LULEÅ TEKNISKA UNIVERSITET

Tentamen i
Objektorienterad programmering och design
Totala antalet uppgifter: 5
Lärare: Håkan Jonsson 073-8201700
Resultatet klart 2015-06-06.

Kurskod	D0010E
Datum	2015-05-16
Skrivtid	5 tim

Tillåtna hjälpmedel: Inga. Bilagor: inga.

Observera att lösningar baserade på andra klasser ur Javas standardbibliotek än sådana som anges i uppgifterna ger noll poäng.

1. **Programming**

   a) Klassen `A` innehåller den publika metoden `x()`. Klassen `B` ärver `A` och innehåller en ny deklaration av `x()`. Klassen `C` ärver sen `B` och innehåller även den en ny deklaration av `x()`. Samtliga deklarationer av `x()` har samma typsignatur.

   Om nu ett program innehåller deklarationen `B b = new C();` är det då metoden `x()` i `A`, `B` eller `C` som anropas då satsen `b.x();` utförs? (1p)

   b) Det finns 4 synlighetsattribut i Java: `private`, `public`, `protected` samt ett underförstått attribut man inte skriver ut.
   Vad betyder var och en av dessa? (2p)

   c) En *kulpyramid* med höjden n består av en kvadratisk botten med n^2 kulor på vilken det står en kulpyramid med höjden $n-1$. En kulpyramid med höjden 1 består av 1 kula.

   Skriv en *rekursiv* metod i Java som beräknar totala antalet kulor i en kulpyramid som har höjden n. (2p)

2. **Spel** (5p)

   I ett spel har man en rektangulär spelplan med rutor som var och en innehåller en referens till ett objekt av typen `Marker`. Det enda du vet om denna typ, förutom att du inte behöver implementera den, är att `Marker`-objekt kan jämföras med metoden `equals`: Givet två `Marker`-objekt `p` och `q` returnerar `p.equals(q)` värdet `true` om och endast om `p` är lika med `q`.

   En spelplan innehåller en vinnande position om den innehåller ett område på 2×2 rutor med marker som är lika.

   Skriv en metod som tar en spelplan, i form av en rektangulär 2-dimensionell array med element av typen `Marker`, som argument och som returnerar `true` om och endast om den innehåller en vinnande position.

3. **Game scores** (5p)

   Skriv en Javaklass `GameScore` vars objekt håller reda på poängställningen i en match mellan två lag där ett är hemmalag och ett bortalag.

   Poängställningen är 0-0 när ett objekt skapas (dvs en match startar) och ska kunna ökas med en poäng i taget via två metoder `incHomeTeam()` och `incVisitingTeam()`. Nuvarande ställning ska ges av metoden `currentScore()` som ger tillbaka en sträng med hemmalagets poäng, ett bindestreck och bortalagets poäng.

   Det kan spelas flera matcher samtidigt och man ska kunna få det genomsnittliga antalet mål per match genom ett anrop till metoden `statistics()`.

4. Inheritance and exceptions (5p)

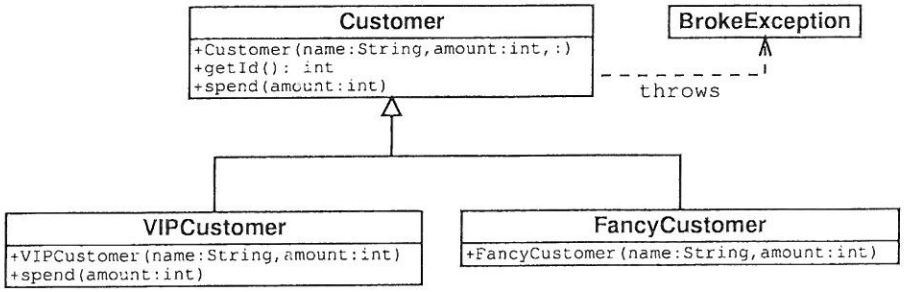

Figure 1: UML diagram för uppgift 4.

Skriv klasser som representerar kunder enligt UML-diagrammet i figur 1 och enligt följande anvisningar:

- Klassen `Customer` representerar en vanlig kund. Varje kund har ett ID-nummer, ett namn, och en viss summa pengar. Varje ny kund som skapas får automatiskt ett ID-nummer i form av ett heltal, som är ett större än den föregående kundens ID-nummer. Den första kunden som skapas får ID-numret 0. Metoden `getId()` ska returnera kundens ID-nummer. Konstrueraren tar två argument: kundens efternamn och kundens förmögenhet.

  Metoden `spend(int amount)` innebär att kunden handlar något för en viss summa.

  - Om `amount` är större än kundens förmögenhet ska köpet inte genomföras. Istället ska undantaget `BrokeException` kastas. Du får anta att klassen som representerar undantaget redan är skriven.
  - Om kunden däremot har råd med köpet ska kundens förmögenhet minskas med `amount` och en utskrift göras med kundens namn följt av en kort text som förklarar att kunden handlat för `amount`.

- Klassen `VIPCustomer`, som utökar `Customer`, representerar kunder med kontakter. De behöver aldrig betala: när `spend`-metoden för en sådan kund utförs ska kundens förmögenhet inte påverkas, utan allt som händer är att texten "Jag känner faktiskt ägaren!" ska skrivas ut på skärmen.

- Klassen `FancyCustomer`, som även den utökar `Customer`, representerar extremt fina kunder.

  - När en `FancyCustomer` skapas ska namnet som skickas till konstrueraren ändras till adlig form, närmare bestämt ska "von" läggas till före namnet innan det lagras. Till exempel, om man i programmet skriver

    ```
    FancyCustomer fc = new FancyCustomer("Anka", 3000);
    ```

    ska kunden få namnet "von Anka".
  - En fin kund får alltid 20% rabatt, dvs om den fina kunden handlar för `amount` så ska förmögenheten bara ändras med 80% av `amount`. Dessa köp ska för övrigt ske precis som vanliga kunders köp.

5. **Priority queue**

Din uppgift är att skriva en prioritetskö, en behållare i vilket man kan stoppa in och ta ut objekt som tilldelats prioriteter. Uppgiften består av två delar.

a) Implementera ett gränssnitt för prioritetsköer. Det ska innehålla följande metoder: (1p)

- `void insert(Object obj, double prio)`, som tar som argument ett `Object` och dess prioritet (i form av ett `double`) och sätter in det i kön.
- `Object first()`, som returnerar en referens till det objekt i kön som har högst prioritet.
- `void removeFirst()`, som tar bort objektet med högst prioritet från kön.
- `boolean empty()` som returnerar `true` om, och endast om, kön är tom.

b) Skriv en klass för prioritetsköer som implementerar gränssnittet från första deluppgiften. Internt ska kön representeras som en länkad lista, med hjälp av den bifogade nodklassen nedan. Nodklassen ska alltså vara en intern klass i kön. (4p)

```
private class Node{
  double priority;
  Object data;
  Node next;

  public Node(double priority, Object data, Node next){
    this.priority = priority;
    this.data = data;
    this.next = next;
  }
}
```