Skolan för datavetenskap och kommunikation

Objektorienterad Programkonstruktion, DD1346

FACIT till Tentamen 2017–03–14, kl. 14.00–17.00

Tillåtna hjälpmedel: Papper, penna och radergummi.

Notera: Frågorna besvaras på separat papper. Svaren till del I kan skrivas på samma sida om de får plats, men behandla högst en uppgift per sida för del II. Kom ihåg att skriva namn och personnummer på alla inlämnade blad. Skriv tydligt!

Betygsgränser: Betyg FX: $\geq$ 17p i del I
Betyg E: $\geq$ 20p i del I
Betyg D: $\geq$ 20p i del I **och** betyg D på del II
Betyg C: $\geq$ 20p i del I **och** betyg C på del II
Betyg B: $\geq$ 20p i del I **och** betyg B på del II
Betyg A: $\geq$ 20p i del I **och** betyg A på del II

Eventuella bonuspoäng adderas till resultatet i del I.
För betyg på del II, se instruktionerna till del II.

Ansvarig: Christian Smith (ccs@kth.se)

Lycka till!

Del I - flervalsfrågor

1. I denna uppgift följer beskrivningar av 6 olika designmönster. För varje beskrivning, ange namnet på mönstret som bäst matchar beskrivningen (välj ur listan nedan). Varje korrekt angett mönster ger 1 p. (6 p)

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

   a) Man skapar först en ursprunglig instans, som sedan kopieras varje gång man behöver nya objekt av typen. Prototype

   b) Man använder ett gemensamt gränssnitt för seriell åtkomst av innehållet ur olika typer av samlingar (**Collections**). Iterator

   c) Man samlar objekt i en trädstruktur, som möjliggör att man behandlar noder och grenar på samma sätt. Composite

   d) Man gör så att bara en tråd i taget får åtkomst till en viss metod. Lock

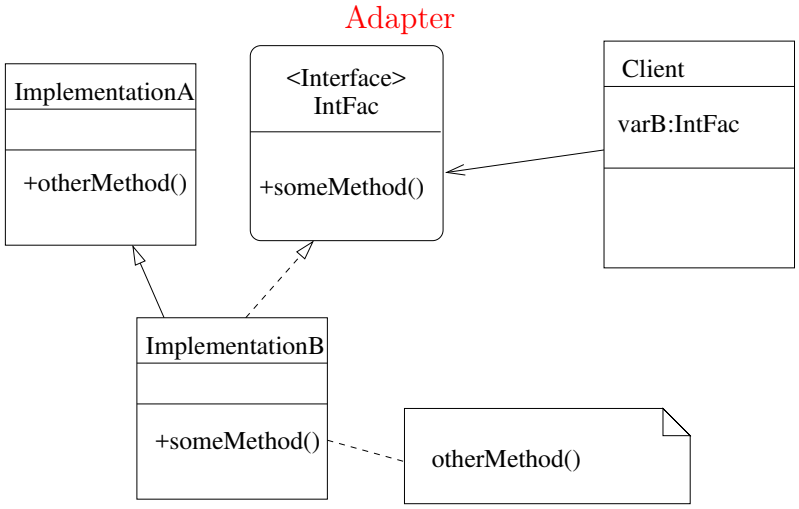
   e) Man delar upp sitt program i tydligt avgränsade delar, som ansvarar för visualisering, datalagring, respektive övergripande programstyrning. MVC

   f) Man samlar en komplex struktur av flera olika klasser och metodanrop bakom en klass som har ett enkelt externt API. Facade

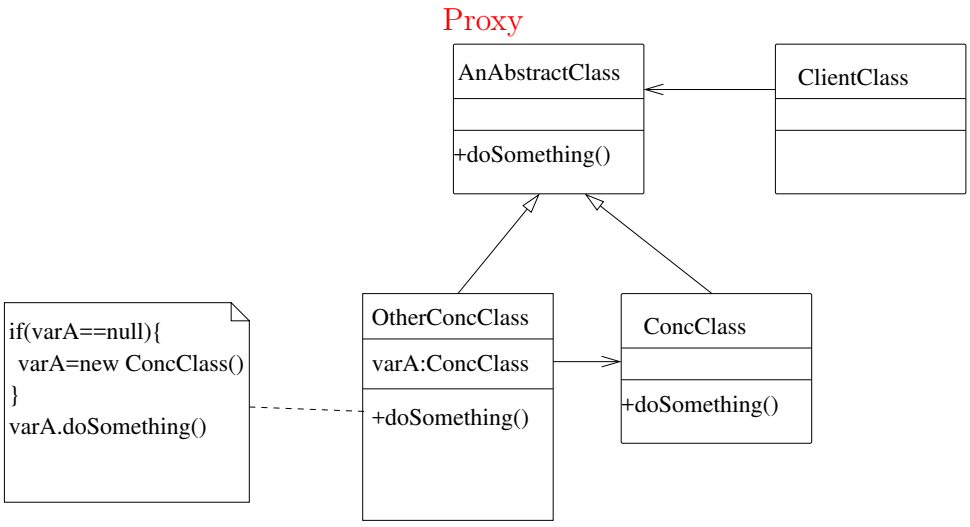
2. I denna uppgift finns 4 UML-diagram. För varje diagram, ange vilket mönster (från listan nedan) som beskrivs. *Observera att generiska namn används i stället för standardiserade namn.*
   Varje korrekt namngivet diagram ger 1 p. (4 p)

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

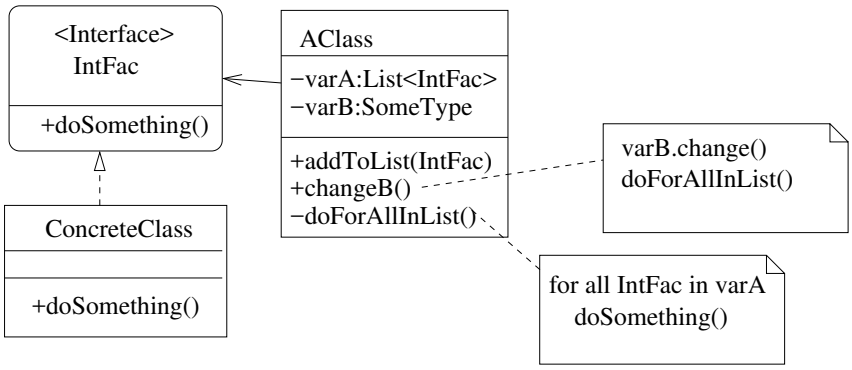
a)



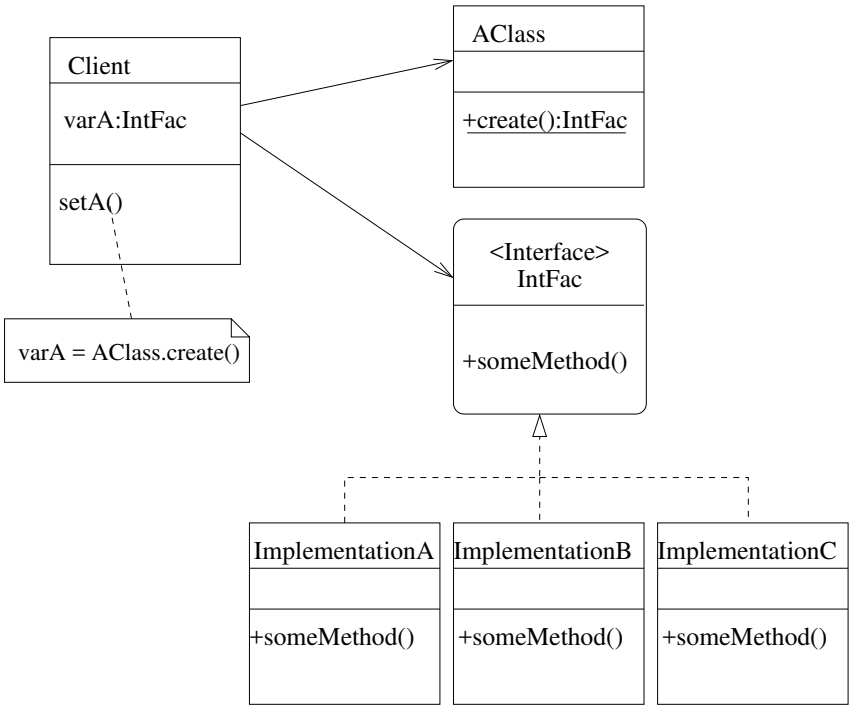
b)



c) Observer



d) Factory



3. Här följer 5 st kodlistningar i Java[1]. För varje kodlistning, ange om den kommer att fungera, ge kompileringsfel eller ge fel när man kör den. Kod anses fungera **omm** den ger en deterministisk utskrift av ett tal vid körning. Anta att varje klass **X** finns i en fil som heter '`X.java`', och kompileras med kommandot '`javac X.java`', och körs med '`java X`'. (5 p)

   a) Fel vid körning (ingen main()-metod)

```
public class A{

    private A(){
        this(3);
    }

    private A(int a){
        System.out.println(a);
    }

}
```

   b) Fungerar

```
public class B{

    static private B myB;

    public static void main(String[] args){
        myB = new B(5);
    }

    private B(int b){
        System.out.println(b);
    }

}
```

[1]För tydlighets skull anses här Java 8, som finns i skolans datasalar

c) Fel vid körning (icke-deterministisk)

```
public class C{

    public static void main(String[] args){

        C myC = new SubC();
        myC.printC(5);
    }

    public static void printC(int c){
        System.out.println(c * Math.random());
    }


    static class SubC extends C{

        public static void printC(int c){
            System.out.println(c);
        }
    }

}
```

d) Fel vid körning (icke-deterministisk)

```
public class D implements Runnable{

    static int d = 0;
    static final Object lock = new Object();

    public static void main(String[] args)throws InterruptedException{
        Thread th1 = new Thread(new D());
        Thread th2 = new Thread(new D());

        th1.start();
        th2.start();

        th1.join();
        th2.join();

        System.out.println(d);

    }

    public void run(){
        while(d < 50000){
            incD();
        }
    }

    private void incD(){
        synchronized(lock){
            d++;
        }
    }

}
```

e) Fungerar

```
public class E extends Thread{

    private static int e = 0;
    private final Object lock = new Object();

    public static void main(String[] args) throws InterruptedException {
        E E1 = new E();
        E E2 = new E();

        E1.start();
        E1.join();
        E2.start();
        E2.join();

        System.out.println(e);
    }

    public void run(){
        doStuff();
    }

    private void doStuff(){
        while(++e < 50000){
            synchronized(lock){
                e += e++;
            }
        }
    }
}
```

4. För varje påstående om nyckelord 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) Ett fält som deklarerats som **synchronized** i ett objekt kommer automatiskt att uppdateras om motsvarande fält ändras i ett annat objekt av samma klass. Falsk

   b) En metod som deklarerats som **final** kan inte överskuggas i en ärvande klass. Sant

   c) En klass som har deklarerats som **abstract** kan inte instansieras. Sant

   d) Värdet på ett fält som har deklarerats som **protected** får inte ändras efter initialisering. Falskt

5. För varje påstående, 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 0p. (4 p)

   a) Fält som deklarerats som **static** kallas för klassvariabler. Sant

   b) Alla klasser går att instansiera, eftersom de ärver från **Object**, som går att instansiera. Falskt

   c) Klassmetoder kan inte direkt skriva till instansvariabler. Sant

   d) Gränssnitt kan bara innehålla klassmetoder. Falskt

   e) Instansmetoder kan inte direkt anropa klassmetoder. Falskt

6. För varje påstående om arv 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 0p. (4 p)

   a) Om ett gränssnitt ärver från flera andra gränssnitt så måste en implementerande klass implementera samtliga metoder från alla gränssnitt i arvshierarkin. Sant

   b) Man får inte deklarera en klass som både **abstract** och **final**, för den måste antingen kunna instansieras eller ärvas. Sant

   c) Om en variabel deklarerats med en superklass **A** kan man instansiera den med ett objekt av en klass **B** som ärver från **A**. Sant

   d) Om en klass **B** ärver från en superklass **A** får **B** bara vara **abstract** om **A** också är det. Falskt

   e) Det är tillåtet för en klass att ärva från flera andra superklasser, under förutsättning att dessa superklasser bara innehåller abstrakta metoder. Falskt

Del II - fördjupningsfrågor

Följande 5 frågor kan betygsättas med antingen **A** eller **C**. Betyget **A** ges då allt som efterfrågas i uppgiften besvaras korrekt, med en tillräcklig motivering. Mindre detaljfel som inte påverkar helheten kan accepteras. Betyget **C** ges då någon del av uppgiften inte besvarats korrekt eller getts tillräcklig motivering, eller om texten utöver efterfrågat svar innehåller betydande mängd information som inte efterfrågats (dvs. man får inte "dubbelgardera"). Om betydande del av efterfrågat svar saknas eller är otillräckligt motiverat ges inget betyg.

Följande resultat krävs för att uppnå respektive betyg på denna del:

Betyg D: Uppnått lägst betyg **C** på minst 2 uppgifter.
Betyg C: Uppnått lägst betyg **C** på minst 4 uppgifter.
Betyg B: Uppnått betyg **A** på minst 2 uppgifter och **C** på alla resterande.
Betyg A: Uppnått betyg **A** på minst 4 uppgifter och **C** på eventuell resterande.

Besvara varje uppgift på ett separat papper.

7. Vad är polymorfism? Hur kan det användas för att förbättra prestanda vid programkörning?

8. Välj **ett** designmönster där man kan använda sig av antingen gränssnitt eller abstrakta klasser i implementationen. Beskriv hur respektive implementation kan göras, och förklara för- och nackdelar med respektive sätt.

9. En av dina nära vänner, som är mottagningsansvarig på en av KTH:s sektioner, har bett dig göra en app som de kan använda för att hålla koll på vad olika personer gör på campus under olika mottagningsaktiviteter. De vill att varje användare ska kunna använda programmet till att rapportera in var de befinner sig just nu, och att olika ansvariga ska kunna se denna information i en lista, så att de dels lätt kan hitta personer, se att alla positioner är korrekt bemannade, eller bara förbereda sig när någon grupp kommer närmare. Du ställer upp, men innan du hinner börja utveckla programmet kommer det en förfrågan från en annan sektion om de också kan få använda din app, men de vill dessutom kunna visualisera all information på en grafisk karta. Dessutom vore det bra om man kunde ta bilder och skicka dessa så att de ansvariga kan se hur det ser ut på en viss plats just nu. Du svarar att du inte har tid att utveckla mer än den nödvändigaste funktionaliteten för den första vännens sektion, men att du gärna delar med dig av koden, och i ett svagt ögonblick skryter du om att din kod är väldigt lätt att bygga vidare på, vad man än vill göra.

   Beskriv hur du strukturerar programmet, hur olika delar kommunicerar med varandra, och hur du ser till att alla behov ovan tillgodoses. Namnge alla ingående designmönster korrekt.

10. En vän har hört av sig med ett problem som hen vill ha din hjälp med. Hen har läst på om parallellisering och försökt göra ett bra program enligt konstens alla regler. Hen har skrivit ett multitrådat program, där en stor sammansatt uppgift har delats upp i små oberoende deluppgifter, som hanteras av varsin **Runnable**. Dessa kör på en **Threadpool**, och lås används för att undvika alla former av deadlock. Programmet fungerar förvisso, men det går lika långsamt när hen har åtta trådar i poolen som vid en tråd, trots att processorn har åtta kärnor. Vad kan problemet vara, och vad bör hen försöka göra för att få ut bättre effekt av sin parallellisering?

11. Jämför de två designmönstren **flyweight** och **virtual proxy**. Vilka likheter och skillnader har de? Vad används de till, och hur? Hur kan man implementera dem?