Mittuniversitetet
MID SWEDEN UNIVERSITY

Inlämningsuppgift 8

Java för C++-programmerare, 7,5 hp

Syfte:	Att kunna skriva en flertrådad klient-/server-lösning som kommunicerar med varandra genom en Socket.
Att läsa:	Lektion 9 - 10
Uppgifter:	3 (varav 2 frivilliga)
Inlämning:	Inlämningslåda 8 i Moodle

Uppgift 1

I denna uppgift ska du utgå från din lösning på inlämningsuppgift 7. Skapa ett nytt Java-projekt i Eclipse som är en kopia på din tidigare lösning. Dina klasser ska tillhöra paketet: `dt062g.studentid.assignment8` där studentid är ditt användarnamn i studentprotalen/Moodle.

Klasser som använder eller representerar ett grafiskt användargränssnitt ska baseras på Swing. `System.in` får inte användas i dessa klasser för inmatningar från användaren. `System.out` och `System.err` får endast användas för debug- respektive felmeddelanden. Du kan inte räkna med att användaren av programmet ser dessa meddelanden.

Varje klass och gränssnitt (interface) som du skapar ska dokumenteras med en dokumentationskommentar för klassen enligt:

```
/**
* A short description (in Swedish or English) of the class.
*
* @author  Your Name (your student id)
* @version 1.0
* @since   yyyy-mm-dd (last edited)
*/
```

Du ska nu skapa en klient- och serverlösning av ditt ritprogram så att det från ritprogrammet (klienten) går att hämta och spara teckningar (XML-filer) från/till servern.

Du ska börja med att skriva serverdelen och dess klasser, för att sen skriva klienten och till sist anpassa din JFrame så att den kan använda din klient. Alla klasser som servern består av ska tillhöra paketet: `dt062g.studentid.assignment8.server`. Alla klasser som klienten består av ska tillhöra paketet: `dt062g.studentid.assignment8.client`. Här är studentid ditt användarnamn i studentprotalen/Moodle.

Server

Börja med att skapa en ny klass som heter `Server`. Du får själv välja om servern ska ha ett grafiskt användargränssnitt eller om det ska vara konsolbaserat. Vid konsolbaserat gränssnitt är det ok att meddelanden till användaren görs med `System.out` och `System.err`. Denna klass ska:

- Ha en `main`-metod. I `main`-metoden ska du kontrollera om antalet argument (`args.length`) som ges till metoden är fler än 0. Om så är fallet ska du försöka använda det första argumentet som den port servern ska lyssna på. Anges inget argument eller om argumentet inte kan konverteras till en `int`, ska servern lyssna på porten 10000 (porten kan komma att ändras i en annan inlämningsuppgift).
- Använda en `ServerSocket` för att vänta på klienter som ansluter till servern.

- Starta en ny tråd för varje ansluten klient. I denna tråd (se beskrivning nedan) ska alla kommunikation mellan klienten och servern ske.
- Servern ska kunna hantera flera klienter samtidigt.
- Visa ett meddelande om att servern är startad och vilken port den lyssnar på.
- För varje ansluten klient ska ett meddelande visas om att en klient har anslutit sig och från vilken adress.
- När kommunikationen med en klient avbryts på något sätt ska ett meddelande visas om att förbindelsen med klienten är avbruten.

```
Server started on port 10000
New client connected from 127.0.0.1:50081
New client connected from 127.0.0.1:50125
Client from 127.0.0.1:50081 has disconnected
New client connected from 127.0.0.1:50163
Client from 127.0.0.1:50163 has disconnected
Client from 127.0.0.1:50125 has disconnected
```

ClientHandler

Denna klass ska ärva från klassen `Thread`. Klassen ska:

- Ha en instansvariabel av typen `Socket`. Det är via denna socket som kommunikation till/från klienten sker.
- Ha en konstruktor som tar ett `Socket`-objekt som argument. Denna socket ska tilldelas den ovan.
- Överskugga metoden `run` från klassen `Thread`. I denna metod ska all kommunikation till/från klienten ske. Det är även här som filer överförs. Det är ok att denna metod anropar andra metoder i klassen.
- Visa lämpliga meddelanden om vad som sker i kommunikationen mellan klienten och servern. Det kan t.ex. vara att klienten begär att få en lista över serverns filer. Se exempel på utskrifter nedan.
- Andra lämpliga instansvariabler och metoder du anser behövs.

```
Server started on port 10000
New client connected from 127.0.0.1:52124
Command 'list' received from 127.0.0.1:52124
Sending list of files to 127.0.0.1:52124
Client from 127.0.0.1:52124 has disconnected
New client connected from 127.0.0.1:52125
Command 'load' received from 127.0.0.1:52125
Size of 'Circles by Robert.xml' is 1600 bytes
Sending file to 127.0.0.1:52125
File sent to 127.0.0.1:52125
Client from 127.0.0.1:52125 has disconnected
New client connected from 127.0.0.1:52126
Command 'save' received from 127.0.0.1:52126
Receiving 'Circles by Robert (copy).xml' from 127.0.0.1:52126
File received from 127.0.0.1:52126
Client from 127.0.0.1:52126 has disconnected
New client connected from 127.0.0.1:52127
Command 'list' received from 127.0.0.1:52127
```

```
Sending list of files to 127.0.0.1:52127
Client from 127.0.0.1:52127 has disconnected
```

Hur kommunikationen mellan servern och klienten ska gå till är upp till dig att bestämma. Lämpligt kan vara att klienten skickar kommandon (`String`) på vad som ska ske (list, load och save) och att du i `run`-metoden anpassar kommunikationen utifrån dessa kommandon. Ett kommando kan bestå av flera operationer. T.ex. skulle save-kommandot kunna bestå av följande operationer:

- Klienten skickar kommandot save
- Servern tar emot kommandot
- Klienten skickar namnet på filen som ska sparas
- Servern tar emot filnamnet
- Klienten skickar filens storlek (antal bytes)
- Servern tar emot filens storlek
- Klienten skickar filens innehåll (konvertera filen till en array av `byte`)
- Servern tar emot filinnehållet och sparar till en fil på serverns hårddisk enligt det filnamn som tidigare tagits emot (det är ok att skriva över existerande filer med samma namn).

Du ska använda `DataInputStream` och `DataOutputStream` för att skicka kommandon och filer med mera på både server- och klientsidan. Viktigt att du stänger dessa strömmar genom att anropa `close` så snart du inte behöver dem längre. Välj själv vilka strömmar du vill använda för att läsa/skriva filer från/till hårddisk. Viktigt att du om möjligt nästlar strömmarna ovan i en buffrad-ström av något slag.

När klienten och servern är färdig med ett kommando är det lämpligt att avsluta förbindelsen genom att anropa `close` på alla öppna strömmar och sen den `Socket` som används. När det är dags att utföra ett nytt kommando får klienten ansluta på nytt till servern.

Servern ska lagra sina XML-filer i en katalog som heter xml. I denna katalog kan det finnas filer med en filändelse annan än .xml. Du måste filtrera med en `FilenameFilter` så att endast XML-filerna i denna katalog skickas till klienten.

Client

Du ska nu skriva en klass som sköter kommunikationen med servern. Denna klass ska du döpa till `Client`. Klassen ska kunna användas helt separat, utan inblandning av övriga klasser ditt ritprogram består av, för att kommunicera med servern enligt beskrivningen av klasserna `Server` och `ClientHandler`. Tillsammans med denna beskrivning följer klassen `ClientTest`. Du ska skriva din klient så att den kan användas av testklassen utan några förändringar.

Klassen ska minst ha följande:

- En instansvariabel av typen `String` för att lagra adressen till servern.
- En instansvariabel av typen `int` för att lagra vilken port servern lyssnar på.

- En konstruktor som tar en adress (`String`) och port (`int`) som argument. Dessa ska användas vid anslutning till servern.
- En konstruktor utan adress som använder standardvärdena ”localhost” för address och 10000 för port.
- En metod `connect` som ansluter till servern (skapar en Socket till servern och eventuellt de strömmar som behövs för kommunikation). Om metoden anropas och en anslutning redan är etablerad ska ingen ny anslutning göras. Metoden ska returnera `true` om socket/strömmar kunde skapas och `false` om dessa inte kunde skapas.
- En metod `disconnect` som avslutar förbindelsen med servern (stänger och sätter till `null` den socket och eventuella strömmar som används).
- En metod `getFilenamesFromServer` som returnerar en array av strängar (`String[]`) som innehåller filnamnet på alla XML-filer på servern. Metoden ska börja med att ansluta till servern och avsluta med att avbryta anslutningen till servern (innan return görs). Låt metoden returnera `null` om något fel uppstod. Annars returnerar du en array av `String` av längden 0 (noll) om servern inte har några filer eller en array av `String` som innehåller alla filer på servern.
- En metod `getFileFromServer` som tar en sträng som argument. Denna sträng innehåller namnet på den fil som ska hämtas från servern. Metoden ska börja med att ansluta till servern och avsluta med att avbryta anslutningen till servern (innan return görs). Med hämtas menas att filen ska kopieras från xml-mappen på servern till klientens hårddisk (i valfri mapp, dock inte samma som servern). Metoden ska returnera en sträng som innehåller hela sökvägen (på klientens hårddisk) till den fil som kopierats (ex: ”c:\tmp\Mona Lisa.xml”). Om något fel uppstår ska metoden returnera `null`. Om filen inte existerar på servern ska en tom sträng med längden 0 (noll) returneras (`""`).
- En metod `saveAsFileToServer` som tar två strängar som argument. Den första strängen innehåller namnet på en lokal fil som ska skickas till servern för att sparas där. Den andra strängen innehåller vilket namn filen ska sparas på servern. Metoden ska börja med att ansluta till servern och avsluta med att avbryta anslutningen till servern (innan return görs). Med skickas/sparas menas att filen ska kopieras från klientens hårddisk till xml-mappen på serverns hårddisk. Metoden ska returnera `true` om allt gick bra och `false` om något fel uppstod.
- En metod `saveFileToServer` som tar en sträng som argument. Denna sträng innehåller namnet på en lokal fil som ska skickas till servern för att sparas där. På servern ska filen sparas under samma namn som den ursprungliga filen. I övrigt ska denna metod fungera på samma sätt som `saveAsFileToServer`.

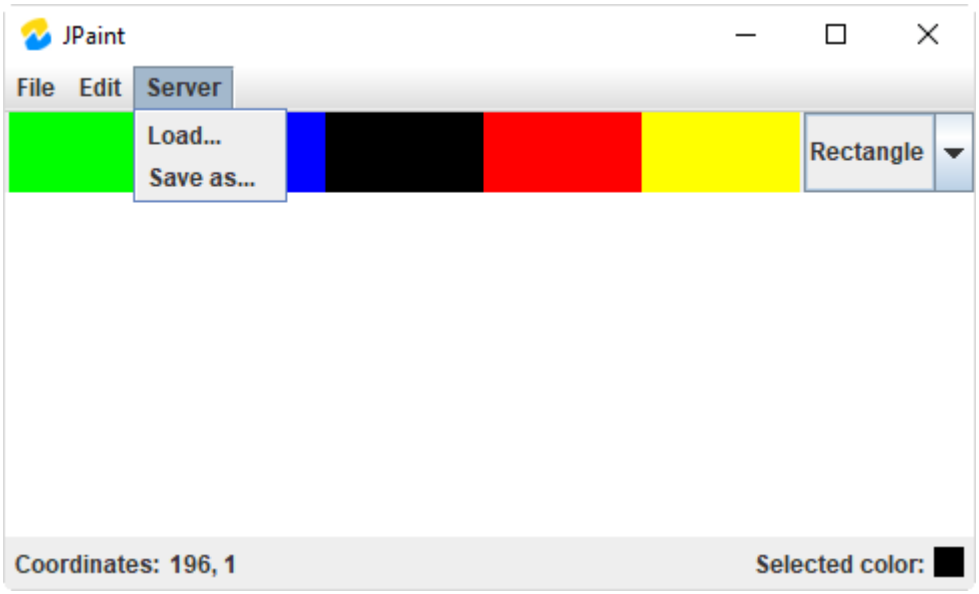
Det är ok att all kod i `Client` exekveras i huvudtråden (main thread). Låt det vara upp till den kod som använder `Client` att se till att metoderna som anropas utförs i en bakgrundstråd. Framför allt i en applikation med ett grafiskt användargränssnitt. Det är även ok att `ClientTest`, som använder `Client`, inte använder bakgrundstrådar.

Nu är du klar, bra jobbat! Om du vill kan du fortsätt med de frivilliga uppgifterna på följande sidor.

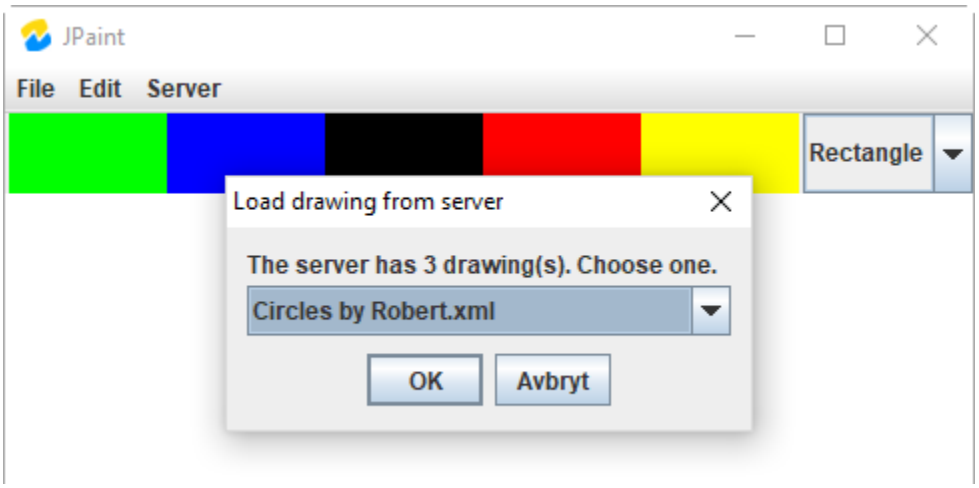
Frivillig extrauppgift 1

JFrame

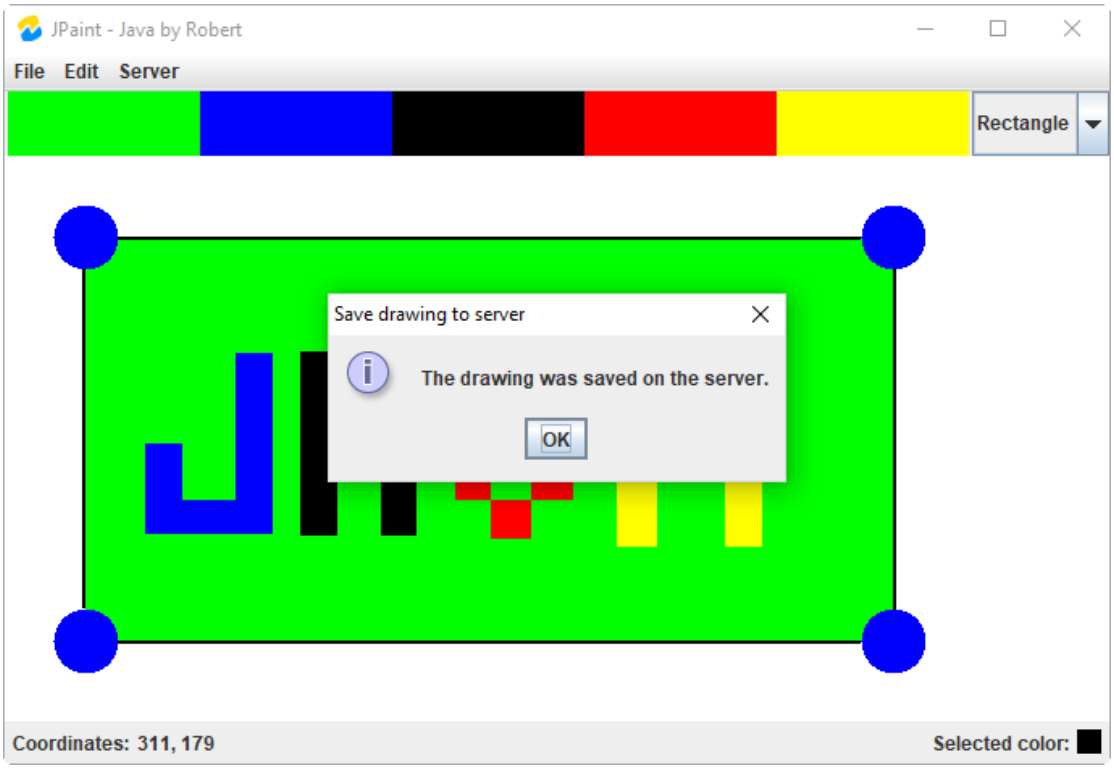
Ditt ritprogram ska skrivas om så att den kan kommunicera med din server. Din `JFrame` ska använda klassen `Client` för att lista, ladda och spara filer. Börja med att lägga till en meny med namnet Server i vilken menyalternativen Load… och Save as… finns med.



När menyalternativet Server | Load… väljs ska klienten ansluta till servern och begära att få en lista över alla filer servern har (endast XML-filer ska returneras av servern). Namnen på alla filer från servern ska visas i klienten som en valbar lista i en dialogruta (se exempel i How to make dialogs i The Java Tutorials från Oracle). När användaren har valt en fil i listan ska denna fil hämtas/kopieras från servern till klienten. Ritytan i fönstret ska uppdateras för att visa den hämtade filen.



När menyalternativet Server | Save as… väljs ska aktuell `Drawing` i ritytan sparas till servern med det filnamn som användaren skriver in i dialogrutan. I dialogrutan ska ett namnförslag visas precis som för menyalternativet File | Save as… När användaren klickar på ok ska aktuell teckning sparas som en fil på servern (det är ok att lagra filen lokalt på hårddisken först).



Tänk på att kommunikation till/från servern INTE får ske på händelsetråden (the event dispatch thread) då kommunikation klassas som ”långa operationer” (long-running tasks) som kan blockera andra händelser som uppstår i programmet (musklick med mera). Använd förslagsvis en `SwingWorker` för varje ”kommando” som utförs mot servern.

Assignment8

Denna klass finns bifogad med uppgiftsbeskrivning. Klassen används för att skapa och visa din `JFrame`. Du får själv ändra i koden så att rätt namn på klassen används. Börja med att ändra i `main`-metoden så att du kontrollerar om det finns ett eller flera argument (`args.length`). Om så är fallet ska du använda första argumentet som adress till servern. Ett eventuellt andra argument ska du använda som den port servern lyssnar på.

Anges inga argument, eller om det andra argumentet inte går att konvertera till ett tal, ska klienten använda adressen ”localhost” eller ”127.0.0.1” (vilken som fungerar bäst i ditt system) och porten ska vara 10000 (porten kan komma att ändras i en annan inlämningsuppgift).

Frivillig extrauppgift 2

Du ska skapa körbara jar-filer för din lösning. En jar-fil för att starta servern och en jar-fil för att starta clienten (din JFrame ska visas). Döp dessa till server.jar respektive client.jar. Inkludera dessa i din zip-fil som du lämnar in.