LUNDS TEKNISKA HÖGSKOLA Nätverksprogrammering 2005
Datavetenskap

EDA095

Tentamensskrivning — Nätverksprogrammering 2005-05-27, kl. 8-13

DEL 2 – Praktiska programmeringsuppgifter

Anvisningar

Tillåtna hjälpmedel för denna del av tentamen:

- Java snabbreferens.
- Kursboken: *Java Network Programming* 1st, 2nd eller 3rd edition, av Elliotte Rusty Harold eller *An Introduction to Network Programming with Java* av Jan Graba.
- Valfri lärobok i Java.
- Utdelat komplement till läroboken ("Kompletterande kursmaterial").
- Utskrift av OH-bilder från föreläsningarna - ej separata exempelprogram.

Denna tentamen i kursen Nätverksprogrammering består av två delar - en del som innehåller frågor av teoretisk/principiell/utredande karaktär och en del som innehåller praktiska programmeringsuppgifter. Detta är del 2. Den ska du ha erhållit tillsammans med ett färgat tentamensomslag när du lämnade in din lösning på del 1 av tentamen.

För godkänt betyg på tentamen krävs sammanlagt minst 20 poäng på tentamen, varav minst 8 poäng på vardera deltentamen. För högre betyg krävs naturligtvis mer, så gör så många uppgifter du kan.

Senast 2005-06-01 anslås på institutionens anslagstavla vilka som deltagit i tentamen men som, enligt institutionens noteringar, ännu inte redovisat laborationerna eller projektet. Deras skrivning rättas inte (och kan på sikt komma att annulleras) såvida inte rättelse skett eller dispens erhållits från ansvarig lärare. Rättelse skall göras senast 2005-06-15.

Uppgifterna nedan kan innehålla en hel del beskrivningar på vad som ska hända i olika specialfall. Om du inte hinner med eller vet hur du ska hantera alla dessa så är det bättre att du ignorerar dem och åstadkommer en lösning som fungerar i övrigt än att du inte löser uppgiften alls. En lösning kan ge mycket poäng även om alla specialfall inte är täckta.

1. Anslagstavla

I den här uppgiften skall du programmera en nätverksanslagstavla.

Anslagstavlan skall representeras av den (nästan) färdigskrivna klassen `BulletinBoard` på vilken man kan sätta upp anslag (lappar). Själva anslagstavlan är en vanlig heltalsmatris (se koden för `BulletinBoard`) där en ledig plats markeras med en nolla i matrisen.
Varje anslag (lapp) på anslagstavlan skall ha en ägare, en storlek (width, height) och en position (x, y*)* på tavlan (= övre vänstra hörnet på lappen). En lapp representeras av den färdigskrivna klassen `Notice`.

Nedan följer några osorterade punkter som programmet du skriver skall uppfylla:

1. Klienterna kopplar upp sig mot servern med hjälp av programmet *Telnet*. Användargränssnittet blir alltså helt textbaserat.
   OBS! Det medför också att inget speciellt klientprogram behöver skrivas.
2. När en klient kopplar upp sig får den anslagstavlan från servern i textform.
3. Flera klienter ska kunna vara igång samtidigt och nya klienter ska kunna ansluta när som helst.
4. Servern ska vara så robust att den överlever att en förbindelse kopplas ned oväntat. När en klient lämnar servern ska programmet rensa den klientens samtliga lappar.
5. Klienten skall kunna placera eller ta bort lappar. Man måste dock vara ägare för att få ta bort en lapp. För att få sätta upp en lapp på anslagstavlan måste servern verifiera att lappen inte överlappar andra lappar och att den får plats på anslagstavlans yta. Varje gång anslagstavlan uppdateras skall servern skicka den nya anslagstavlan till alla klienter.
6. Välj själv lämpliga ledtexter till klienter och hur de ska mata in var de vill placera sina lappar. Tänk dock på att *Telnet* skickar två tecken när man trycker på radslutstangenten – vagnretur (teckenkod 13 eller `'\r'`) och radframmatning (teckenkod 10 eller `'\n'`). Likaledes bör servern skicka dessa två tecken när man vill byta rad på utmatningen. Det är också möjligt att använda `flush()`.
7. För att underlätta hanteringen av själva anslagstavlan skall du använda klasserna `Notice` och `BulletinBoard` som bifogas på nästkommande sidor. Klassen `Notice` beskriver en lapp medan klassen `BulletinBoard` håller reda på var en lapp finns på anslagstavlan, sätter upp en lapp, tar bort en lapp och returnerar en sträng som representerar tavlans aktuella utseende och som är lämplig att skicka direkt till en Telnetklient.
8. För enkelhetens skull så består alla lappar bara av en massa tecken ’x’. Detta är inte fel utan det är för att hålla uppgiften på en rimlig nivå.
9. Klassen `BulletinBoard` är dock inte riktigt ”korrekt” för att klara uppgiften utan du behöver ändra i den på ett *fåtal* rader. Der finns inga småfel och du skall inte ändra koden för att den kan skrivas på ett snyggare sätt utan felen vi tänker på är mer av principiell art för att kunna lösa denna typ av uppgift. Ändringarna kan du göra direkt i koden på nästa sida.
   OBS! Glöm inte lämna in ändringarna av `BulletinBoard` tillsammans med dina andra lösningar.

(20p)

```
package bulletinboard;

public class BulletinBoard {
    private int[][] bb;                     //The actual board
    private int boardWidth, boardHeight;    //The board's sizes

    public BulletinBoard(int boardWidth, int boardHeight) {
        bb = new int[boardWidth][boardHeight];
        this.boardWidth = boardWidth;
        this.boardHeight = boardHeight;
    }

    // Removes a notice from the board: sets the positions to 0
    public boolean remove(Notice b) {
        if ((b.x + b.width <= this.boardWidth) &&
                (b.y + b.height <= this.boardHeight)) {
            for (int i = b.x; i < b.x + b.width; i++)
                for (int j = b.y; j < b.y + b.height; j++)
                    bb[i][j] = 0;
            return true;
        } else
            return false;
    }

    // Posts a notice on the board: sets the positions to the
   // ident value (identifier of the participant)
    public boolean post(Notice b) {
        if (available(b)) {
            for (int i = b.x; i < b.x + b.width; i++)
                for (int j = b.y; j < b.y + b.height; j++)
                    bb[i][j] = b.owner;
            return true;
        } else
            return false;
    }

    // Checks if the notice fits on the board surface and
    // does not overlap
    private boolean available(Notice b) {
        if (!((b.x + b.width <= this.boardWidth) &&
                (b.y + b.height <= this.boardHeight)))
            return false;
        for (int i = b.x; i < b.x + b.width; i++)
            for (int j = b.y; j < b.y + b.height; j++)
                if (bb[i][j] != 0)
                    return false;
        return true;
    }

    public String toString() {
        StringBuffer s = new StringBuffer();
        for (int j = 0; j < boardHeight; j++) {
            for (int i = 0; i < boardWidth; i++) {
                if (bb[i][j] == 0)
                    s.append('.');
                else
                    s.append('x');   //All content == 'x'...
            }
            s.append('\r');
            s.append('\n');
        }
        return s.toString();
    }
}
```

```
package bulletinboard;
/*
*Describes a notice on the board
 */
public class Notice {
    public int owner;   // The owner of the notice
    public int x;       // upper corner x
    public int y;       // upper corner y
    public int width;   // width of notice
    public int height;  // height of notice

    /** Creates a new instance of Notice */
    public Notice(int owner, int x, int y, int width, int height) {
        this.owner = owner;
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
    }

    public boolean equals(Notice b) {
        if ((this.owner == b.owner) && (this.x == b.x) &&
            (this.y == b.y) && (this.width == b.width) &&
            (this.height == b.height))
                 return true;
        return false;
    }

    public String toString() {
        StringBuffer s = new StringBuffer();
        s.append(owner);
        s.append(" ");
        s.append(x);
        s.append(" ");
        s.append(y);
        s.append(" ");
        s.append(width);
        s.append(" ");
        s.append(height);
        s.append(" ");
        return s.toString();
    }
}
```