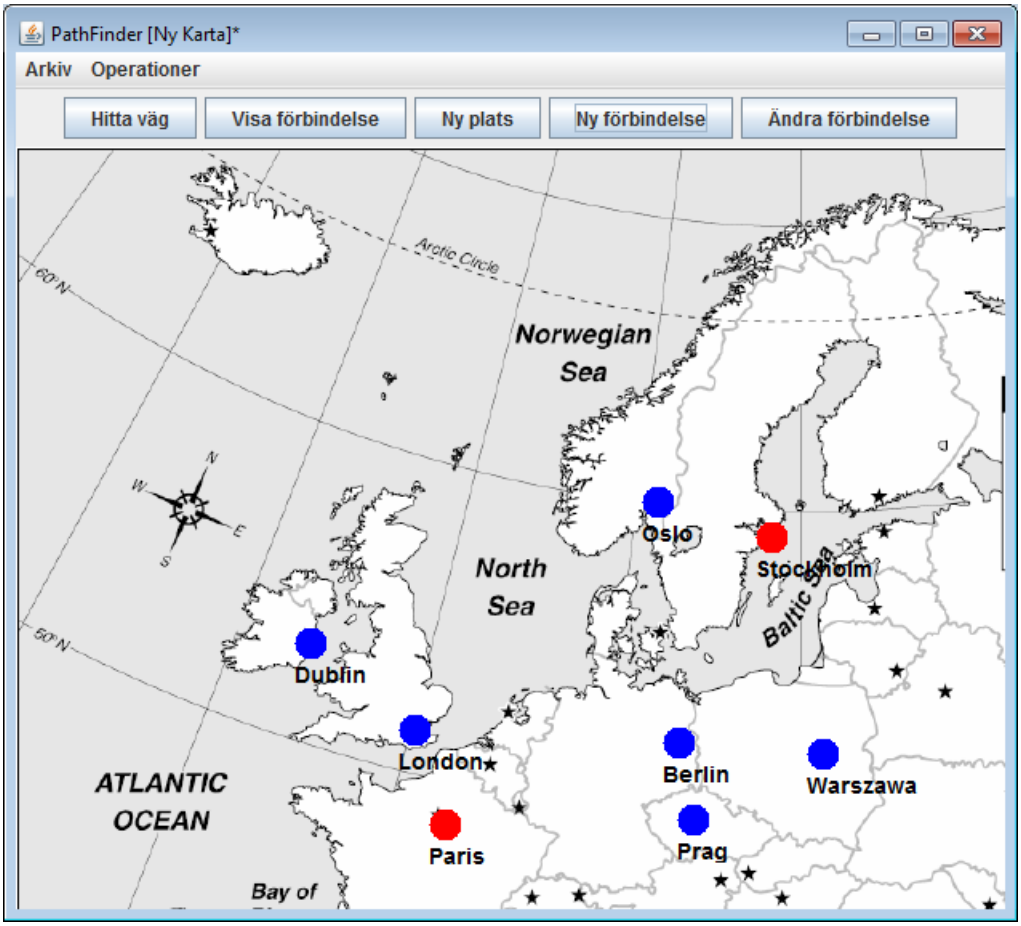
Inlämningsuppgift 2

Uppgiften löses enskilt eller i grupper om högst 3 personer.
Instruktioner för redovisning och inlämning lämnas ut lite senare i ett separat dokument.

Nedan finns en funktionell beskrivning av uppgiften. Vissa delar av uppgiften kan göras mer eller mindre ambitiöst och ger då högre eller lägre betyg. Betygskriterierna finns sist i dokumentet.

Beskrivning

Uppgiften går ut på att skriva ett program som låter användaren öppna en karta (en bakgrundsbild), med musklickningar definiera platser på kartan samt förbindelser mellan dessa platser och sedan kan hitta en väg (ev. den snabbaste) mellan två av användaren valda platser:



Platserna har namn (t.ex. ”Kista”, ”New York”, ”Andromedagalaxen” o.s.v.). De kan vara förbundna med varandra med förbindelser (som dock inte behöver ritas ut på skärmen för lägre betyg) som har ett namn (t.ex. ”Buss 514”, ”Tunnelbanans gröna linje”, ”Rymdskeppet Enterprise” o.s.v.) och en tid det tar att färdas genom denna förbindelse.

Användaren kan välja två platser genom att klicka på dem på kartan och sedan begära att få reda på en väg (ev. den snabbaste) mellan dessa platser. Användaren ska också kunna definiera nya platser, definiera förbindelser mellan två platser och ändra tiden för en förbindelse mellan två platser.

Hela programmet ska tillsammans med sin källkod paketeras som en körbar jar-fil[1] för enkel körning och distribution av programmet. Källkoden ska ligga separerad från class-filerna i en separat mapp i JAR-arkivet. Tänk på att bibehålla källkodfilernas mappstruktur i JAR-arkivet.

Programmet ska bestå av två delar – ett litet klassbibliotek för grafer och ett tillämpningsprogram.

Graf-biblioteket

Grafbiblioteket ska ligga i ett paket[2] vid namn `graphs`. Den ska minst (för betyget E) bestå av en klass `ListGraph` representerande viktade oriktade grafer för noder av en specifik typ (t.ex. Stad, Plats eller vad det är för klass du vill använda som noder i grafen), implementerade med kopplingslistor, en klass Edge (för representation av bågar, se nedan) och din nod-klass.

För högre betyg ska klasserna vara generiska[3] – noder ska kunna vara av valfri typ som anges av tillämpningen och det ska inte finnas någon nod-klass i biblioteket.

En annan utökning för högre betyg är att biblioteket förbereds för andra implementeringar av grafer genom att funktionaliteten deklareras i ett gränssnitt (*interface*) `Graph`, som `ListGraph` ska implementera, och att gemensamma operationer (som `pathExists` och `getPath`) flyttas ut till en samlingsklass `GraphMethods`.

En tredje utökning är att även implementera `MatrixGraph`.

`ListGraph` ska innehålla följande metoder:

- `add` – tar emot en nod och stoppar in den i grafen. Om noden redan finns i grafen blir det ingen förändring.
- `connect` – tar två noder, en sträng (namnet på förbindelsen) och ett heltal (förbindelsens vikt) och kopplar ihop dessa noder med bågar med detta namn och denna vikt. Om någon av noderna saknas i grafen ska undantaget `NoSuchElementException` från paketet `java.util` genereras. Om vikten är negativ ska undantaget `IllegalArgumentException` genereras. Om en båge redan finns mellan dessa två noder ska undantaget `IllegalStateException` genereras (det ska finnas högst en förbindelse mellan två noder).

  Obs att grafen ska vara oriktad, d.v.s. bågar riktade mot den andra noden måste stoppas in hos de båda noderna. I en oriktad graf förekommer ju alltid bågarna i par: från nod 1 till nod 2 och tvärtom.
- `setConnectionWeight` – tar två noder och ett heltal (förbindelsens nya vikt) och sätter denna vikt som den nya vikten hos förbindelsen mellan dessa två noder. Om någon av noderna saknas i grafen eller ingen båge finns mellan dessa två noder ska undantaget `NoSuchElementException` från paketet `java.util` genereras. Om vikten är negativ ska undantaget `IllegalArgumentException` genereras.

[1] JAR-arkiv gås igenom på föreläsning 14.
[2] Paket beskrivs på föreläsning 14.
[3] Genericitet beskrivs på föreläsning 13.

- `getNodes` – returnerar en kopia av mängden av alla noder.
- `getEdgesFrom` – tar en nod och returnerar en kopia av samlingen av alla bågar som leder från denna nod. Om noden saknas i grafen ska undantaget `NoSuchElementException` genereras.
- `getEdgeBetween` – tar två noder och returnerar bågen mellan dessa noder. Om någon av noderna saknas i grafen ska undantaget `NoSuchElementException` genereras. Om det inte finns någon båge mellan noderna resturneras null.
- `toString` – returnerar en lång sträng med strängar tagna från nodernas `toString`-metoder och bågarnas `toString`-metoder, gärna med radbrytningar så att man får information om en nod per rad för förbättrad läslighet.

Obs! Följande två metoder behöver inte implementeras för lägre betyg, men ska finnas med för högre betyg:

- `disconnect` - tar två noder och tar bort bågen mellan dem, ger `NoSuchElementException` om någon av noderna inte finns i grafen, gör ingenting om noderna inte har någon direktförbindelse
- `remove` - tar en nod och tar bort den och alla bågar från och till den från grafen, gör ingenting om noden inte finns i grafen

Obs! Följande två metoder flyttas ut från klassen `ListGraph` till klassen `GraphMethods` om man föbereder biblioteket för alternativa implementeringar av grafer.

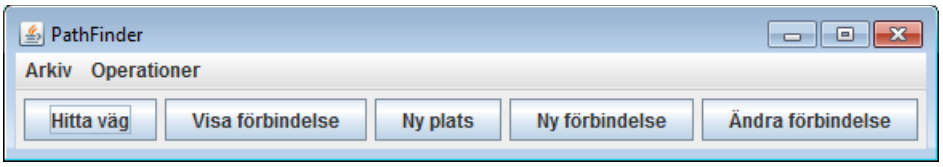
- `pathExists` – tar två noder och returnerar `true` om det finns en väg genom grafen från den ena noden till den andra (eventuellt över många andra noder), annars returneras `false`. Om någon av noderna inte finns i grafen returneras också `false`. Använder sig av en hjälpmetod för djupet-först-sökning genom en graf.
- `getPath` – tar två noder och returnerar en lista (`java.util.List`) med bågar som representerar en väg mellan dessa noder genom grafen, eller `null` om det inte finns någon väg mellan dessa två noder. I den enklaste varianten (för betyget E) räcker det alltså att metoden returnerar någon väg – i en mer avancerad variant ska den reurnerar den snabbaste vägen.

- En klass `Edge`. Alla värden för instansvariabler i denna klass ska skickas som argument till konstruktorn, och objekt av klassen ska inte kunna skapas av klasser utanför `graphs`-paketet. Klassen ska innehålla följande metoder:
  - `getDestination` – returnerar den nod som bågen pekar ut.
  - `getWeight` – returnerar bågens vikt.
  - `setWeight` – tar ett heltal och sätter bågens vikt till det angivna värdet. Om vikten är negativ ska undantaget `IllegalArgumentException` genereras.
  - `getName` – returnerar bågens namn.
  - `toString` – returnerar en sträng som innehåller information om den aktuella bågen.

Tillämpningsprogrammet[4]

Tillämpningsrogrammet går ut på att hjälpa en användare att hitta en väg (ev. den snabbaste) mellan två platser i ett nät av förbundna platser som användaren själv får definiera.

När man först startar programmet ska ett fönster öppnas med följande utseende:



Fönstret har två menyer: "Arkiv" och "Operationer" och fem knappar för de olika operationerna som användaren ska kunna begära.

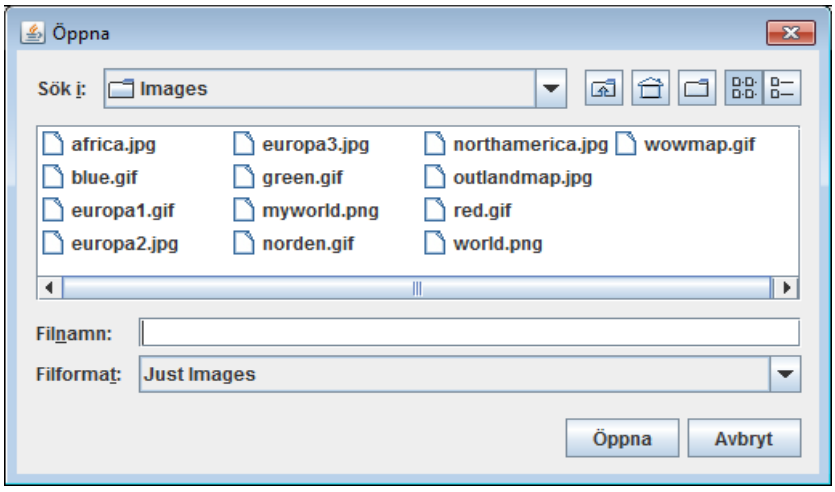
Menyn "Arkiv" ska innehålla menyvalen "Nytt" och "Avsluta" (för höga betyg ska det även innehålla alternativen "Öppna", "Spara" och "Spara som"). Menyn "Operationer" innehåller menyvalen "Hitta väg", "Visa förbindelser", "Ny plats", "Ny förbindelse" och "Ändra förbindelse" - samma operationer som finns på knapparna (tanken är att användaren ska kunna välja samma operation från menyn som från knapparna).

Namnet ”PathFinder” i titelbalken är mitt namn på programmet, du får kalla programmet vad du vill.

Alla dialogrutor som ber användaren om någon sorts input ska kunna avbrytas (via den vanliga ”Avbryt/Cancel”-knappen) utan att några ändringar sker. Vidare ska all användarinput kontrolleras så att den är giltig (siffror i sifferfält etc.), och om den inte är det ska ett felmeddelande visas och den pågående operationen avbrytas. Om användaren skulle orsaka att undantag kastas ska dessa fångas och användaren göras uppmärksam på vad för fel som uppstått.

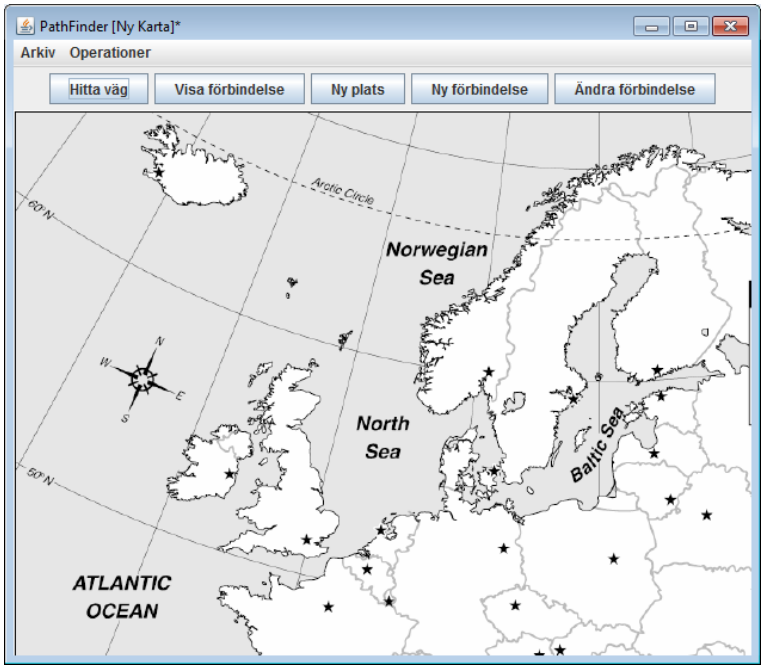
Arkiv-menyn

”Ny” innebär att användaren vill börja arbeta med en ny karta. Själva kartan är bara en bakgrundsbild som ska visas i den centrala arean. En fildialog öppnas, t.ex. (filtrering av filnamn på filtyper är inte obligatorisk):



[4] Tillämpningsprogrammet kräver olika komponenter och tekniker som illustreras på föreläsningarna 15-18

Användaren kan välja en bildfil (GIF-, JPG- ellen PNG-filer) varpå bilden laddas in och visas så här:



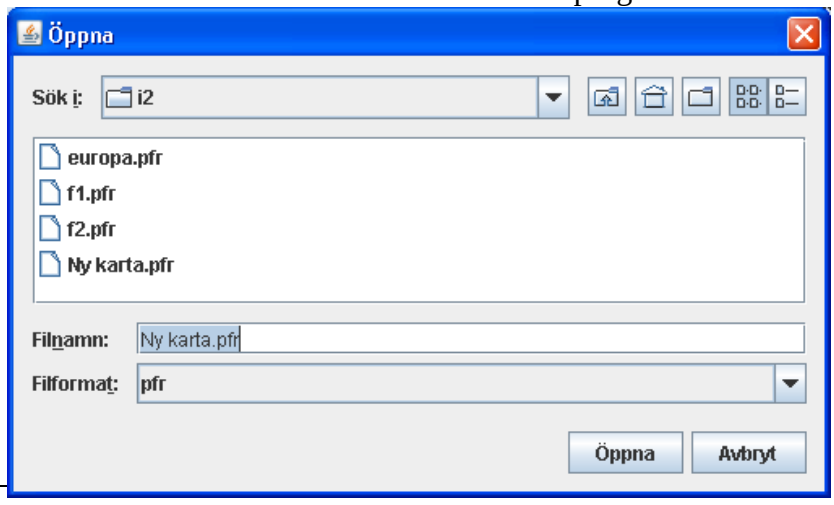
Kartan är som sagt bara en bakgrundsbild. Fönsterstorleken bör anpassas så att hela kartan visas. Obs att kartan inte borde förändras när användaren förstorar eller förminskar fönstret (eftersom platser kommer att definieras på fasta koordinater på kartan skulle deras position på kartan förändras om man tillät att kartan skalades om).

Eftersom platserna bara är meningsfulla tillsammans med sin bakgrundsbild ska alla eventuella noder och kopplingar mellan dem tas bort när man laddar en ny bild. Programmet ska alltså ”nollställas” varje gång man laddar en bild.

”Avsluta” ska avsluta programmet. Programmet ska även kunna stängas via programmets stängningsruta i övre högra hörnet.

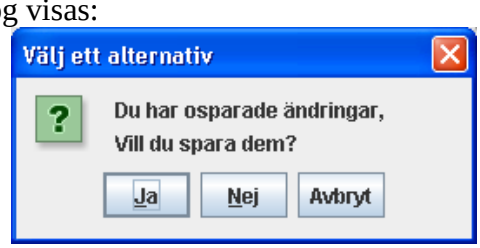
Obs! de övriga alternativen i Arkiv-menyn har att göra med att spara data på resp. läsa in dem från en fil[5], de behöver endast göras för högre betyg!

"Öppna" ska visa en fildialog för att välja ut och öppna filen. Det kan vara lämpligt att bestämma sig för någon viss ändelse för filnamnet, t.ex. ".pfr" eller ".fko" och filtrera de filer som visas i dialogrutan (detta är dock inte obligatoriskt). Om ni bestämmer er för någon sådan ändelse borde ni helst se till att den inte kolliderar med ändelser för andra program ni använder. Fildialogen:



[5] Filhantering gås igenom på föreläsning 19

Obs! dock att om användaren tidigare har hämtat en karta eller öppnat en fil och definierat nya platser eller förbindelser, eller ändrat tiden för en förbindelse (kort sagt gjort några ändringar) och inte sparat så ska följande dialog visas:



Vid "Ja" öppnas en spara-dialog (se under "Save as") och informationen sparas, vid "Nej" förkastas de gjorda ändringarna och det nya "dokumentet" öppnas, vid "Avbryt" avbryts operationen. Ovanstående gäller även vid "Nytt" (som ju innebär att man börjar arbete med ett nytt "dokument"), vid "Avsluta" samt om användaren klickar i fönstrets stängningsruta. Ert program bör alltså hålla reda på om det finns osparade ändringar.
När filen har öppnats ska fönstret visa den sparade kartan med alla platser, platserna ska visas i de båda listorna och frågor om förbindelserna och snabbaste väg ska fungera som vanligt.

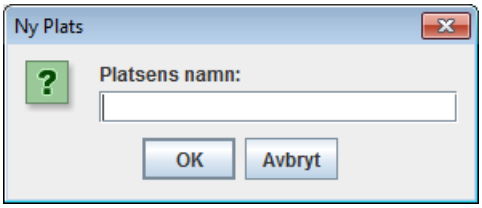
"Spara" ska bete sig på två olika sätt beroende på om det aktuella "dokumentet" är namngivet eller inte. "Dokumentet" är inte namngivet om det har skapats genom "Nytt" och inte sparats, har det däremot sparats tidigare eller öppnats från en fil så är det namngivet. Om "dokumentet" inte är namngivet så ska "Spara" bete sig som "Spara som", om det är namngivet så ska informationen sparas utan någon mer dialog med användaren.

"Spara som" ska öppna en fildialog där användaren ska kunna ange under vilket namn informationen ska sparas.

Knapparna och Operationer-menyn

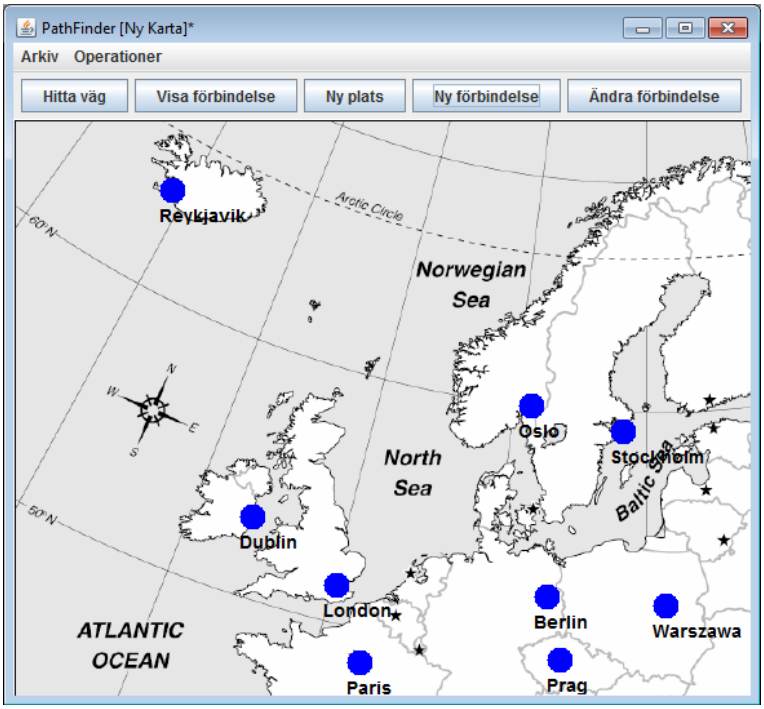
Dessa ska bara kunna användas efter att en karta laddats in i programmet.

”Ny plats” ska som namnet antyder användas för att lägga till platser på kartan. När man tryckt på knappen ska muspekaren ändras till + (crosshair) i väntan på att användaren klickar på den plats på kartan där han vill ha sin plats. Efter att användaren klickat ska en dialog lik denna dyka upp:



När användaren klickat på OK ska den nyskapade platsen dyka upp på kartan tillsammans med det namn användaren angav i dialogrutan.

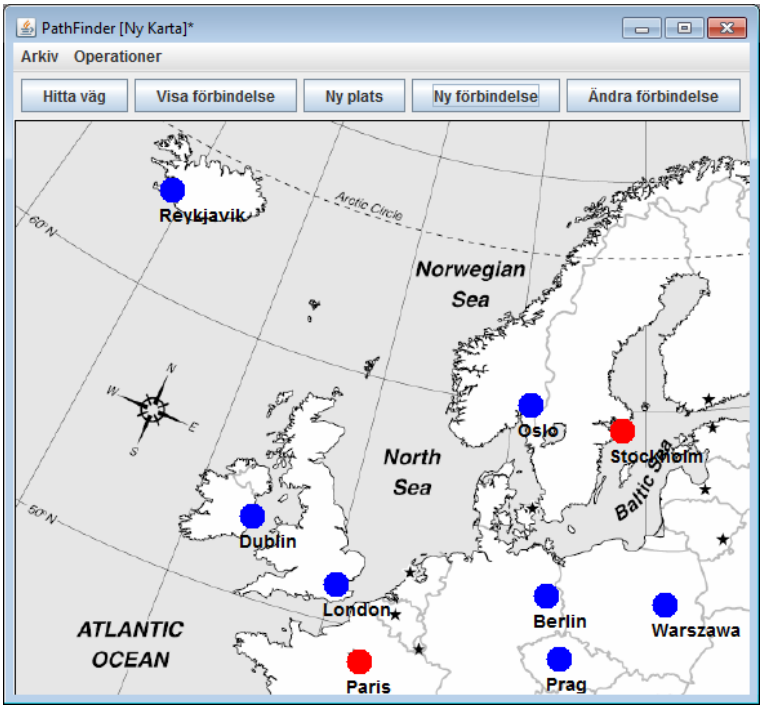
Så här skulle fönstret kunna se ut efter att några platser lagts till:



Användaren ska kunna markera platser genom att helt enkelt klicka på den plats han vill välja. Det ska aldrig gå att välja mer än två platser åt gången, och det ska gå att avmarkera de platser man klickat på så att ingen plats är markerad. Försöker man markera en tredje plats när två redan är markerade ska ingenting hända.

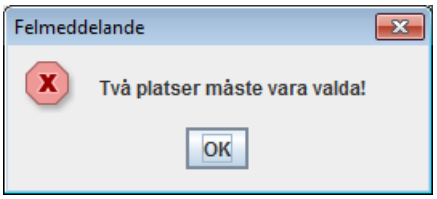
Det är viktigt att se till att platsvalet inte dras med konstiga buggar – exempelvis ska det inte vara möjligt för samma plats att vara start och mål.

De platser som f.n. är valda ska särskilja sig från övriga platser på något sätt, t.ex. genom att de ritas ut i en annan färg än övriga. Här är ett exempel, med Stockholm och Paris valda:

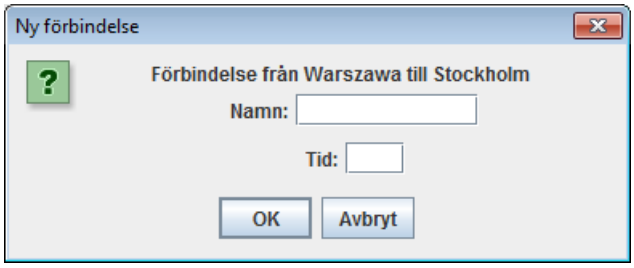


Eftersom alla operationer utom ”Ny plats” kräver att två platser är valda ska användaren hindras

från att använda dem om fel antal platser är valda. Exempelvis kan programmet visa ett felmeddelande likt det här om man klickar på knapparna utan rätt antal valda platser:

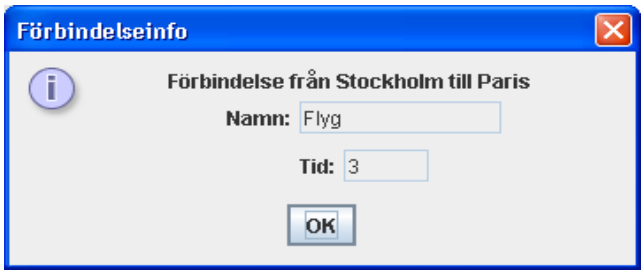


”Ny förbindelse” ska användas för att skapa förbindelser mellan platser. Om det redan finns en förbindelse mellan de två valda platserna ska ett lämpligt felmeddelande visas, annars ska ett fönster visas som frågar efter namn på förbindelsen och hur lång tid den ska ta. Den ska även indikera mellan vilka platser förbindelsen kommer att skapas. Dialogrutan kan se ut så här:



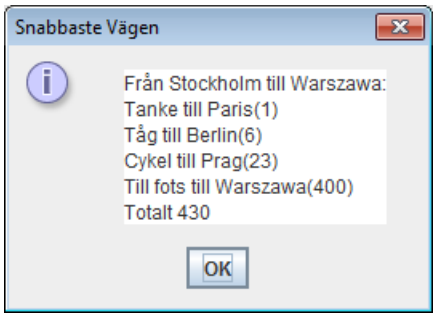
När användaren klickat på OK ska den önskade förbindelsen skapas. I grundkraven behöver inte förbindelser ritas ut på kartan, men för högre betyg ska de ritas ut. En vidare utökning är att förbindelserna är klickbara, d.v.s. om man klickar på en förbindelse så väljer man de två platser som den förbinder.

”Visa förbindelse” visas uppgifter om förbindelsen mellan de två valda platserna (eller ett felmeddelande om ingen förbindelse finns):



”Ändra förbindelse” ska öppna en liknande dialog, fast tid-fältet ska vara editerbart och man ska kunna ändra tiden. Tänk på att grafen ska vara oriktad – ändringar åt ett håll längs en förbindelse ska återspeglas åt andra hållet.

”Hitta väg” används för att söka igenom grafen efter en väg mellan de två valda platserna. Det här sker genom att programmet använder sig av den relevanta metoden i grafbiblioteket och skriver ut resultatet i en dialogruta. Dialogrutan ska innehålla all relevant information om resan, alltså var man börjar och slutar, vilka platser som passeras, vilka färdmedel som används och hur lång tid varje delsträcka tar. Den ska även ange hur lång tid resan tar totalt. Ett sätt att presentera resultatet ser ut så här:



Finns det ingen väg mellan de valda platserna så ska dialogrutan meddela detta (om det inte är den snabbaste vägen som visas bör fönstrets titel ändras).

Betygskriteria

På flera ställen i beskrivningen ovan anges olika ambitionsnivåer som motsvarar högre eller lägre betyg. Nedan följer betygskriterier för betygsättning av lösningarna, angivna i termer av dessa ambitionsnivåer i den funktionella beskrivningen.

Obs att betygen motsvarar de valda ambitionsnivåerna – lösningarna måste vara korrekta (fungerande) enligt den valda ambitionsnivån för ett godkänt betyg. Om den delen av programmet som skulle ge ett högre betyg inte fungerar så ska den tas bort från programmet för att man ska kunna få det lägre betyget – program som inte fungerar accepteras inte.

Ett annat generellt krav är att koden är rätt indenterad, att konventioner för stavning av namn på de olika syntaktiska enheterna följs o.s.v.

Nedan anges vad som krävs för de olika betygen. Med **fet** stil markeras vad som särsklijer betyget från närmast lägre betyg.

A. Paketet graphs:

- både ListGraph- **och MatrixGraph**-implementeringar ska finnas med
- gränssnittet (interface) Graph som grafklasser implementerar ska finnas med
- ListGraph, matrixGraph och Graph ska vara generiska
- även metoderna disconnect och remove är implementerade
- metoderna pathsExists och getPath är implementerade som generiska statiska metoder i en separat klass GraphMethods (motsvarande Collections för samlingar och Arrays för arrayer)
- metoden getPath returnerar den snabbaste vägen

Tillämpningsprogrammet:

- sparande på fil/inläsning från fil ska vara implementerade
- förbindelser mellan platserna ritas ut **och de är klickbara i en liten omgivning (3-5 pixlar) kring sträcket), implementerade som en egen komponent**
- ska vara förberett för byte av karta

B. Paketet graphs:

- grafimplementeringen ska vara ListGraph (MatrixGraph behöver inte finnas)
- det ska även finnas ett gränssnitt (interface) Graph som ListGraph implementerar
- ListGraph och Graph ska vara generiska
- **även metoderna disconnect och remove är implementerade**
- metoderna pathsExists och getPath är implementerade som generiska statiska metoder i en separat klass GraphMethods (motsvarande Collections för samlingar och Arrays för arrayer)
- metoden getPath returnerar den snabbaste vägen

Tillämpningsprogrammet:

- **sparande på fil/inläsning från fil ska vara implementerade**
- **även förbindelser mellan platserna ritas ut, men de behöver inte vara klickbara**
- ska vara förberett för byte av karta

C. Paketet graphs:

- grafimplementeringen ska vara ListGraph (MatrixGraph behöver inte finnas)
- det ska även finnas ett gränssnitt (interface) Graph som ListGraph implementerar
- ListGraph och Graph ska vara generiska
- metoderna disconnect och remove behöver inte vara implementerade
- **metoderna pathsExists och getPath är implementerade som generiska statiska metoder i en separat klass GraphMethods (motsvarande Collections för samlingar och Arrays för arrayer)**
- **metoden getPath returnerar den snabbaste vägen**

Tillämpningsprogrammet:

- sparande på fil/inläsning från fil behöver inte vara implementerade
- endast platser behöver ritas ut, inga förbindelser
- ska vara förberett för byte av karta

D. Paketet graphs:

- **grafimplementeringen ska vara ListGraph (MatrixGraph behöver inte finnas)**
- **ListGraph ska vara generisk och alltså hantera noder av godtycklig typ**
- metoderna disconnect och remove behöver inte vara implementerade
- pathExists och getPath kan vara instansmetoder i ListGraph
- getPath kan returnera en väg som inte behöver vara "snabbast"

Tillämpningsprogrammet:

- sparande på fil/inläsning från fil behöver inte vara implementerade
- endast platser behöver ritas ut, inga förbindelser
- **ska vara förberett för byte av karta**

E. Paketet graphs:

- grafklassen kan vara ListGraph eller MatrixGraph (endast en av dem behöver finnas)

- grafklassen behöver inte vara generisk utan kan vara konstruerad för en specifik typ (t.ex. Stad)
- metoderna pathExists och getPath kan vara instansmetoder i ListGraph
- getPath kan returnera en väg mellan noderna som inte behöver vara "snabbast"
- metoderna disconnect och remove behöver inte vara implementerade

Tillämpningsprogrammet:

- sparande på fil/inläsning från fil behöver inte vara implementerade
- endast platser behöver ritas ut, inga förbindelser
- behöver inte vara förberett för byte av karta (behöver inte se till att nollställa sig o.s.v.)