

Omkontrollskrivning, Programmeringsteknik för D/C

2012–08–23, 8.00–13.00

Anvisningar: Preliminärt ger uppgifterna $13 + 7 + 16 + 4 = 40$ poäng. För godkänt betyg krävs ungefär 25 poäng. Tillåtet hjälpmedel: Java-snabbreferens.

Betyget på denna skrivning utgör betyget på kontrollskrivningarna, oavsett resultatet på tidigare skrivningar (dock sänks inte ett tidigare godkänt betyg). Resultatlistan anslås på institutionens anslagstavla när rättningen är klar.

Bakgrund

Denna tentamen handlar om att producera rättningsprotokoll till en tentamen med enklare tentamensuppgifter (till exempel typtal). För varje tentamensuppgift finns det ett antal saker som man får poäng för att göra. Det finns även saker som man inte ska göra, och de ger poängavdrag. Som exempel tar vi en uppgift där studenterna ska implementera en klass för punkter i xy -planet:

```
public class Point {
    private double x, y;

    /** Skapar en punkt med koordinaterna (x,y) */
    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }

    /** Tar reda på avståndet till punkten other */
    public double distanceTo(Point other) {
        double dx = x - other.x;
        double dy = y - other.y;
        return Math.sqrt(dx * dx + dy * dy);
    }
}
```

Man får full poäng om man gör följande (dessa punkter ger pluspoäng, så vi kallar dem plus-punkter):

- p1 (1 poäng): Deklarerar två reella attribut.
- p2 (1 poäng): Ger attributen rätt startvärden.
- p3 (2 poäng): Räknar ut avstånd på rätt sätt.

Man kan även göra ett antal fel (dessa punkter ger avdrag, så vi kallar dem minus-punkter):

- m1 (−1 poäng): Glömmer att deklarerera lokala variabler.
- m2 (−1 poäng): Deklarerar parametrar vid metदानrop.
- m3 (−1 poäng): Använder klassnamn som beteckning för `this`.
- ...

Beteckningarna p1, p2, m1, m2, ... är koder som rättarna skriver i särskilda markeringsrapporter. Varje student får en egen markeringsrapport för sin lösning av tentan. I rapporten skriver rättaren de punkter som påträffas på varje uppgift.

En rättare som ser en helt korrekt lösning (som den ovan) skriver i markeringsrapporten:

Uppgift 1: p1, p2, p3

Dessa markeringar läses in i ett program och översätts till ett rättningsprotokoll enligt:

Uppgift 1.

- 1/1 p: Deklarerar två reella attribut.
- 1/1 p: Ger attributen rätt startvärden.
- 2/2 p: Räknar ut avstånd på rätt sätt.

Sammanlagt 4 poäng av 4 möjliga.

Följande lösning är inte lika bra:

```
public class Point {
    private double x, y;

    /** Skapar en punkt med koordinaterna (x,y) */
    public Point(double x, double y) {
        x = x;
        y = y;
    }

    /** Tar reda på avståndet till punkten other */
    public double distanceTo(Point other) {
        dx = Point.x - other.x;
        dy = Point.y - other.y;
        return Math.sqrt(dx + dy);
    }
}
```

Här deklareras attributen, så rättaren skriver p1 i markeringsrapporten. Men varken p2 eller p3 är korrekt implementerade, så rättaren skriver inte dem i rapporten. Dessutom innehåller lösningen de två felen m1 och m3, så i markeringsrapporten får vi:

Uppgift 1: p1, m1, m3

vilket ger rättningsprotokollet:

```
Uppgift 1.
1/1 p: Deklarerar två reella attribut.
-1 p: Glömmer att deklarera lokala variabler.
-1 p: Använder klassnamn som beteckning för this.
0/1 p: Ger attributen rätt startvärden.
0/2 p: Räknar ut avstånd på rätt sätt.
Sammanlagt 0 poäng av 4 möjliga.
```

Utöver alla punkterna i markeringsrapporten ska alltså även uppgiftens övriga pluspunkter skrivas ut i protokollet, men eftersom de inte finns med i lösningen så ska de ge 0 poäng. Vad gäller uppgiftens minuspunkter så är det bara de som förekommer i lösningen som ska skrivas ut. Det spelar ingen roll i vilken ordning punkterna skrivs ut.

Sammanlagt kan man aldrig få mindre än 0 poäng på en uppgift, så i detta exempel blir poängen 0 även om summan är -1.

Uppgiftsbeskrivning

Du ska skriva program för att skapa rättningsprotokoll och kommer att använda följande klasser:

- *Item (färdigskriven)*: Beskriver en plus-/minuspunkt enligt exemplet ovan.
- *Problem*: Beskriver en uppgift på en tentamen. En uppgift har flera plus-/minuspunkter.
- *MarkReport*: Beskriver en markeringsrapport för lösningarna som en student lämnat in. För varje uppgift finns en lista med de plus-/minuspunkter som förekommer enligt exemplet ovan.
- *Student (färdigskriven)*: Beskriver en student som tenterat. Varje student har en markeringsrapport med markeringar för de inlämnade lösningarna.
- *Exam*: Beskriver en hel tentamen med ett antal uppgifter och de studenter som tenterat.

Klassen `Item` är färdigskriven och beskriver punkterna. Varje punkt har en beskrivning och ger ett antal poäng. För saker som man ska göra (pluspunkter) är poängen > 0 , för saker man inte får göra (minuspunkter) är poängen < 0 .

`Item`

```
/** Undersöker om två punkt-objekt beskriver samma punkt. */
boolean equals(Item other);

/** Ger en beskrivning av punkten, exempelvis "Räknar ut avstånd på rätt sätt." */
String description();

/** Ger antalet poäng för punkten. Detta värde är  $> 0$  för sådant
    som man ska göra och  $< 0$  för sådant som man inte får göra. */
int points();

/** ... konstruktor och ytterligare metoder som inte behövs i denna tentamen. */
```

I denna tentamen förutsätter vi att punkterna som rättarna skrivit på varje uppgift i markeringsrapporten är inlästa och att motsvarande `Item`-objekt har skapats. Den programkod du ska skriva handlar om att hantera dessa punkter och skriva ut rättningsprotokoll.

En uppgift beskrivs av klassen `Problem`. Varje uppgift har ett nummer och en lista med punkter (`Item`) som kan förekomma i uppgiften, dvs listan innehåller både pluspunkter och minuspunkter.

`Problem`

```
/** Skapar en uppgift med nummer number (1..) och de punkter items
    som kan förekomma i uppgiften. */
Problem(int number, ArrayList<Item> items);

/** Ger uppgiftens nummer. */
int number();

/** Avgör om uppgiften other har samma nummer som denna uppgift. */
boolean equals(Problem other);

/** Ger uppgiftens maxpoäng. */
int maxPoints();

/** Ger en lista med alla punkter som kan förekomma i uppgiften. */
ArrayList<Item> itemsFor();
```

Klassen `MarkReport` beskriver en markeringsrapport med de punkter som förekommer i en students lösningar av uppgifterna. Klassen samlar markeringarna för de olika uppgifterna så att man kan skriva ut ett rättningsprotokoll (enligt exemplet ovan) för en given uppgift.

`MarkReport`

```
/** Skapar en markeringsrapport för en students lösning av en tenta
    med nbrOfProblems uppgifter. */
MarkReport(int nbrOfProblems);

/** Markerar att punkten item förekommer i lösningen till uppgiften problem. */
void add(Problem problem, Item item);

/** Skriver ut rättningsprotokollet för uppgiften problem. */
void printProtocol(Problem problem);
```

En student som tenterar beskrivs av en färdigskriven klass Student:

Student

```
/** Skapar en student med markeringsrapporten marks för lösningarna. */  
Student(MarkReport marks);  
  
/** Ger studentens markeringsrapport. */  
MarkReport marks();  
  
/** Ger en textbeskrivning av studenten på formen  
    "910101-0123 - Andersson, Anna". */  
String identity();
```

En tenta beskrivs av klassen Exam och består av ett antal uppgifter. Dessutom håller ett Exam-objekt reda på kurskod och datum för den aktuella tentan samt vilka studenter som tenterat:

Exam

```
/** Skapar en tenta med kurskoden course på datumet date och  
    med uppgifterna problems. */  
Exam(String course, String date, ArrayList<Problem> problems);  
  
/** Lägger till en ny student. */  
void addStudent(Student student);  
  
/** Ger maxpoängen på hela tentan. */  
int maxPoints();  
  
/** Skriver ut ett rättningsprotokoll för alla uppgifterna på tentan för  
    studenten student (se nedan). */  
void printProtocol(Student student);  
  
/** Skriver ut rättningsprotokoll för samtliga studenter som skrivit tentan. */  
void printAllProtocols();
```

Rättningsprotokollet för en student ska bestå av en rubrik med kurskod och datum, sedan studentens identitet, sedan rättningsprotokollet för varje uppgift:

Tentamen i EDA001 2012-06-27

Skrivande: 910101-0123 - Andersson, Anna

Uppgift 1.

1/1 p. Deklarerar två reella attribut.

...

Sammanlagt 4 poäng av 4 möjliga.

Uppgift 2.

3/3 p. Hämtar punkten längst ifrån origo.

...

Sammanlagt 12 poäng av 15 möjliga.

...

Uppgifter

1. Implementera klassen Exam.
2. Implementera klassen Problem.
3. Implementera klassen MarkReport.

I klassen MarkReport ska det finnas en lista för varje uppgift på tentan. Varje lista innehåller de markeringar studenten fått på just den uppgiften. Klassen ska alltså ha ett attribut som är en vektor med sådana listor (ArrayList<Item>-objekt).

Att skapa en vektor med referenser till listor av typen ArrayList är tyvärr lite krångligare än att skapa vektorer med referenser till objekt, och det är inget vi har gått igenom på kursen. För att skapa en vektor demo med referenser till n listor med Item-objekt så hade det naturliga varit att skriva:

```
ArrayList<Item>[] demo = new ArrayList<Item>[n];
```

Men man kan inte göra så i Java. I stället ska man skriva:

```
ArrayList<Item>[] demo = (ArrayList<Item>[]) new ArrayList[n];
```

Sedan använder man vektorn demo precis som vanligt. Det enda som är lite speciellt är alltså hur man skapar vektorn.

4. I en del kurser har man tentor som skrivs anonymt. I stället för namn används då en anonym kod (några bokstäver) och ett löpnummer för att identifiera studenterna, till exempel dfgr 319. Klasserna i de tidigare uppgifterna ska kunna användas både för tentor som skrivs anonymt och för "vanliga" tentor. Då måste klassen Student i beskrivningen ovan vara abstrakt (det är metoden identity() som är abstrakt) och ha två subclasser:

- AnonymousStudent, och
- KnownStudent.

Specifikationen för AnonymousStudent är, utöver metoderna i Student:

AnonymousStudent

```
/** Skapar en anonym skrivande med ett givet löpnummer (1..) och en
    given anonym kod (typ 'dfgr'). */
AnonymousStudent(MarkReport marks, int seq, String code)

/** Ger identiteten på formen 319 - dfgr, dvs med löpnummer och kod. */
String identity()
```

- a) Implementera klassen AnonymousStudent som en utvidgning av den abstrakta klassen Student.
- b) Behöver man göra några ändringar i klassen Exam om klassen Student är abstrakt och man skickar in ett AnonymousStudent-objekt till metoden printProtocol? Vilka, i så fall? Du behöver inte skriva någon kod, bara förklara i ord.