröna kan det exakta antalet som säkert missas variera. Vissa kan ge lågt antal medan andra ger högre. Det är ovanstående som gör att ”metod 3” måste simulera många gånger, med olika frön, för att hitta det mest troliga (största, värsta) minsta antal kassor som minimerar antal missade kunder.

Varje simulering med ett visst frö görs av ”metod 2”. Denna metod ska alltså hitta det antal kassor, som för just det givna fröet, minimerar antalet missade. Metoden kan pröva sig fram till rätt antal kassor på två sätt:

- Programmet prövar sig fram med 1 kassa, 2, kassor, 3 kassor osv och slutar så snart antalet missade som blir resultatet av simuleringen är det bästa möjliga (minsta antalet missade alltså). Detta är ineffektivt, precis som `recRaiseOne` i lab 1, och tar lång tid för stora populära snabbköp.

- Programmet binärsöker bland antalet kassor dvs halverar i steg intervallet där det optimala antalet kassor finns tills endast ett (det optimala) antalet återstår. Detta är lika effektivt som `recRaiseHalf` i lab 1, dvs behöver bara några tiotals simuleringar för att hitta det optimala antalet i snabbköp som får ha flera *miljarder* kassor (☺!), och fungerar förstås i praktiken bra för alla snabbköp.

Simuleringarna gör ”metod 2” genom att anropa ”metod 1”, som i praktiken är RunSim.java men utan utskrifter.

Hur många simuleringar ska då ”metod 3” göra? Och hur ska resultatet från dessa simuleringar användas?

”Metod 3” ska fortsätta simulera (med olika frön) tills det största minsta antalet vi hittat inte ändrats i 100 simuleringar i rad. Ett tips är att ha en räknare för antal gånger senast hittade största minsta antal stått sig och att nollställa den varje gång ett nytt, högre, största minsta antal hittas.

100 är här tyvärr lika godtyckligt. Egentligen beror antalet simuleringar som behövs på simuleringsparametrarna, men att utreda exakt hur många är utmanande så därför nöjer vi oss med 100. Det viktiga är att det är ett antal så att det är "troligt" att svaret är korrekt.

Viktigt är att den generella simulatorn inget vet om den specifika. Den ska fungera för vilken specifik simulator som helst, inte bara en snabbköpssimulator.

Simuleringsexempel 1

```
PARAMETRAR
==========
Antal kassor, N.........: 2
Max som ryms, M.........: 5
Ankomshastighet, lambda..: 1.0
Plocktider, [P_min..Pmax]: [0.5..1.0]
Betaltider, [K_min..Kmax]: [2.0..3.0]
Frö, f..................: 1234

FÖRLOPP
=======
   Tid Händelse  Kund  ?  led    ledT    I     $    :-(   köat    köT   köar  [Kassakö..]
  0,00 Start
  0,44 Ankomst     0  Ö    2    0,87    0     0     0      0    0,00      0  []
  0,49 Ankomst     1  Ö    2    0,97    1     0     0      0    0,00      0  []
  0,64 Ankomst     2  Ö    2    1,28    2     0     0      0    0,00      0  []
  1,26 Plock       0  Ö    2    2,52    3     0     0      0    0,00      0  []
  1,42 Ankomst     3  Ö    1    2,68    3     0     0      0    0,00      0  []
  1,46 Plock       1  Ö    1    2,72    4     0     0      0    0,00      0  []
  1,57 Plock       2  Ö    0    2,72    4     0     0      0    0,00      0  []
  2,15 Plock       3  Ö    0    2,72    4     0     0      1    0,58      1  [2]
  2,51 Ankomst     4  Ö    0    2,72    4     0     0      2    1,30      2  [2, 3]
  3,18 Plock       4  Ö    0    2,72    5     0     0      2    2,64      2  [2, 3]
  3,91 Betalning   0  Ö    0    2,72    5     0     0      3    4,82      3  [2, 3, 4]
  4,10 Ankomst     5  Ö    0    2,72    4     1     0      3    5,21      2  [3, 4]
  4,41 Betalning   1  Ö    0    2,72    5     1     0      3    5,84      2  [3, 4]
  4,70 Plock       5  Ö    0    2,72    4     2     0      3    6,13      1  [4]
```

```
  6,76 Betalning    2  Ö    0    2,72    5     2     2      5   11,18      3  [4, 5, 6]
  6,87 Betalning    3  Ö    0    2,72    4     3     2      5   11,39      2  [5, 6]
  9,08 Betalning    5  Ö    0    2,72    3     4     2      5   13,60      1  [6]
  9,10 Betalning    4  Ö    0    2,72    2     5     2      5   13,60      0  []
  9,84 Ankomst      9  Ö    1    3,46    1     6     2      5   13,60      0  []
 10,00 Stänger    ---  Ö    1    3,62    2     6     2      5   13,60      0  []
 10,65 Plock        9  S    1    4,27    2     6     2      5   13,60      0  []
 10,74 Ankomst     10  S    0    4,27    2     6     2      5   13,60      0  []
 11,42 Betalning    6  S    0    4,27    2     6     2      5   13,60      0  []
 13,26 Betalning    9  S    1    6,11    1     7     2      5   13,60      0  []
999,00 Stop
```

RESULTAT
========

1) Av 10 kunder handlade 8 medan 2 missades.

2) Total tid 2 kassor varit lediga: 6,11 te.
   Genomsnittlig ledig kassatid: 3,06 te (dvs 23,03% av tiden från öppning tills sista kunden betalat).

3) Total tid 5 kunder tvingats köa: 13,60 te.
   Genomsnittlig kötid: 2,72 te.

Simuleringsexempel 2

```
PARAMETRAR
==========
Antal kassor, N.........: 2
Max som ryms, M.........: 7
Ankomshastighet, lambda..: 3.0
Plocktider, [P_min..Pmax]: [0.6..0.9]
Betaltider, [K_min..Kmax]: [0.35..0.6]
Frö, f..................: 13

FÖRLOPP
=======
   Tid Händelse  Kund  ?  led    ledT    I     $    :-(   köat    köT   köar  [Kassakö..]
  0,00 Start
  0,10 Ankomst     0  Ö    2    0,21    0     0     0      0    0,00      0  []
  0,38 Ankomst     1  Ö    2    0,75    1     0     0      0    0,00      0  []
  0,92 Plock       0  Ö    2    1,85    2     0     0      0    0,00      0  []
  1,11 Plock       1  Ö    1    2,03    2     0     0      0    0,00      0  []
  1,37 Ankomst     2  Ö    0    2,03    2     0     0      0    0,00      0  []
  1,46 Betalning   0  Ö    0    2,03    3     0     0      0    0,00      0  []
  1,47 Ankomst     3  Ö    1    2,05    2     1     0      0    0,00      0  []
  1,57 Betalning   1  Ö    1    2,15    3     1     0      0    0,00      0  []
  1,77 Ankomst     4  Ö    2    2,55    2     2     0      0    0,00      0  []
  1,98 Plock       2  Ö    2    2,97    3     2     0      0    0,00      0  []
  2,29 Plock       3  Ö    1    3,28    3     2     0      0    0,00      0  []
  2,34 Betalning   2  Ö    0    3,28    3     2     0      0    0,00      0  []
  2,49 Plock       4  Ö    1    3,43    2     3     0      0    0,00      0  []
  2,61 Ankomst     5  Ö    0    3,43    2     3     0      0    0,00      0  []
  2,64 Ankomst     6  Ö    0    3,43    3     3     0      0    0,00      0  []
  2,82 Betalning   3  Ö    0    3,43    4     3     0      0    0,00      0  []
  2,95 Betalning   4  Ö    1    3,56    3     4     0      0    0,00      0  []
  2,95 Ankomst     7  Ö    2    3,56    2     5     0      0    0,00      0  []
```

```
3,04 Ankomst      8  Ö    2    3,74    3     5     0      0    0,00      0  []
3,08 Ankomst      9  Ö    2    3,82    4     5     0      0    0,00      0  []
3,13 Ankomst     10  Ö    2    3,91    5     5     0      0    0,00      0  []
3,14 Ankomst     11  Ö    2    3,93    6     5     0      0    0,00      0  []
3,23 Plock        5  Ö    2    4,13    7     5     0      0    0,00      0  []
3,51 Plock        6  Ö    1    4,41    7     5     0      0    0,00      0  []
3,60 Betalning    5  Ö    0    4,41    7     5     0      0    0,00      0  []
3,67 Plock        7  Ö    1    4,47    6     6     0      0    0,00      0  []
3,87 Plock        8  Ö    0    4,47    6     6     0      0    0,00      0  []
3,94 Plock        9  Ö    0    4,47    6     6     0      1    0,08      1  [8]
3,99 Plock       10  Ö    0    4,47    6     6     0      2    0,16      2  [8, 9]
4,03 Plock       11  Ö    0    4,47    6     6     0      3    0,29      3  [8, 9, 10]
4,06 Ankomst     12  Ö    0    4,47    6     6     0      4    0,40      4  [8, 9, 10, 11]
4,09 Betalning    6  Ö    0    4,47    7     6     0      4    0,54      4  [8, 9, 10, 11]
4,12 Betalning    7  Ö    0    4,47    6     7     0      4    0,62      3  [9, 10, 11]
4,45 Ankomst     13  Ö    0    4,47    5     8     0      4    1,29      2  [10, 11]
4,47 Ankomst     14  Ö    0    4,47    6     8     0      4    1,32      2  [10, 11]
4,63 Betalning    8  Ö    0    4,47    7     8     0      4    1,65      2  [10, 11]
4,67 Plock       12  Ö    0    4,47    6     9     0      4    1,69      1  [11]
4,69 Betalning    9  Ö    0    4,47    6     9     0      5    1,72      2  [11, 12]
4,74 Ankomst     15  Ö    0    4,47    5    10     0      5    1,77      1  [12]
5,12 Ankomst     16  Ö    0    4,47    6    10     0      5    2,15      1  [12]
5,14 Plock       13  Ö    0    4,47    7    10     0      5    2,17      1  [12]
5,20 Betalning   10  Ö    0    4,47    7    10     0      6    2,28      2  [12, 13]
5,28 Betalning   11  Ö    0    4,47    6    11     0      6    2,37      1  [13]
5,32 Ankomst     17  Ö    0    4,47    5    12     0      6    2,37      0  []
5,33 Ankomst     18  Ö    0    4,47    6    12     0      6    2,37      0  []
5,35 Plock       14  Ö    0    4,47    7    12     0      6    2,37      0  []
5,47 Plock       15  Ö    0    4,47    7    12     0      7    2,49      1  [14]
5,56 Ankomst     19  Ö    0    4,47    7    12     0      8    2,66      2  [14, 15]
5,56 Betalning   12  Ö    0    4,47    7    12     1      8    2,67      2  [14, 15]
5,71 Betalning   13  Ö    0    4,47    6    13     1      8    2,82      1  [15]
5,81 Plock       16  Ö    0    4,47    5    14     1      8    2,82      0  []
6,08 Plock       17  Ö    0    4,47    5    14     1      9    3,09      1  [16]
```

```
  6,15 Betalning   14  Ö    0    4,47    5    14     1     10    3,22      2  [16, 17]
  6,17 Betalning   15  Ö    0    4,47    4    15     1     10    3,24      1  [17]
  6,20 Ankomst     20  Ö    0    4,47    3    16     1     10    3,24      0  []
  6,22 Plock       18  Ö    0    4,47    4    16     1     10    3,24      0  []
  6,36 Ankomst     21  Ö    0    4,47    4    16     1     11    3,37      1  [18]
  6,58 Betalning   16  Ö    0    4,47    5    16     1     11    3,60      1  [18]
  6,65 Betalning   17  Ö    0    4,47    4    17     1     11    3,60      0  []
  6,95 Plock       20  Ö    1    4,77    3    18     1     11    3,60      0  []
  7,00 Plock       21  Ö    0    4,77    3    18     1     11    3,60      0  []
  7,09 Ankomst     22  Ö    0    4,77    3    18     1     12    3,69      1  [21]
  7,18 Betalning   18  Ö    0    4,77    4    18     1     12    3,77      1  [21]
  7,43 Betalning   20  Ö    0    4,77    3    19     1     12    3,77      0  []
  7,56 Betalning   21  Ö    1    4,91    2    20     1     12    3,77      0  []
  7,88 Plock       22  Ö    2    5,54    1    21     1     12    3,77      0  []
  8,00 Stänger    ---  Ö    1    5,66    1    21     1     12    3,77      0  []
  8,38 Betalning   22  S    1    6,04    1    21     1     12    3,77      0  []
  9,04 Ankomst     23  S    2    6,04    0    22     1     12    3,77      0  []
999,00 Stop

RESULTAT
========

1) Av 23 kunder handlade 22 medan 1 missades.

2) Total tid 2 kassor varit lediga: 6,04 te.
   Genomsnittlig ledig kassatid: 3,02 te (dvs 36,04% av tiden från öppning tills sista kunden
betalat).

3) Total tid 12 kunder tvingats köa: 3,77 te.
   Genomsnittlig kötid: 0,31 te.
```