Skolan för datavetenskap och kommunikation

Objektorienterad Programkonstruktion, DD1346

Inklusive lösningsförslag

Tentamen 2013–03–12, kl. 09.00-12.00

Tillåtna hjälpmedel: Papper, penna och radergummi.

Notera: Frågorna i del I ska besvaras på för ändamålet lämnad plats i tentamenslydelsen. Frågorna i del II besvaras på separat papper. Använd gärna både fram- och baksida, men behandla högst en uppgift per sida. Kom ihåg att skriva namn och personnummer på alla inlämnade blad. Skriv tydligt!

Betygsgränser: Betyg XF: $\geq$ 17p i del I
Betyg E: $\geq$ 20p i del I
Betyg D: $\geq$ 20p i del I **och** $\geq$ 5p i del II
Betyg C: $\geq$ 20p i del I **och** $\geq$ 10p i del II
Betyg B: $\geq$ 20p i del I **och** $\geq$ 15p i del II
Betyg A: $\geq$ 20p i del I **och** $\geq$ 20p i del II

Ansvarig: Christian Smith (ccs@kth.se)

Lycka till!

Del I - flervalsfrågor

1. I denna uppgift finns 8 program(-delar), antingen i form av Java-kod eller i UML-diagram. *Generiska namn används i stället för de mer vanliga namn som avslöjar vilket mönster det är.* För varje program, ange det designmönster som bäst beskriver det. Välj från listan nedan. Varje korrekt angivet designmönster ger 1 p. (8 p)

Singleton	**MVC**	**Composite**	**Proxy**	**Adapter**	**Flyweight**
Lock	**Observer**	**Factory**	**Builder**	**Prototype**	**Facade**

   a) BUILDER

```
public class A{

    MyType1 myObject1;
    MyType2 myObject2;

    public A(){
        myObject1 = null;
        myObject2 = null;
    }

    public void setObject1(Object o){
        myObject1 = new MyType1();
        myObject1.doSomeInitialStuff(o);
    }

    public void modifyObject1(Object o){
        myObject1.doSomeShallowStuff(o);
    }

    public void setObject2(Object o){
        myObject2 = new MyType2();
        myObject2.doSomeInitialStuff(o);
    }

    public void modifyObject2(Object o){
        myObject2.doSomeDeeperStuff(o);
    }

    public myType3 getType3(){
        myType3 myOutputObject = new myType3(this);
        return myOutputObject;
    }
```

```
}
```

b) FACADE

```
public class B{

    MyType1 myObject1;
    MyType2 myObject2;
    MyType3 myObject3;
    MyType4 myObject4;

    public B(){
        myObject1 = new MyType1();
    }

    public int doStuff(Object o){
        myObject1.doSomeInitialStuff(o);
        myObject2 = myObject1.getType1();
        myObject2.setSomeParameters(0,1,2,3,4);
        myObject1.setSomeOtherParameters(myObject2.getSomeParameters());
        myObject3 = new MyType3(myObject1, myObject2);
        myObject4 = new MyType4();
        myObject4.setChildObjects(myObject1,myObject3);
        return myObject4.calculateHairyStuff();
    }

}
```

c) ADAPTER

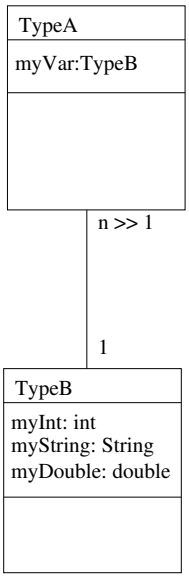
```
public class C{

    MyType1 myObject1;

    public C(){
        myObject1 = new MyType1();
    }
```

```
    public int doOtherStuff(float f){
        return myObject1.doMyStuff((double) f);
    }

}
```

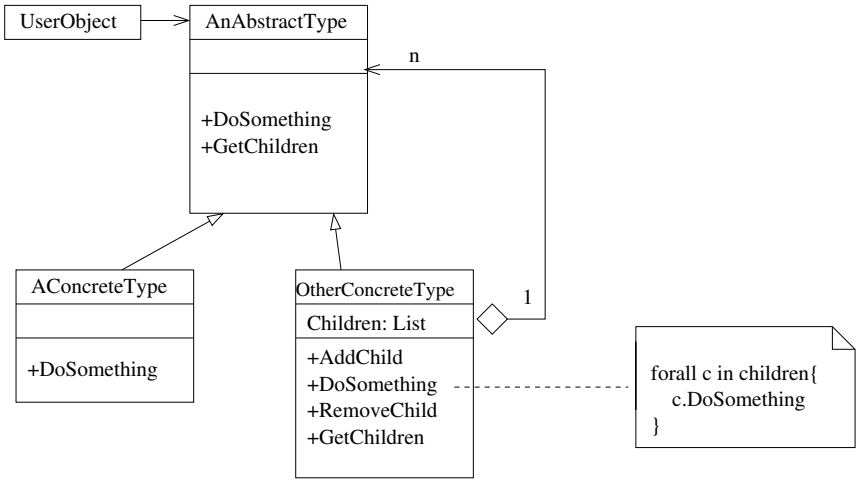
d) FACTORY

```
public class D{

    public static GenericType getObject(int a){
        if(decisionFunction(a)){
            return new SpecificType1();
        }else{
            return new SpecificType2();
        }
    }
}
```

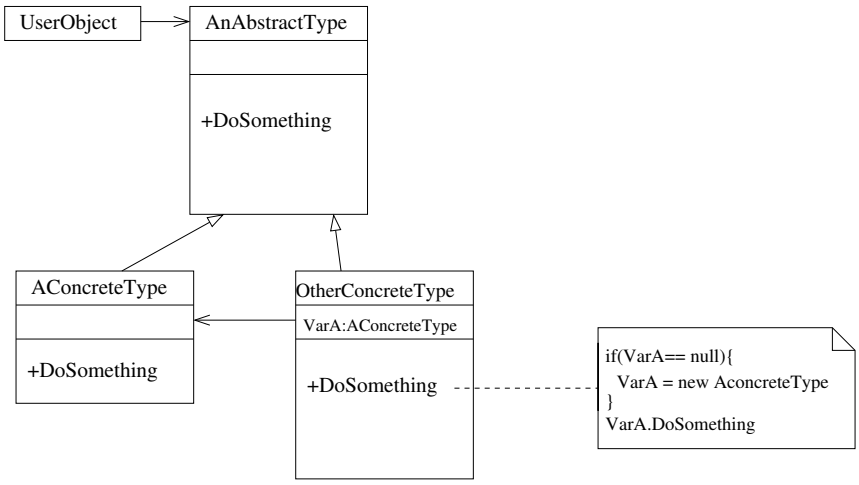
e) FLYWEIGHT



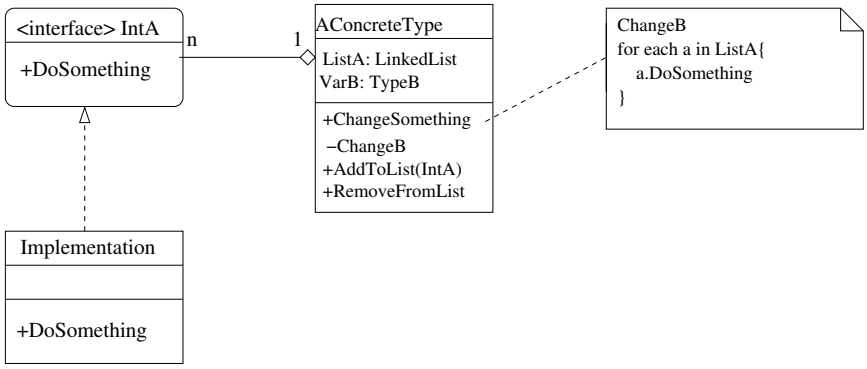
f) COMPOSITE



g) (VIRTUAL) PROXY



h) OBSERVER



2. Nedan följer 4 exempel på Java-program. För varje program, ange vad programmet skriver ut. Svara antingen med ett heltal mellan 0 och 20, eller ange om det är mindre än 0 eller större än 20. Varje korrekt analyserat program ger 1 p. (4 p).

a) -1

```
public class A2{

    static int a = 0;

    public static void main(String[] args){

        for(int i = 0; i>a--; i++){
            i=++i+a--;
        }

        System.out.println(a);

    }
}
```

b) >20 (21)

```
public class B2{

    static int a = 0;

    public static void main(String[] args){

        for(int i = 0; i<20; a++){
            i=a;
        }

        System.out.println(a);

    }
}
```

c) 15

```
public class C2{

    static int a = 0;

    public static void main(String[] args){
        int a = 5;
        doStuff(a);
    }

    private static void doStuff(int a){
        for(int i = 0; i<10; i++){
            a++;
        }
        System.out.println(a);
    }
}
```

d) 0

```
public class D2{

    static int a = 0;

    public static void main(String[] args){

        int a = 5;
        doStuff(a);
        System.out.println(myPrint());
    }

    private static void doStuff(int a){
        for(int i = 0; i<10; i++){
            a++;
        }
    }

    private static String myPrint(){
        return String.valueOf(a);
    }
}
```

3. För varje påstående om trådar, synkronisering och parallella program i Java, ange om det är sant eller falskt. 5 korrekta svar ger 4p, 4 korrekta svar ger 2p, 3 korrekta svar ger 1p, 2 eller färre korrekta svar ger 0 p (4 p)

   a) För att två olika trådar ska kunna anropa samma metod i samma objekt samtidigt, måste metoden vara deklararerad med nyckelordet `synchronized`.Falskt

   b) En `threadpool` används till att minska risken för `deadlock`.Falskt

   c) Ett `Lock` används till att minska risken för `thread interference`.Sant

   d) En `atomär operation` är säker mot `thread interference`.Sant

   e) Javakompilatorn kan hitta de flesta situationer där `livelock` uppstår, och ge felmeddelancen eller varningar.Falskt

4. För varje påstående om designmönster nedan, ange om det är sant eller falskt. 4 korrekta svar ger 2p, 3 korrekta svar ger 1p, 2 eller färre korrekta svar ger 0 p (2 p)

   a) I mönstret `Model View Controller` är det vanligt att man kan ha flera View-objekt men bara en Controller.Sant

   b) En `Iterator` behövs för att sortera arrayer.Falskt

   c) Med en `Singleton` samlar man ihop ett större antal klasser och metoder till ett gemensamt objekt med en (eller ett fåtal) enkla metodanrop.Falskt

   d) `Prototype`-mönstret kan användas internt i en `Factory`-metod för att skapa nya objekt.Sant

5. För varje påstående om gränssnitt (`interface`) i Java nedan, ange om det är sant eller falskt. 4 korrekta svar ger 2p, 3 korrekta svar ger 1p, 2 eller färre korrekta svar ger 0 p (2 p)

   a) Gränssnitt specificerar hur en viss metod ska implementeras.Falskt

   b) Utan gränssnitt skulle inte ett objekt av en klass kunna anropa en metod i ett objekt av en annan klass.Falskt

   c) Gränssnittet `Comparable` behövs för att använda Javas inbyggda sorteringsalgoritmer.Sant

   d) Om en klass **A** ärver en annan klass **B**, och **B** implementerar gränssnittet **C**, så implicerar det att **A** också implementerar **C**.Sant

6. Nedan följer 5 kodexempel i Java. För varje kodexempel, ange om koden går att kompilera. Eventuella fel kan förväntas finnas inom den listade koden, dvs du kan anta att alla `import`-påståenden är korrekta, och att alla anrop av externa metoder fungerar som man förväntar sig. Varje korrekt analyserad programkod ger 1p. (5 p)

   a) Kompilerar

```
public class A6 implements Comparable<A6>{

    private int a = 1;

    public int compareTo(A6 other){
        return this.a - other.a;
    }
}
```

   b) Kompilerar inte, a ej statisk

```
public class B6{

    private int a = 1;

    public static void main(String[] args){
        System.out.println(a);
    }

}
```

   c) Kompilerar inte, myC6 ej initialiserad

```
public class C6{

    public String myHej;

    public static void main(String[] args){
        C6 myC6;
        myC6.myHej = "Hej";
        System.out.println(myC6.myHej);
    }

}
```

d) Kompilerar

```
import java.util.*;

public class D6{

    public static Object[] getArray(Iterable i){

        ArrayList<Object> myList = new ArrayList<Object>();

        for(Object o : i){
            myList.add(o);
        }

        return myList.toArray();
    }
}
```

e) Kompilerar inte, variabel deklarerad som subbklass till Date kan ej tilldelas en variabel av typen Date

```
import java.util.*;

public class E6 extends Date{

    public static E6 myE6;

    public static void assignDate(Date d){
        myE6 = d;
    }

}
```

Del II - fördjupningsfrågor

Följande uppgifter besvaras på separat papper.

7. a) Förklara begreppet polymorfism, och på ge exempel på hur polymorfism kan underlätta för programmerare. (2 p)

   Polymorfism när man kan deklarera en variabel som en superklass (eller gränssnitt), och instansiera den med godtycklig klass som ärver från den deklarerade klassen (eller implementeras gränssnittet). Ett exempel på hur detta kan underlätta för programmerare är att man inte behöver ange explicit vilka klasser som kan användas som parametrar till ett metodanrop, utan bara behöver ange vilket gränssnitt som dessa klasser måste uppfylla. Detta gör koden mer generell, och man behöver inte veta vilka klasser som kommer att användas i framtiden när man skriver sin metod.

   b) Hur hanterar Java polymorfism? Vilka är tillåtna typer för att deklarera resp. instansiera variabler med? Vilka metodanrop är tillåtna? Hur/när avgörs vilken metod som ska anropas? (3 p)

   I Java är det tillåtet att deklarera en variabel som en godtycklig klass eller gränssnitt. Variabeln får sedan instansieras med variabler av icke-abstrakta klasser som antingen är eller ärver från den deklarerade typen, eller implementerar det deklarerade gränssnittet. Alla metoder som finns deklarerade i den deklarerade typen är tillåtna att anropa (givet rätt åtkomsträttigheter). Om det är en statisk metod så avgörs det vid kompilering att metoden skall anropas som den står i den deklarerade typen, för icke-statiska metoder avgörs vid programmets körning att metoden skall anropas som den står i den instansierade typen (dynamisk bindning).

8. Antag att du har fått i uppgift att skriva ett program för att simulera partikelflöden i rörsystem. I programmet ska man kunna rita upp sina rörsystem, bestående av sammansättningar av färdiga rördelar som användaren kan välja från ett bibliotek. Man ska kunna modifera de färdiga rördelarnas sammansättning och egenskaper, och kunna spara både rördelar, delsystem och kompletta rörsystem till fil för att återanvända senare.

   Du har hittat en uppsättning förenklade differensekvationer för beräkning av partiklarnas rörelser i diskret tid, som du tänker använda tills din kollega har härlett något bättre. I din förenklade modell kan varje partikel uppdateras obereoende av de andra i varje steg, men man måste ha med väldigt många partiklar för att få realistiska resultat. Efter att en simulering har körts ska man dels kunna spela upp en animering av flödet, och dels kunna visualisera olika statistiska mått, som t.ex lokala partikeltätheter.

   Beskriv en bra struktur för programmet. Förklara vilka designmönster som används, i grova drag hur de implementeras (vilken information finns var, hur kommunicerar programmets olika delar med varandra, hur styrs olika programflöden, osv). Motivera varför din lösning är bra! För full poäng skall hänsysn ha tagits till alla delar som beskrivs ovan. Du får använda UML eller (pseudo-)kod i ditt svar om du tycker att det förenklar presentationen, men det är inte ett krav. (10 p)

   Denna uppgift har förstås flera möjliga lösningar. Här ges ett exempel.

   Som grundläggande struktur kan vi med fördel använda MVC. Vi har en model som innehåller de olika rörsystemen med komponenter, partiklarna och deras differensekvationer. Rörsystemen och dess komponenter kan gärna ligga i en Composite-struktur för att underlätta editering av delsystem. modellen lagrar också data från en simulering. De hierarkiska strukturerna i Composite lämpar sig dessutom väl för att lagra rörsystemen i XML-format då man sparar/läser filer från disk.

   View-delen innehåller flera olika GUI, dels för att bygga och modifiera rörsystemen, och dells för att spela upp simuleringar. viewdelen observerar model, så att editor-GUI:et alltid visar den aktuella rörstrukturen (eller en del av den), och uppspelaren visar tillståndet i den tidpunkt som just nu spelas upp. Controller observerar view för att få tillgång till användarens inmatning.

   När simuleringen körs så kan controller-delen skapa en threadpool, som sprider ut partiklarnas positionsuppdateringar mellan lagom många trådar för att belasta processorn fullt ut. Om man bara använder en tråd har man outnyttjad kapacitet, och använder man för många blir det onödig overhead för att hålla reda på trådarna. Utformningen av de runnables som utför uppdateringen är helt definierad i model

   Under själva simuleringen har man en till observer, som observerar partiklarnas tillstånd i modellens och skriver detta till en fil. Hur partiklarnas tillstånd representeras i modellen och i filen på disk definieras i model, och controller behöver inte ha explicit vetskap om detta. Vid uppspelning låter controllern modellen ladda in en ny uppsättning partikeltillständ från filen i varje tidssteg.

När kollegan har utvecklat en bättre fysikalisk modell, så ändrar man i model, och controller och view är oförändrade. För att detta ska fungera måste alla metodanrop i model vara definierade tillräckligt abstrakt, tex “loadNextTimestep()”, eller “getRunnable()”, så att controller inte behöver ändras när simuleringsmodellen byts ut.

9. Förklara hur man kan använda `Observer` i `MVC` (2 p) Man kan låta view observera modellen, så att alla ändringar i modellen genast uppdateras utan att implicit blanda in controller. Detta gör det lättare att variera vilka view-objekt som skall finnas när programmet körs. Vidare kan man låta controllern observera view, om view innehåller de interaktiva GUI-kompionenterna, så att controller kan reagera på användarens inmatning.

10. Förklara skillnaden mellan ett vanligt `Observer`-mönster och en `publisher/subscriber`. Vilka fördelar har det senare? (3 p) I ett vanligt observer-mönster så håller det observerade objektet reda på vilka objekt som observerar det och alltså skall notifieras när det händer något. I publisher/subscriber så finns det en kommunikationskanal, en topic, som man kan ansluta sig till både som observerat och observerande objekt. I det senare fallet behöver det observerade objektet inte implicit hålla reda på vilka andra objekt som skall uppdateras, och man har åstadkommit ännu lösare koppling.

11. `XML` kan ibland vara ett bra format för att lagra data, men inte alltid. Förklara när det är bra, och när det passar dåligt, och ge exempel för båda fallen (2 p) XML passar bra när man har hierarkiska data med mycket meta-information. Det har dessutom fördelen att det är hyggligt lättläst för en människa. Det passar dock dåligt när man inte har en tydlig hierarki, men stora mängder data som ska hanteras effektivt. Ett exempel där XML passar bra är ett textdokument med olika kapitel och sektions-nivåer. Ett exempel där det passar dåligt är för att lagra stora högupplösta foton.

12. En vän ringer dig och ber dig hjälpa till med att fixa ett problem i ett program. Hen har försökt skriva om ett Matlab-program till Java, men något går snett, och beräkningarna ger helt andra resultat i Java-versionen. Du föreslår att hen ska skriva ut alla delresultaten av en körning i både Matlab- och Java-versionen, jämföra dessa och på så sett hitta var det börjar divergera. Efter en halvtimme ringer hen igen och säger att hen gjort det men inte kan se att resultaten skiljer sig åt någonstans, inte ens slutresultatet. Det var kanske inget fel i alla fall. 10 minuter senare ringer telefonen igen. Nu har hen tagit bort alla störande utskrifter, men nu fungerar det inte längre. Vad är det (troligen) som händer? Vad kallas fenomenet och hur kan det uppstå? (3 p)

    Det troliga är att felutskrifterna i sig påverkar programmets körning. Ett vanligt fall då detta inträffar är i flertrådade program, då utskrifterna kan påverka en eller flera trådars tidsåtgång för olika operationer, så att de inte längre försöker komma åt samma data samtidigt, dvs man får olika mycket thread interference beroende på om man har med utskrifterna. Detta fenom kallas för en "heisenbug".