TDDC77s Datorbaserad Tentamen

2016-01-12 kl 14:00-18.00

- Det är inte tillåtet att ha telefoner, böcker, icke tomma papper eller annat hjälpmaterial till hands under tentamen.
- Det är tillåtet att ha lösa tomma papper och pennor.
- Frågor om uppgifterna svaras endast via kommunikationsfönstret.
- Räck upp handen för att få hjälp om något inte fungerar i systemet
- För betyg 3 krävs godkänt på en av uppgifterna. För betyg 4 krävs två godkända uppgifter. För betyg 5 krävs tre godkända uppgifter.
- Det spelar **ingen roll** i vilken ordning du löser uppgifterna. Börja med den som verkar enklast för dig. Fortsätt med nästa uppgift så fort du har skickat en lösning.
- API:et kan öppnas med chrome under:
  `http://www.ida.liu.se/~TDDC77/extra/api-7/index.html`

Rättning:

- Steg 1: Vi kompilerar och provkör ditt program. Vi tittar på programkoden. Programmet kan bli godkänt (isf går vi till steg 2). Annars:
  - Ditt program får komplettering. E.g. vi har några testfall som programmet inte klarar, eller programmet uppfyller inte vissa krav. Du får kommentarer och möjligheten, om tiden tillåter det, att skicka ett nytt program.
  - Ditt program blir underkänt: programmet går inte att kompilera eller klarar inte de testfall som kommer med uppgiften. Du får inte möjlighet att skicka komplettering för den uppgiften. Du får fortsätta med nästa uppgift.
- Steg 2: Om uppgiften blev godkänd uppdateras strax ditt betyg.

1 Upprepnings avkodning

Skriv ett program som läser in sina argument från terminalen:

- Man anropar programmet med "`java RepetitionDecode r b1 ... bn`" där `r` är ett positivt udda heltal och `b1 ... bn` är en sekvens av heltal där varje `bi` är mellan 0 och 1 och `n%r` är noll (dvs. `n` är delbart med `r`).
- Programmet ska hantera alla situationer där användaren skriver in något som inte matchar beskrivningen ovan. I dessa fall ska programmet inte krascha utan terminera (utan errors) efter att ha skrivit ut:

```
Syntax: RepetitionDecode r b1 b2 ... bn
With  : r satisfies (r%2 == 1) and (r >= 1)
        n satisfies (n%r == 0) and (n >= 0), and
        bi satisfies (0 <= bi <= 1) for each i
E.g.  : RepetitionDecode 3 1 0 1 1 0 0
```

- Du ska använda dig av följande metod i klassen `Integer`:
  `public static int parseInt(String s) throws NumberFormatException`
- Antar att programmet får `r b1 ... bn` som argument. Programmet ska då räkna upp (och skriva ut) `m` heltal (där `m == (n/r)`) `o1, ... om`. För varje `i`, `oi` tilldelas `0` om majoritet bland alla `bj` där `(j/r) + 1 == i` är `0` (`oi` tilldelas `1` om majoritet är `1` istället)[1].
- Exempel på möjliga körningar:

```
$ java RepetitionDecode 1 1 0 1 0 0 1
Decoded message:  1 0 1 0 0 1

$ java RepetitionDecode 5 1 0 1 0 1
Decoded message:  1

$ java RepetitionDecode 5
Decoded message:

$ java RepetitionDecode 0 1 0 1 0 0 1
Syntax: RepetitionDecode r b1 b2 ... bn
With  : r satisfies (r%2 == 1) and (r >= 1)
        n satisfies (n%r == 0) and (n >= 0), and
        bi satisfies (0 <= bi <= 1) for each i
E.g.  : RepetitionDecode 3 1 0 1 1 0 0
```

[1] Upprepningskodning används i telekommunikation. En avsändare och en mottagare kan använda ett protokoll med ett specifikt repetitionsvärde "r". Avsändaren upprepar varje bit "r" gånger. Mottagaren kan korrigera eventuella transmissionsfel genom att räkna upp majoriteten (för varje original bit) för att härleda innehållet av det originella meddelandet. Till exempel kan avsändaren skickar "1 1 1 0 0 0" som kodning av "1 0" (här "r" är 3). Ert program avkodar meddelandet på mottagarens sida.

2 Pizza

Nedan följer ett antal faktapunkter. Din uppgift är att skapa den specificerade hierarki. Ange så pass mycket, MEN INTE MER, av alla klasser, gränssnitt (eng. interfaces), variabler, metoddeklarationer och implementationer så att allt som är specificerat i uppgiften implementeras. Din kod skall vara väl dokumenterad. Den ska dessutom respektera inkapsling och använda sig av lämpliga namn som tillämpar Java konventioner. Kopiera filen `Menu.java` som finns under `given_files/` till en ny mapp under din hemkatalog. Skapa och lägg till de Java-filer som behövs för din lösning. Man ska kunna kompilera dina filer och köra `Menu` för att få som output innehållet av `Menu.output.txt`. Skicka alla Java-filer.

Specifikationer:

- En `Pizza` är ett gränssnitt (eng. interface) med två metoder `int getPrice()` och `String getDescription()`.
- En `FlourPizza` är en klass som implementerar `Pizza` där `getPrice()` returnerar `30` och `getDescription()` returnerar `"flour"`.
- En `RyePizza` är en klass som implementerar `Pizza` där `getPrice()` returnerar `35` och `getDescription()` returnerar `"rye flour"`.
- En `PizzaDecorator` är en abstrakt klass som implementerar `Pizza` och som har en variabel som heter `pizza`. Variabeln är protected och har `Pizza` som typ.
- En `TomatoPizzaDecorator` är en klass som utökar (eng. extends) klassen `PizzaDecorator`. Klassens konstruktör (eng. constructor) tar ett `Pizza` objekt som argument och tilldelar den till variabeln `pizza`. Klassens metod `getPrice()` returnerar `12` plus `pizza`-variabelns-`getPrice()` medan dess metod `getDescription()` returnerar `pizza`-variabelns-`getDescription()` plus (eng. concatenated) med värdet `", tomato"`.
- En `MozzarellaPizzaDecorator` är en klass som utökar (eng. extends) klassen `PizzaDecorator`. Klassens konstruktör (eng. constructor) tar ett `Pizza` objekt som argument och tilldelar den till variabeln `pizza`. Klassens metod `getPrice()` returnerar `13` plus `pizza`-variabelns-`getPrice()` medan dess metod `getDescription()` returnerar `pizza`-variabelns-`getDescription()` plus (eng. concatenated) med värdet `", mozzarella"`.
- En `OlivePizzaDecorator` är en klass som utökar (extends) `PizzaDecorator`. Klassens konstruktör (eng. constructor) tar ett `Pizza` objekt som argument och tilldelar den till variabeln `pizza`. Klassens metod `getPrice()`returnerar `7` plus `pizza`-variabelns-`getPrice()` medan dess metod `getDescription()` returnerar `pizza`-variabelns-`getDescription()` plus (eng. concatenated) med värdet `", olive"`.

3 Fyra i rad

Kopiera filen `given_files/FourInARow.java` till en ny mapp under din hemkatalog. Vi ska implementera spelet "fyra i rad". Det är ett sällskapsspel för två personer. Varannan gång lägger en spelare en bricka i spelet. Den som först får fyra av sina brickor i rad (vertikalt, horisontalt eller diagonalt) vinner. Komplettera den givna koden enligt de anvisningar som finns i form av kommentarer märkta med "TODO: ...". Koden skall skrivas enligt Javas kodkonventioner och kommenteras väl. Koden ska kunna köras. En körning kan se ut så här:

```
 java FourInARow

| 0 0 0 0 0 0 0 0 0 | 6
| 0 0 0 0 0 0 0 0 0 | 5
| 0 0 0 0 0 0 0 0 0 | 4
| 0 0 0 0 0 0 0 0 0 | 3
| 0 0 0 0 0 0 0 0 0 | 2
| 0 0 0 0 0 0 0 0 0 | 1
----------------------
| 1 2 3 4 5 6 7 8 9 |

Player 1 Choose a non-full column between 1 and 9: q
Player 1 Choose a non-full column between 1 and 9: 0
Player 1 Choose a non-full column between 1 and 9: 1

| 0 0 0 0 0 0 0 0 0 | 6
| 0 0 0 0 0 0 0 0 0 | 5
| 0 0 0 0 0 0 0 0 0 | 4
| 0 0 0 0 0 0 0 0 0 | 3
| 0 0 0 0 0 0 0 0 0 | 2
| 1 0 0 0 0 0 0 0 0 | 1
----------------------
| 1 2 3 4 5 6 7 8 9 |

Player 2 Choose a non-full column between 1 and 9: 2

 ...

Player 1 Choose a non-full column between 1 and 9: 4

| 0 0 0 0 0 0 0 0 0 | 6
| 0 0 0 0 0 0 0 0 0 | 5
| 0 0 0 1 0 0 0 0 0 | 4
| 0 0 1 2 0 0 0 0 0 | 3
| 2 1 2 1 0 0 0 0 0 | 2
| 1 2 1 2 0 0 0 0 0 | 1
----------------------
| 1 2 3 4 5 6 7 8 9 |

Player 1 won !!
```