DUGGA: Objektorienterade applikationer

Läs detta!

- *Uppgifterna är inte avsiktligt ordnade efter svårighetsgrad.*
- **Skriv rent dina svar. Oläsliga svar r ä t t a s e j!**
- Programmen skall skrivas i Java 5, eller senare version, vara indenterade och renskrivna och i övrigt vara utformade enligt de principer som lärts ut i kursen.
- Onödigt komplicerade lösningar ger poängavdrag.
- Givna deklarationer, parameterlistor, etc. får ej ändras, såvida inte annat sägs i uppgiften.
- Läs igenom tesen och förbered ev. frågor.

I en uppgift som består av flera delar får du använda dig av funktioner klasser etc. från tidigare deluppgifter, även om du inte löst dessa.

Lycka till!

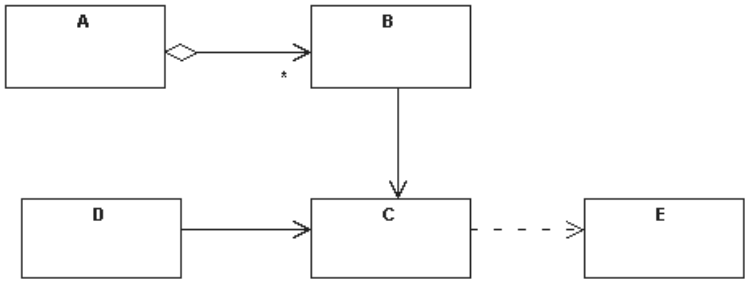
Uppgift 1 Modellering

a) Rita ett UML-diagram som motsvarar följande Java-kod. För poäng krävs att rätt piltyper används.

```
public interface A { ... }
public interface B extends A { ... }
public abstract class C implements B { ... }
public class D implements A { ... }
public class E extends D { ... }
```

(2 p)

b) Skriv (minimala) klassdeklarationer i Java för klasserna A-C som motsvarar följande UML-diagram. Ge exempel på hur instansvariabler, konstruktorer och metoder kan lagra och skapa relationerna som visas i diagrammet.



(3 p)

Uppgift 2 Aktiva objekt och trådar

Följande klass används för att lagra bankkonton:

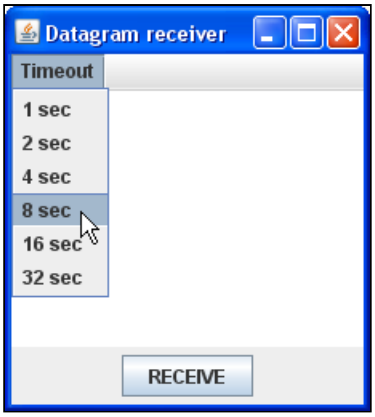
```
public class Account {
    private int balance;

    public Account(int balance) {
        this.balance = balance;
    }
    public void transaction(int amount) { // > 0 = ins., < 0 = uttag
        int newBalance = balance + amount;
        if ( newBalance >= 0 )
            balance = newBalance;
    }
}
```

Som vi såg i föreläsningen om trådar så är klassen osäker om flera trådar samtidigt anropar `transaction`. Skriv om `transaction` så att den blir säkrare. Utnyttja javas möjligheter till synkronisering. Om `transaction` anropas med ett uttagsbelopp som överskrider saldot, så skall metoden returnera först när tillräckligt belopp satts in på kontot (av en annan tråd).

(5 p)

Uppgift 3 Grafiska gränssnitt

När man skapar ett objekt av klassen `Gui` skall fönster med följande utseende visas:



I fönstret skall finnas en meny, ett textfält av typen `JTextArea` med 10 rader och 20 tecken per rad, samt en knapp. I menyn skall man kunna välja olika tider. När man trycker på knappen skall text skrivas i textarean. Textfältet skall ej vara redigerbart för användaren. Texten skall hämtas med hjälp av klassen `DatagramReceiver`. Vi skall återkomma till denna klass i en senare uppgift, men här kan du anta att den är färdig:

```
public class DatagramReceiver {
    public DatagramReceiver(int port)

    /**
     * receive  Tries to receive a datagram via the port
     *          stored in this object.
     * @param timeout  The maximum time in milliseconds to wait
     *          for a datagram.
     * @return  If a datagram is received within
     *          the specified timeout,
     *          its text contents is returned, ow null.
     */
    public String receive(int timeout)
}
```

Klassen är delvis färdig. Uppgiften blir att konstruera metoden `makeFrame`, samt ev. ytterligare hjälpmetod om du tycker att det behövs. `makeFrame` skall skapa fönstret. Knappen skall ha en lyssnare som tar emot ett datagram och visar innehållet i textfönstret. Om inget datagram kommer efter den inställda tiden skall texten `No message received` visas i textfönstret.

```
public class Gui extends JFrame {
    private final int receivePort = 1234;
    private int timeout = 1000;  // ms (default)
    private JTextArea textArea;
    private DatagramReceiver datagramReceiver;

    public Gui(){
        makeFrame();
        datagramReceiver = new DatagramReceiver(receivePort);
    }

    private void makeFrame() { // Skriv denna! }

    private void makeMenuBar() { // färdig metod }
 }
```

(5 p)

Uppgift 4 Kommunikation

Klassen `DatagramReceiver` tar emot datagram. Klassen skall ha metoden `receive` som tar emot ett datagram via ett angivet portnummer. Metoden skall ha en heltalsparameter som anger hur länge (millisekunder) den som mest får vänta på datagrammet. Om det tas emot inom den angivna tiden skall paketets textinnehåll returneras, annars returneras **`null`**. Om något undantag kastas skall det fångas och **`null`** returneras.

Ex.

```
DatagramReceiver r = new DatagramReceiver(1234); // port 1234
String msg = r.receive(10000);  // timeout efter 10 sek.
```

Skriv färdigt konstruktorn och metoden `receive`:

```
public class DatagramReceiver {
    private final int BUFFSIZE = 1024;
    private DatagramSocket socket;
    private DatagramPacket packet;

    public DatagramReceiver(int port) { // skriv denna! }
    public String receive(int timeout) { // skriv denna! }
}
```

(5 p)

Uppgift 5 Strömmar och filer

I en binärfil lagras temperaturdata från ett antal mätstationer. Först i filen finns ett tal av typen `long` som anger antalet mätvärden i filen. För varje mätvärde lagras två tal: Mätstationens idnummer som ett heltal av typen `int`, och därefter temperaturvärdet som ett flyttal av typen `float`. Uppgiften är att implementera metoden

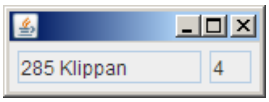
```
public static Map<Integer,Float> readMaxTemperatures(String filename)
```

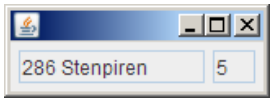
Den skall läsa en fil av ovanstående typ, samt i en tabell (map) för varje mätpunkt lagra dess högsta temperaturvärde. Tabellen skall returneras. Om något fel inträffar vid läsningen skall metoden returnera `null`.

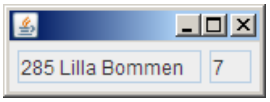
(5 p)

Uppgift 6 Designmönster

Nedan visas några skärmbilder från ett GUI för en applikation som visar hållplatsinformation (en liknande skärm finns på Lindholmspiren). I det vänstra textfältet visas linje och destination och i det högra tid till nästa avgång.







Programmet innehåller tre klasser: `Line`, `Time` och `Gui`. `Line` uppdaterar `Gui` med aktuell linje och destination så snart den förra blivit inaktuell. `Time` uppdaterar `Gui` med tid till nästa avgång varje minut. Båda skickar strängar. Klasserna `Main` och `Gui` är delvis färdiga, men det återstår en sak att göra, nämligen att tillämpa designmönstret Observer så att `Gui` blir observatör till `Line` och `Time`. Vid uppdatering från modellklasserna med ny information skall denna skrivas i resp. textfält. `Line` och `Time` är givna och ingår ej i uppgiften. Komplettera koden vid A, B resp. C med lämpliga arv, metoder och metodanrop!

```
public class Gui extends JFrame A
{
    private JTextField lineField,timeField;

    public Gui() {
        // bygger fönstret (denna metod är färdig)
    }

    B
}

public class Main {
    public static void main(String[] arg) {
        Time time = new Time();
        Line line = new Line();
        Gui gui = new Gui();

        C
    }
}
```

(5 p)