Skolan för datavetenskap och kommunikation

# Objektorienterad Programkonstruktion, DD1346

Tentamen 2012–06–11, kl. 10.00-13.00

**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: $\geq$ 18p i del I
Betyg E: $\geq$ 20p i del I
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!***

---

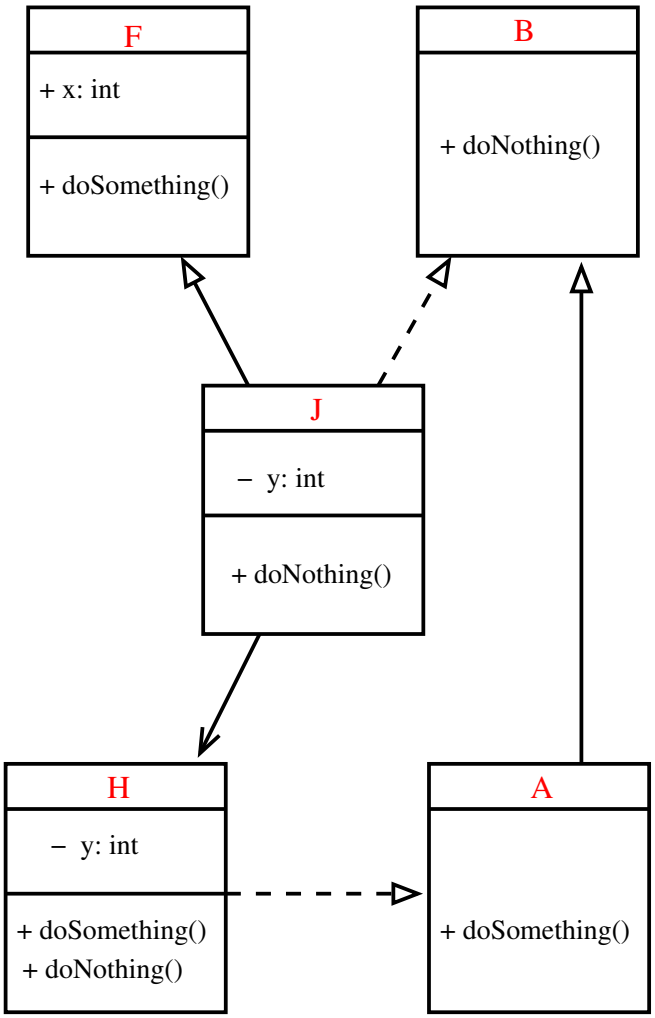
# Del I - flervalsfrågor

1. Nedan följer namnen på 8 olika designmönster. För varje mönster 1–8, ange med bokstav a–h vilket som är (det främsta) syftet med mönstret. Ett poäng ges för varje rätt svar. (8 p)

   1) Builder d
   2) Observer h
   3) Flyweight b
   4) Adapter a
   5) Prototype e
   6) Factory c
   7) Lock g
   8) Facade f

   a) Att möjliggöra för ursprungligen inkompatibla klasser att interagera
   b) Att spara minne vid förekomsten av ett stort antal liknande objekt
   c) Att kunna välja vilken klass som ska instansieras under pågående körning
   d) Att kunna tillgodose olika behov vid ett objekts skapande och dess användande
   e) Att minska resursanvändningen vid skapandet av nya objekt
   f) Att minska komplexiteten hos serier av invecklade anrop av flera klasser
   g) Att begränsa åtkomsten av en resurs till en enda tråd
   h) Att definiera ett ett-till-många förhållande mellan objekt, så att ett objekt kan uppdatera flera andra.

2. Fälten för namnen på delarna i nedanstående klassdiagram är tomma. Fyll i fälten med rätt namn (A–J) från klass- och gränssnittsdefinitionerna som följer från nästa sida. Vissa definitioner kan beskriva klasser/gränssnitt som inte fungerar eller inte finns med i UML-diagrammet. Det finns totalt 10 definitioner, från vilka 5 namn skall väljas ut. Varje rätt ifyllt namnfält ger 1 p. (5 p)



```
public interface A extends B{

    public void doSomething();

}


public interface B {

    public void doNothing();

}

public class C implements B inherits F {

    public void doNothing(){
        public G = new G();
    }

}

public class D implements B{

    public F myF;
    private G myG;
    private int y;

    public void doNothing(){
    }
}

public class E implements B{

    private class MyF extends F{
        private int y;
    };

    public void doNothing(){
        public myG = new G();
    }
}
```

```
public class F {

    public int x;

    public void doSomething(){
    }

}

public class G extends C{

    private int y;

    public void doSomething(){
    }

    public void doNothing(){
    }
}

public class H implements A {

    private int y;

    public void doNothing(){
    }

    public void doSomething(){
    }

}
```

```
public class I extends B implements F{

    int y;

    public void doSomething(){
        C myG = new G();
    }

    public void doNothing(){
    }

}

public class J extends F implements B{

    private int y;

    public void doNothing(){
        H myH = new H();
    }

}
```

3. Antag att vi deklararerar följande fält i klassen **K**, som finns i paketet **KAB**.

```
public     int w;
protected  int x;
private    int y;
static     int z;
```

Antag att vi sedan deklarerar följande klasser:

Klassen **A** ingår i paketet **KAB**, och ärver från **K**.
Klassen **B** ingår i paketet **KAB**, men ärver inte från **K**.
Klassen **C** ingår inte i paketet **KAB**, men importerar det och ärver från **K**
Klassen **D** ingår inte i paketet **KAB**, men importerar det men ärver inte från **K**
Klassen **E** ingår inte i paketet **KAB**, och importerar det inte och ärver inte från **K**

För varje klass, ange vilka fält ur **K** som den kan komma åt. (0.5 p för varje klass som ges en korrekt lista med fält): (3 p)

**A**: wxz
**B**: wxz
**C**: wx
**D**: w
**E**: inget, eller bara w. Detta beror mycket på övrig struktur i programmet, så båda svaren accepteras
**K**: wxyz

4. För varje påstående om Exceptions i Java, 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) Kod som anropar en metod som kan kasta en eller flera Exception måste alltid inneslutas i en `try`/`catch` -struktur. Falskt

   b) En Exception är ett objekt som kan innehålla information om varifrån det kastades och varför. Sant

   c) Man är alltid helt garanterad att kod i ett `finally`-block kommer att exekveras. Falskt

   d) Alla typer av Exceptions som man kan kasta finns redan definierade i Javas standard-API. Falskt

5. För varje påstående om variabler och objekt 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) Man kan definiera egna primitiva datatyper i Java. Falskt

   b) En variabel kan bara peka på objekt av den klass som den deklarerats som. Falskt

   c) För varje primitiv datatyp som finns definierad i Java, finns det en motsvarande wrapperklass ur vilken man kan skapa en "objektversion" av typen. Sant

   d) Arrayer kan enbart innehålla variabler av primitiva typer. Falskt

   e) En egendefinierad klass kan ärva från en primitiv datatyp. Falskt

6. För vart och ett av följande fyra program, **A**, **B**, **C** och **D**, ange om programmet alltid ger samma utskrift, ger olika utskrift från gång till gång, eller om det låser sig och alltså aldrig ger någon utskrift alls. Varje korrekt svar ger 1p. Skriv svaret direkt under varje kodsnutt! (4 p)

```
public class A extends Thread{

    static int a = 0;

    public static void main(String[] Args){

        A MyA = new A();
        MyA.start();
        runLoop();
        System.out.println(a);
    }

    public static void runLoop(){
        for(int i = 0; i<1000000; i++){
            a=i;
        }
    }

    public void run(){
        runLoop();
    }
}
```

A ger olika resultat från gång till gång

```
public class B extends Thread{

    static int a = 0;

    public static void main(String[] Args){
        B MyB = new B();
        MyB.start();
        runLoop(1000000);
        System.out.println(a);
    }

    public static synchronized void runLoop(int b){
        for(int i = 0; i<b; i++){
            a=i;
        }
    }

    public void run(){
        runLoop(12345678);
    }
}
```

B ger olika resultat från gång till gång

```
public class C extends Thread{

    static int a = 0;

    public static void main(String[] Args) throws Exception{
        C MyC = new C();
        MyC.start();
        MyC.join();
        runLoop(1000000);
        System.out.println(a);
    }

    public static void runLoop(int b){
        for(int i = 0; i<b; i++){ a=i;}
    }

    public void run(){
        runLoop(12345678);
    }
}
```

C ger samma resultat varje gång

```
public class D extends Thread{

    static int a = 0;

    public static void main(String[] Args) throws Exception{
        D MyD = new D();
        MyD.start();
        runLoop(1000000);
        MyD.join();
        System.out.println(a);
    }

    public static void runLoop(int b){
        for(int i = 0; i<b; i++){
            a=i;
        }
    }

    public void run(){
        runLoop(12345678);
    }
}
```

D ger olika resultat från gång till gång

# Del II - fördjupningsfrågor

Följande uppgifter besvaras på separat papper.

7. Förklara begreppen *deadlock* och *livelock*. (2 p)

   Deadlock är når två (eller fler) trådar har blockerat varandra, genom att båda två håller ett lås som den andre väntar på.
   Livelock är når två (eller fler) trådar blockerar varandra trots att de löpande släpper sina gamla lås och tar nya, jmfr två personer som ska passera varandra i en korridor, och byter sida fram och tillbaka i samma tempo i all evighet och alltså inte kommer förbi varandra.

8. Beskriv hur man kan använda klasserna `Socket` respektive `ServerSocket` för att upprätta en TCP/IP-förbindelse mellan två Java-program. (3 p)

   I det ena programmet (A) skapar man en ServerSocket, SSA som får lyssna på en given port med commandot accept(). I det andra programmet (B) skapar man en Socket SB, som ansluter till den givna porten på den IP-address där A finns, genom att man angett IP och port i konstruktoranropet för SB. Då kommer SSA.accept() att returnera en Socket, SA, som är ansluten till SB. Både new()-anropet i B och accept()-anropet i A kan kasta flera olika Exceptions som måste hanteras.

9. Antag att vi har skrivit ett Javaprogram som tillhandahåller en bildmanipuleringsserver. Klienter kan ansluta till servern utifrån, och skicka in en bild, som servern bearbetar så att den ser gammal och gulnad ut, för att sedan skicka tillbaka den till klienten. Denna tjänst blir populär, och servern får in en stor mängd anslutningar utifrån, som mest upp emot 20 anslutningar i sekunden. Servern börjar gå långsamt, och användare är besvikna på de långa väntetiderna, ibland flera minuter. Själva bildbehandlingen görs i ett snabbt externt program som du saknar källkoden till, men du vet att det inte är multitrådat, och att det kan konvertera en bild på 50 ms — inklusive programmets uppstartstid — och alltså borde orka med anstormningen. Dessutom vet du att serverdatorn har en flerkärnig processor och att din nätverksanslutning inte heller har några problem att skicka mer än 20 bilder per sekund i båda riktningarna.

   Ditt Javaprogram startar en ny tråd för varje ny anlsutning, och varje tråd tar emot en bild, kör den genom det externa bildprocessprogrammet, och skickar tillbaka den innan den terminerar. Du har upptäckt att ditt program ofta kör uppemot tusen trådar samtidigt.

   Ge en rimlig konkret förklaring till varför programmet går långsamt, och föreslå hur man ska ändra koden för att lösa problemet enkelt och effektivt. För full poäng krävs rätt namn på ingående designmönster. (5 p)

   En trolig förklaring är att det tagit längre tid än förväntat att ladda upp bilderna, t.ex för att klientens anslutning är långsam, eller nätverket är hårt belastat. Då kommer det att skapas nya trådar för nya anslutningar i högre takt än vad de gamla trådarna blir färdiga och returnerar. När antalet trådar väl blivit tillräckligt högt kommer en stor del av tillgängliga resurser gå åt till att hantera själva trådarna, deras skapande och context switching, och bara en liten del av resurserna till att faktiskt hantera bildkonverteringsanrop. Ett sätt att lösa detta är att använda en Thread Pool, och därmed begränsa antalet trådar, och mängden context switching. Man kapar dessutom resursförbrukningen för att skapa nya trådar. Det optimala antalet trådar i threadpoolen får man troligtvis hitta empiriskt, men man kan börja med ett lågt antal, t.ex 10 st.

10. Antag att du ska skriva ett Javaprogram för att skapa och redigera UML-diagram. Vid redigering ska man kunna välja antingen bara en enskild komponent (t.ex en textrad eller en pil), eller en sammansatt komponent (t.ex en klassruta), eller större delträd. Man ska sedan kunna ändra färg på, flytta, kopiera, eller göra andra operationer på den/de valda komponenterna.

    a) Vilket/vilka designmönster passar för programmets interna datastrukturer? Hur fungerar dessa mönster i kontexten av detta program? (3 p)

    Composite passar bra här. Man definierar en abstrakt komponentklass med de operationer man vill ha, t.ex flytt och kopiering, men även grafiskt representation, och sedan en composite-klass som kan representera delträd och klassrutor, medan löv-klassen används till de minsta, odelbara, enheterna, vilket i detta fall bör vara pilar, textrader, och själva rutan runt en klass. Om många grafiska komponenter delar utseende kan de dessutom lagra detta externt, enligt mönstret Flyweight.

    b) Vilket/vilka designmönster är lämpliga för programmets övergripande struktur och kommunikatio nen mellan de olika komponenterna? Hur fungerar dessa mönster i kontexten av detta program? (4 p)

    Som i många fall med GUI-programm är det lämpligt att använda MVC. Modellen är då själva composite-trädet, medan den grafiska representationen finns i View, och Controller-delen styr interaktionen, t.ex genom att skicka vidare användarens kommandon till rätt del av composite-trädet, och genom att styra när trädet behöver byggas om för att användaren definierar nya delträd. Lyssnare används för att uppdtare relevanta delar av programmet när någonting ändras. Till exempel kan den grafiska representationen lyssna på inställningar där användaren byter färg.

11. a) Ge exempel på när mönstret *flyweight* är lämpligt att använda. (1 p)

När man har ett stort antal objekt av samma klass, där de delar på en begränsad mängd data, t.ex grafiska objekt med identiskt utseende, som. t.ex föremål i ett spel.

b) Ge exempel på när mönstret *prototype* är lämpligt att använda. (1 p)

När det är resurskrävande att skapa nya objekt, men det är samma procedur som krävs för skapandet av varje nytt objekt, eventuellt med små variationer under programmets körning.

c) Bör man använda djupa eller grunda kopior när man klonar objekt i de ovanstående mönstrena? Förklara varför! (3 p)

I Flyweight är det oftast lämpligt med grunda kopior, eftersom data ändå lagras externt, och varje objekt bara innehåller pekare till denna gemensamma lagringsplats.
I Prototype är det oftast lämpligt med djupa kopior, eftersom man inte vill att de redan skapade objekten ska påverkas då vi ändrar prototypen eller andra objekt ur samma mall.

12. Nätverkspaket som skickas med protokollet TCP är garanterade att komma fram, och dessutom i rätt ordning. Förklara varför detta gör det olämpligt för realtidsapplikationer. (3 p) Eftersom alla paket är garanterade att komma fram i rätt ordning måste alla senare paket vänta in tidigare paket som kommit bort eller försenats. Om man skickar 100 paket i sekunden, och ett enskilt paket blir 5 s försenat, så har man alltså fördröjt 500 paket med upp till 5 s. I detta fall hade det varit bättre att strunta i det försvunna paketet och hantera de senare paketen utan fördröjning i stället.