Tentamen
Nätverksprogrammering
Lösningsförslag

2011–06–03, 14.00–19.00

Del 1

1. REST-ramverk använder HTTP metoder för att interagera med en server i en databasliknande kontext. Målet är att implementera CRUD- (create, retrieve, update, delete) kommandon med HTTP. En allmän mappning är Create: POST, Read (Retrieve): GET, Update: PUT och Delete: DELETE.
   Doodle har en API som tillåter att:
   1. skapa ett möte med POST
   2. hämta ett mötes data med GET
   3. ändra ett mötes egenskaper med PUT
   4. ta bort ett möte med DELETE

2. En XML/DOM-parser är en parser som bygger ett syntaxträd från ett XML dokument. DOM definierar standardiserade objek typer och ett Java-gränsnitt (API) för att axessa parse-trädet. DOM-modellen innehåller till exempel typen Document för att representera ett träd, Node för att representera en nod och Element för att representeratt XML-element. JAVA API:n innehåller klasser och metoder för att från ett XML dokument bygga ett träd av XML/DOM-objekt. JAVA API:n innehåller metoder för att traversera träd, för att manipulera noder, för att infoga noder och för att ta bort noder.

3. XSL/XSLT är ett XML-språk för att applicera transformationer på XML-dokument, till exempel för att transformera innehållet i en XML-databas till en XHTML presentation.
   XSL/XSLT är ett template-språk var ett program traverserar XML-trädet motsvarande dokumentet givet som indata. Man använder XSL/XSLT genom att skriva regler som matchar en nod (<xsl:template match ...>) i källdokument och som producerar en transformation från noden, till exempel till XHTML utdata. XSL/XSLT är ett komplett språk som innehåller loops, conditions, etc.

4. 1. Falskt (en dator kan mycket väl ha både en IPv4-adress och en IPv6-adress och paket av båda typerna kan skickas på samma nätverk)
   2. Falskt (TCP och UDP är snarare exempel på transportprotokoll)
   3. Sant
   4. Falskt (multicast är datagrambaserat och håller inte några fasta uppkopplingar öppna)
   5. Sant
   6. Sant

5. *Marshalling* innebär att data som ska skickas mellan klienten och servern vid ett RMI-anrop, dvs metodparametrar och/eller returvärden, konverteras från sin normala datarepresentation till ett externt (serialiserat) format som kan skickas över nätverket.

6. a) I operationen `pause` har Ida använt sig av en teknik kallad *busy-wait* för att vänta på att attributet `paused` ska få värdet false. Det innebär att programmet kommer att använda all tillgänglig CPU-kraft till att köra while-loopen i stället för att, som var avsikten, överlåta CPU-tiden åt andra trådar.

   b)
```
public class PauseControl {
    private boolean paused = false;

    public synchronized void setPause(boolean state) {
        paused = state;
        notifyAll();
    }

    public synchronized void pause() {
        while(paused) {
            try { wait(); } catch(InterruptedException e} { System.exit(1); }
        }
    }
}
```

7. a) Javascript
   b) CGI
   c) JSP
   d) PHP
   e) AJAX
   f) Servlets

8. a) Programmet läser till slutet på strömmen snarare än till nästa radslut. Slutet på strömmen kommer när strömmen stängs – i detta fall när Telnet-klienten kopplar ner – och det är ju uppenbart inte vad vi var ute efter. Slutet på strömmen upptäcker man genom att `read()` returnerar -1.

   b) Innan vi skriver den modifierade koden behöver vi bestämma oss för hur vi upptäcka och hantera radslut. Olika operativsystem och program markerar radslut på olika sätt. I Unix används t.ex. normalt tecknet linefeed med teckenkoden 10. I Windows – och i programmet Telnet oavsett operativsystem – används en sekvens av två tecken: carriage return (teckenkod 13) följt av linefeed (teckenkod 10). Om vi väljer den senare konventionen får vi också bestämma oss för hur vi vill göra ifall vi får in teckensekvenser som inte följer standarden, t.ex. ett ensamt carriage return eller linefeed. I programkoden nedan har vi bestämt oss för att rader antas avslutas med ett linefeed och eventuella carriage return ignoreras. På så sätt klara vi radslut à la Unix såväl som Windows/Telnet.

```
public String readLine(InputStream is) throws IOException {
    String result = new String();
    int ch = is.read();
    while (ch!=-1 && ch!=10) {
        if (result!=13) {
        result = result + (char) ch;
        }
        ch = is.read();
    }
    return result;
}
```

   En variant på lösningen är att i stället kapsla in strömmen i en `DataInputStream` och ersätta inläsningen med ett anrop av `readLine()`.

Del 2 – DownloadHelper

1. a) Den färdiga HTML-parsern i Java utgörs av den abstrakta inre klassen `HTMLEditorKit.Parser` vilket gör att ett parserobjekt inte kan skapas på normalt sätt. Det måste göras genom ett anrop av metoden `getParser()` i `HTMLEditorKit.Parser`. Det som ytterligare försvårar saken är att `getParser()` är deklarerad `protected` vilket gör att vi måste skapa en ny subklass till `HTMLEditorKit.Parser` med en metod deklarerad `public` som i sin tur anropar `getParser()` åt oss. Se koden nedan. För att erhålla ett parserobjekt skapar vi en instans av den nya klassen och anropar den publika metoden. Onödigt krångligt kan tyckas. Se boken sid 248 samt föreläsningsbilderna från föreläsningen om URL och HTML.

```
public class ParserGetter extends HTMLEditorKit {
   public HTMLEditorKit.Parser getParser() {
        return super.getParser();
    }
}
```

   b) Se sidan 248 och framåt i boken samt föreläsningsbilderna från föreläsningen om URL och HTML.

```
public class LinkExtractor extends HTMLEditorKit.ParserCallback {

    private URL baseURL;
    private URL startURL;
    private List<URL> urlList;

    public LinkExtractor(URL startURL) {
        this.startURL = startURL;
        this.baseURL = this.startURL;
        this.urlList = new ArrayList<URL>();
    }

    public void handleStartTag(HTML.Tag tag, MutableAttributeSet attributes, int position) {
        if (tag == HTML.Tag.A) {
            String href = (String) attributes.getAttribute(HTML.Attribute.HREF);
            try {
                urlList.add(new URL(baseURL, href));
            } catch (MalformedURLException e) {
                //e.printStackTrace();
            }
        }
        if (tag == HTML.Tag.FRAME) {
            String href = (String) attributes.getAttribute(HTML.Attribute.SRC);
            try {
                urlList.add(new URL(baseURL, href));
            } catch (MalformedURLException e) {
                //e.printStackTrace();
            }
        }
    }

    public void handleSimpleTag(HTML.Tag t, MutableAttributeSet a, int pos) {
        if (t == HTML.Tag.BASE) {
            String href = (String) a.getAttribute(HTML.Attribute.HREF);
            try {
                baseURL = new URL(href);
            } catch (Exception e) {
            }
        }
    }
```

```
    public List<URL> getUrlList() {
        return urlList;
    }
}
```

c) Man skulle naturligtvis kunna skriva allt nedan i en och samma main-metod, men för överskådlighetens skull har vi brutit ned koden i mindre metoder. Lösningen nedan innehåller dessutom lite extra finesser, såsom att kontrollera att sidan vi läser faktiskt innehåller text och HTML-kod, som inte krävdes i uppgiften.

```
public class Nedladdare {
    String targetURLString;
    String destFolder;
    List<URL> linkList;

    public Nedladdare(String targetURL, String destFolder) {
        this.targetURLString = targetURL;
        this.destFolder = destFolder;
    }

    public List<URL> extractLinksFromURL(URL url) {
        LinkExtractor callback = null;

        try {
            URLConnection uc = url.openConnection();
            String type = uc.getContentType().toLowerCase();
            if ((type != null) && !type.startsWith("text/html")) {
                System.out.println(url + " ignored. Type " + type);
                return null;
            }

            ParserGetter kit = new ParserGetter();
            HTMLEditorKit.Parser parser = kit.getParser();

            callback = new LinkExtractor(url);

            InputStream in = new BufferedInputStream(uc.getInputStream());
            InputStreamReader r = new InputStreamReader(in);
            parser.parse(r, callback, true);

        } catch (IOException e) {
            System.out.println("Could not read: " + url);
            //e.printStackTrace();
            return null;
        } catch (Exception e) {
            System.out.println("Could not read: " + url);
            return null;
        }
        return callback.getUrlList();
    }

    public void execute() throws MalformedURLException {
        URL targetURL = new URL(targetURLString);
        linkList = extractLinksFromURL(targetURL);
        String pdfFileName;
        FileOutputStream fwriter;
        int inx;
        byte [] buffer = new byte[256];
```

```
            try {
                for (URL link : linkList) {
                    if (link.toString().endsWith(".pdf")) {
                        System.out.println("Reading link: " + link);
                        inx = link.toString().lastIndexOf("/", link.toString().length());
                        pdfFileName = link.toString().substring(inx);
                        System.out.println("Writing file: " + pdfFileName);
                        fwriter = new FileOutputStream(destFolder + pdfFileName);
                        InputStream is = link.openStream();
                        int cnt;
                        while ((cnt = is.read(buffer, 0, 256)) != -1) {
                            fwriter.write(buffer, 0, cnt);
                        }
                    }
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        public static void main(String[] args) throws MalformedURLException {
            Nedladdare ned;
            if (args.length == 2) {
                ned = new Nedladdare(args[0], args[1]);
            } else {
               System.out.println("Usage: java Nedladdare TargetURL DestinationFolder");
                return;
            }
            ned.execute();
        }
    }
```

2. En enkeltrådad version av programmet kommer att spendera mycket tid åt att vänta på nätverksrelaterade saker såsom att uppkopplingar till webbservrar ska ske eller att data ska anlända från servern. Genom att göra programmet flertrådat kan denna tid ägnas åt att parallellt behandla andra uppkopplingar där data finns tillgängligt. På detta sätt kan den totala tiden för nedladdning minskas betydligt.

3.
```
public class Runner extends Thread {
    Nedladdare ned;
    String destFolder;

    public Runner(String destFolder, Nedladdare ned) {
        super(Integer.toString(name));
        this.destFolder = destFolder;
        this.ned = ned;
    }

    public void run() {
        String pdfFileName;
        FileOutputStream fwriter;
        int inx;
        byte[] buffer = new byte[256];
        URL link;
        while (true) {
            if ((link = ned.returnHead()) == null) {
                return;
            }
```

```
                try {
                    if (link.toString().endsWith(".pdf")) {
                        inx = link.toString().lastIndexOf("/", link.toString().length());
                        pdfFileName = link.toString().substring(inx);
                        fwriter = new FileOutputStream(destFolder + pdfFileName);
                        InputStream is = link.openStream();
                        int cnt;
                        while ((cnt = is.read(buffer, 0, 256)) != -1) {
                            fwriter.write(buffer, 0, cnt);
                        }
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
```

4.
```
    public void execute() throws MalformedURLException {
        URL targetURL = new URL(targetURLString);
        linkList = extractLinksFromURL(targetURL);
        for (int i = 0; i < threadCnt; i++) {
            new Runner(i, destFolder, this).start();
        }
    }
```

Med ovanstående Runner-version behöver vi dessutom lägga till en synchronized-metod i klassen `Nedladdare`. Det behövs för att inte två trådar ska kunna hämta samma PDF-fil. Andra lösningar på detta problem är tänkbara, t.ex. att alla trådarna direkt har tillgång till URL-listan. I så fall måste man dock se till att välja en datastruktur för listan som är trådsäker, dvs att flera trådar samtidigt ska kunna försöka ta ut ett element ur listan utan att det blir problem. Om denna funktionalitet implementerats under föregående deluppgift räknas den poängmässigt ändå till denna deluppgift.

```
    public synchronized URL returnHead() {
        if (!linkList.isEmpty()) {
            return linkList.remove(0);
        } else {
            return null;
        }
    }
```