TENTAMEN: Objektorienterade applikationer

Läs detta!

- *Uppgifterna är inte avsiktligt ordnade efter svårighetsgrad.*
- Börja varje uppgift på ett nytt blad.
- Skriv ditt idnummer på varje blad (så att vi inte slarvar bort dem).
- **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.
- Programkod som finns i tentamenstesen behöver ej upprepas.
- Givna deklarationer, parameterlistor, etc. får ej ändras, såvida inte annat sägs i uppgiften.
- Läs igenom tentamenstesen 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

Designen av nedanstående klasser har bristen att `Clock` beror av gränssnittsklasserna `Display` och `Console`. Det borde vara tvärtom. Komplettera koden på nästa sida så att designmönstret Observer realiseras. Det räcker om du visar mönstret för klasserna `Clock` och `Display` (`Console` blir analogt).

```
public class Clock {
    private Time time;
    Display display;
    Console console;

    public void tick() {
        time++;
        display.show( time );
        console.print( time );
    }
    ...
}

public class Display {
    public show( Time time ) { ... }
    ...
}

public class Console {
    public print( Time time ) { ... }
    ...
}
```

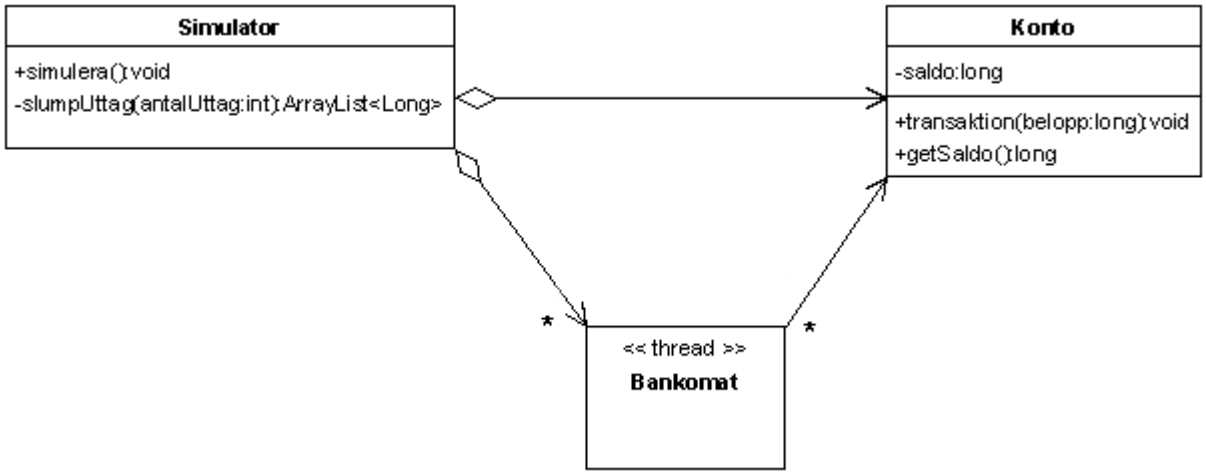
forts.

```
public class Clock
{
    private Time time;

    public void tick() {
        time++;


    }




}

public class Display
{




    public Display( Clock clock ) {



    }

    private show( Time time ) { ... }









}
```

(8 p)

Uppgift 2

En liten simulator för ett bankomatsystem har följande arkitektur



Simuleringen går ut på att testa om många parallella bankomattrådar uppdaterar ett konto korrekt. Varje bankomat gör tio slumpmässiga uttag mellan 500 och 1000 kronor från kontot. Om man på förhand summerar dessa slumpmässiga uttag för, säg, 1000 bankomater och sätter saldot på kontot till denna summa, så skall saldot vara exakt 0 kr efter avslutad simulering. Metoden `slumpUttag` i simulatorn är färdig. Den ger tillbaks en lista med `antalUttag` negativa tal (=uttag).

a) Implementera klassen `Konto`. Positivt belopp betyder insättning, negativt uttag. Saldot får bli negativt. För full poäng krävs att du beaktar att metoden `transaktion` kan anropas av flera trådar samtidigt.

```
public class Konto {




}
```

(4 p)

b) Implementera klassen `Bankomat`. Varje objekt av klassen skall exekvera i en egen tråd. När tråden exekveras skall uttagen i en uttagssekvens utföras mot ett konto. Förmedla kontot och uttagssekvensen från simulatorn till bankomatobjektet på lämpligt sätt.

```
public class Bankomat
{




}
```

(4 p)

v.g.v.

c) Skriv klart klassen `Simulator`. Lagra lämpligen bankomatobjekten i ett fält. Metoden simulera skall starta bankomattrådarna, invänta dem, och därefter skriva ut saldot på kontot. Utskriften skall exempelvis se ut så här :

```
Startsaldo: 7480664
Slutsaldo: 0
```

```
public class Simulator {
    private static int antalBankomater = 1000;
    private static int antalUttag = 10; // antal uttag per bankomat
    private static long startSaldo = 0;









    public Simulator() {









}


// Skapar en lista av slumpmässiga uttagsbelopp
private ArrayList<Long> slumpUttag(int antalUttag) {
    Random rand = new Random();
    ArrayList<Long> l = new ArrayList<Long>(antalUttag);
    for ( int i = 0; i < antalUttag; i++ ) {
        int belopp = 500 + rand.nextInt(500);
        startSaldo += belopp;
        l.add((long)-belopp);
    }
    return l;
}
```

v.g.v.

```
        public void simulera() {

        }
    }
```

(8 p)

Uppgift 3

Java tillåter som bekant att klasser som inte är kända då programmet kompileras ändå kan laddas dynamiskt under exekvering. I det här exemplet implementerar dessa klasser gränssnittet

```
public interface Command {
    public void execute();
}
```

Skriv ett program som låter användaren mata in ett kommandonamn via en pop-up-dialog. Därefter skall kommandot exekveras genom att först ladda klassen med namnet i fråga, skapa ett objekt av den och slutligen anropa `execute`-metoden. För att kontrollera om ett inmatat kommandonamn är korrekt finns den färdiga klassen

```
public final class Constants {
      ...
      public static boolean isCommand(String s) { ... }
}
```

Om `isCommand(x)` returnerar `true` finns en klass med namnet x som implementerar `Command`. Om det inte finns något kommando som motsvarar den inmatade strängen skall ett meddelande visas.

(10 p)

Uppgift 4

I ett enkelt anslagstavelsystem kan användare sända in sina åsikter om vad som helst via klienter. Meddelandena skickas till en applikation som visar inkomna texter i ett fönster. Det finns en instans av mottagarapplikationen, men i princip är ett obegränsat antal klienter möjliga. Uppgiften går ut på att konstruera sändar- och mottagarklasserna i de två applikationerna.

Texterna som användarna skriver i sina klientfönster skall sändas som datagram till mottagarapplikationen på adress 123.45.67.89 och port nr 9867. Här följer skelett till klientapplikationens sändarklass. *Komplettera nedan!*

```
public class MessageSender {



    public MessageSender(String adress, int port) throws SocketException {




    }

    public void send(String text) { // skicka meddelandet









   }
}
```

v.g.v.

Mottagarapplikationens klass sköter mottagning och visning av meddelanden. *Komplettera!*

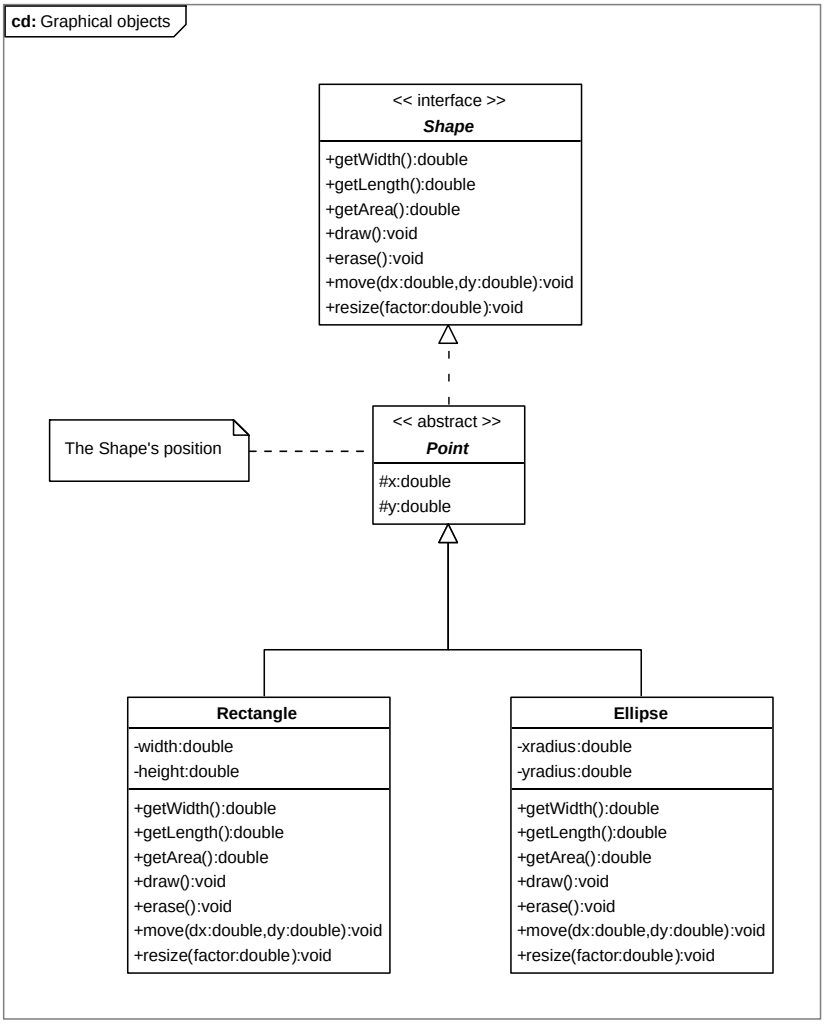
```
public class MessageReceiver {
    private JTextArea textDisplay;  // för utskrift i GUI:t




    public MessageReceiver(int port) {


    }

    public void receiveMessages() throws SocketException {









    }

    // anropas av main
    public void setTextDisplay(JTextArea textDisplay) {
        this.textDisplay = textDisplay;
    }
}
```

I uppgiften ingår *inte* att programmera de grafiska användargränssnitten.

(16 p)

Uppgift 5

Antag att man har tillgång till följande klasser för att representera geometriska figurer



Klasserna är givna och får inte ändras. Antag att man vill kunna addera en tredje dimension till rektangel- och ellipsobjekt så att de får en volym. De skall alltså även ha en höjd. Inför klassen `Volume`. Klassen skall bl.a. ha metoden

```
public double getVolume()
```

Visa hur designmönstret Decorator kan användas för att lösa problemet, utan att ärva från rektangel- eller ellipsklassen.

a) Komplettera klassdiagrammet med klassen `Volume`, samt visa vilka instansvariabler och metoder klassen skall ha. Visa dessutom hur den är relaterad till andra klasser i diagrammet. Det räcker om de befintliga klasserna anges med namn i det nya diagrammet.
(3 p)

b) Implementera klassen `Volume` i Java.
(6 p)

c) Ge ett kodexempel som skapar en kub och skriver ut dess volym.
(1 p)