Tentamen, EDAA10 Programmering i Java

2010–05–31, 8.00–13.00

Anvisningar: Denna tentamen består av 4 uppgifter. Preliminärt ger uppgifterna $16 + 17 + 5 + 7 = 45$ poäng. För godkänt betyg krävs 20 poäng.

Uppgift 3 är en extrauppgift för dig som eftersträvar högre betyg.

Tillåtna hjälpmedel:

- Java-snabbreferens.

När rättningen är klar meddelas detta på kursens hemsida.

Tentamensvillkor: Om du tenterar utan att vara tentamensberättigad annulleras din skrivning. För att undvika att någon skrivning annulleras av misstag anslås vilka som, enligt institutionens noteringar, tenterat utan att vara tentamensberättigade. Namnen anslås (lätt krypterade) senast en vecka efter tentamen på kursens hemsida. Felaktigheter i institutionens noteringar kan därefter påtalas fram till nästa tentamenstillfälle då resterande skrivningar annulleras.

Efter Eyjafjallajökulls utbrott tidigare i år tvingades man ställa in många flygningar, eftersom halterna av aska var för höga över stora delar av Europa – vi skall i denna tentamen skriva programkod för att hantera några av de problem som uppstod i samband med detta.

Uppgift 1

Sfärisk geometri kan vara ganska knepig, så för att göra uppgiften enklare har vi en *färdigskriven* klass `Location` som beskriver punkter på jordytan – den har följande specifikation:

```
public class Location {
  /** Skapar en ny punkt med given latitud och longitud. Latitud och
      longitud ges som strängar, men det spelar ingen roll för uppgiften. */
  public Location (String latitude, String longitude);

  /** Ger avståndet till en annan punkt (other). Avståndet mäts i kilometer. */
  public double distanceTo(Location other);

  /** Ger en lista med punkter utmed storcirkelbågen (d.v.s. närmaste vägen) fram
      till en annan punkt (other). Punkterna i den returnerade listan har ungefär
      en kilometers avstånd till varandra och vi kan förutsätta att de går jämnt ut. */
  public ArrayList<Location> geodesicTo(Location destination);

  /** Ger en sträng som beskriver punkten. */
   public String toString();
}
```

I vårt system har vi även en *färdigskriven* klass `Airport`, som beskriver en flygplats – dess specifikation är:

```
public class Airport {
  /** Skapar en flygplats.
      id — flygplatsens "id", t.ex. "CPH"
      name — flygplatsens namn, t.ex. "Kastrup"
      city — närmaste stad, t.ex. "Copenhagen"
      location — flygplatsens position */
  public Airport (String id, String name, String city, Location location);

  /** Ger en sträng som beskriver flygplatsen. */
  public String toString();

  /** Ger flygplatsens position (som ett Location-objekt). */
  public Location getLocation();

  /** Ger flygplatsens "id". */
  public String getId();

  /** Ger flygplatsens namn. */
  public String getName();

  /** Ger närmaste stad. */
  public String getCity();
}
```

Vi vill också ha en klass som håller reda på alla flygplatser. Denna klass skall du själv skriva, och den skall ha följande specifikation:

```
public class AirportDatabase {
  /** Skapar en tom databas. */
  public AirportDatabase();

  /** Lägger in en ny flygplats (airport), om det inte redan finns en flygplats med
      samma "id"  i databasen. Returnerar true om den gick att lägga in och false om
      den redan fanns med i databasen. */
  public boolean add(Airport airport);

  /** Ger flygplatsen med identiteten id, eller null om ingen sådan flygplats finns. */
  public Airport get(String id);

  /** Ger en lista med de flygplatser som ligger inom ett givet avstånd (distance)
      från en given plats (location). */
  public ArrayList<Airport> findWithin(Location location, double distance);

   /** Ger den flygplats som ligger närmast en given flygplats (airport).
       Flygplatsen själv räknas naturligtvis inte in. */
  public Airport findClosest(Airport airport);
}
```

Implementera klassen `AirportDatabase` enligt beskrivningen ovan. Du skall naturligtvis utnyttja de färdigskrivna klasserna `Location` och `Airport`

Uppgift 2

För att planera flygningar måste vi ha information om hur askan är fördelad i atmosfären, och vi har för detta en *färdigskriven* klass `AshData`. Den beskriver koncentrationen av aska på olika nivåer i atmosfären på olika platser. Varje nivå består av ett ca 500 meter högt skikt, där lägsta nivån börjar på ungefär 6 000 meters höjd (lägre kan man inte flyga), och den högsta nivån är på ca 11 000 meters höjd (högre kan man inte flyga). Klassen har följande specifikation:

```
public class AshData {
  /** Läser in information om askkoncentrationer från en fil
      med namnet fileName. */
  public AshData(String fileName);

  /** Ger den mängd aska man får igenom en 1 m^2 stor yta som färdas 1 km
      vinkelrät mot luften i området ovanför en given position (loc),
      på en viss höjdnivå (level), som kan vara  0..nbrOfLevels()-1. */
  public double getAshAmount(Location loc, int level);

  /** Ger antalet olika höjdnivåer som luftrummet är indelat i.
      Nivåerna räknas från 0 och uppåt (till nbrOfLevels()-1).
      Nivå 0 motsvarar ungefär 6000 meter. */
  public int nbrOfLevels();

 /** Ger höjden (i meter) för ett givet nivånummer (level).
     Nivåerna räknas i intervallet 0..nbrOfLevels()-1. */
  public double getAltitude(int level);
}
```

Det som avgör om man kan flyga eller inte är mängden aska som passerar genom motorn, och i uppgiften skall vi därför räkna ut hur mycket aska som passerar igenom varje m^2 av motorernas yta (den yta som riktas framåt).

Operationen `getAshAmount` ger den mängd aska man får igenom en $1\,m^2$ stor yta som färdas 1 km vinkelrät mot luften i området ovanför en given punkt, på en given höjdnivå, och den är vald så att vi enkelt kan summera mängderna när vi går igenom en flygsträcka (se `geodesicTo` i klassen `Location`) – varje steg i våra `geodesicTo`-listor är 1 km långt.

Mängden aska beror av vilken höjd-nivå vi flyger på. Vi kommer att (i klassen `Flight` nedan) testa alla möjliga nivåer, och se vilken nivå som ger minst aska. Vi förutsätter att vi flyger på samma nivå genom hela flygningen (även start och landning räknas på denna nivå).

Flygturerna klassificeras med avseende på sammanlagd mängd aska som passerar en m^2 av motorn, och vi har följande gränser:

- klass 0: Mindre än 10 kg aska passerar – medför inga problem att flyga.
- klass 1: Minst 10 kg, och mindre än 25 kg – medför liten risk att flyga.
- klass 2: Minst 25 kg, och mindre än 70 kg – medför ganska stor risk att flyga.
- klass 3: Minst 70 kg aska, gör att flygning är direkt farlig (och därmed förbjuden).

Vi vill ha en klass `Flight` som försöker planera en flygning mellan två givna flygplatser, med en given ask-koncentration i atmosfären. Denna klass skall du skriva, och den skall ha följande specifikation:

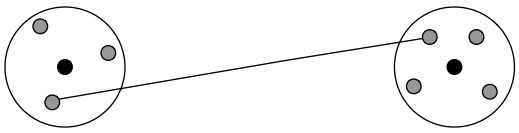
```
public class Flight  {
  /** Skapar en flygtur från flygplatsen origin till flygplatsen destination. */
  public Flight (Airport origin, Airport destination);

  /** Ger startflygplatsen. */
  public Airport getOrigin();

  /** Ger destinationsflygplatsen. */
  public Airport getDestination();

  /** Räknar ut hur mycket aska vi passerar per m^2 om vi flyger på nivån level.
      level – nivån
      ashdata – information om ask-koncentrationerna (se klassen AshData). */
  public double getAccumulatedAsh(AshData ashData, int level);

  /** Bestämmer den nivå (räknas från 0 och uppåt) som ger minst aska
      (den högsta möjliga nivån, om flera skulle ge samma värde).
      ashData – information om ask-koncentrationerna (se klassen AshData). */
  public int getBestLevel(AshData ashData);

   /** Bestämmer flygningens klassifikation om vi flyger på bästa möjliga nivå.
       ashData – information om ask-koncentrationerna (se klassen AshData). */
  public int getClassification(AshData ashData);

  /** Bestämmer bästa höjd att flyga på (i meter).
      ashData – information om ask-koncentrationerna (se klassen AshData). */
  public double getBestAltitude(AshData ashData);
}
```

Ledning 1: I implementationen av metoden `getAccumulatedAsh` ska man hämta punkterna med hjälp av lämplig metod i klassen `Location`.
Ledning 2: I implementationen av metoderna `getClassification` och `getBestAltitude` är det lämpligt att använda sig av metoderna `getBestLevel` och/eller `getAccumulatedAsh`.

Uppgift 3

Om en flygtur inte kan göras, på grund av att mängden aska man skulle passera igenom är för stor, så ställs flygturen in. Om man kan acceptera en viss landtransport, kan man ibland hitta en alternativ flygtur till och från flygplatser i närheten av den planerade start- resp slutflygplatsen.

Vi vill därför i klassen `Flight` ha en operation, `getSafeAlternative`, som ger någon annan, helt säker flygtur (alltså klass 0), och som för oss tillräckligt nära vårt mål. När vi anropar operationen talar vi om hur långt vi är beredda att åka till lands (med tåg, buss eller bil) för att ta oss till och från flygplatserna, både start- och slut-flygplatsen (dvs om vi är beredda att åka 100 km till en annan flygplats (från vår tänkta startflygplats), och lika långt från en annan flygplats till vår tänkta målflygplats, så anropar vi operationen med värdet 100 som andra parameter). Se figuren nedan.



Figur 1: De svarta prickarna i figuren motsvarar de flygplatser vi vill flyga mellan (origin och destination). De grå prickarna är flygplatser inom ett visst avstånd (motsvarande radien i de båda cirklarna) från dessa. Vi söker en säker flygning mellan en av flygplatserna i den vänstra cirkeln och en i den högra. (Den utritade linjen skulle kunna representera en sådan.)

Metoden `getSafeAlternative` i klassen `Flight`, som du skall implementera, har följande specifikation:

```
/** Ger en alternativ flygtur (med klassificering 0), om denna flygning skulle
    vara för farlig. Om det inte finns någon sådan säker flygtur returneras null.
    airports – en databas med alla flygplatser.
    maximumGroundDistance – den största sträcka vi kan tänka oss att färdas på marken.
    ashData – information om ask-läget i atmosfären.*/
public Flight getSafeAlternative(AirportDatabase airports,
                                 double maximumGroundDistance,
                                 AshData ashData);
```

Uppgift 4

Nedan finns ett påbörjat huuvudprogram som du skall skriva färdigt. Din uppgift är att komplettera programmet så att det :

- Skapar ett `AshData`-objekt utgående från en fil med namnet ”ashdata.txt”
- Skapar en flygning mellan två flygplatser med identiteterna ”CPH” och ”BOS”. Du får förutsätta att flygplatser med dessa identiteter finns.
- Tar reda på klassifikationen för denna flygning.
- Om flygningen inte är helt säker, skapar en ny alternativ flygning och om sådan finns skriver ut mellan vilka städer den går. Du får själv bestämma hur långt du är beredd att åka till lands.

```
public static void main(String[] args) {
   AirportDatabase airports = new AirportDatabase();
   // Här finns programkod som med hjälp av metoden add i klassen
   // AirportDatabase fyller databasen med flygplatser.
   // Detta behöver du inte skriva.
   ...
}
```