Skolan för datavetenskap och kommunikation

Objektorienterad Programkonstruktion, DD1346

Tentamen 2015–03–14, 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 – behandla högst en uppgift per sida. 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** $\geq$ 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

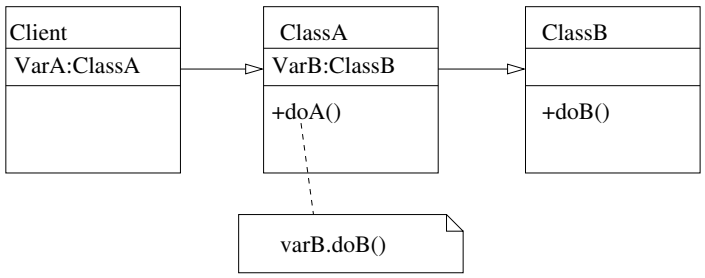
Ansvarig: Christian Smith (ccs@kth.se)

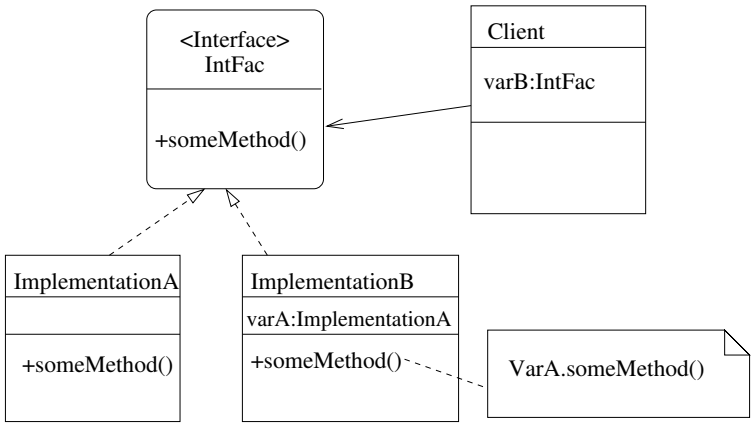
Lycka till!

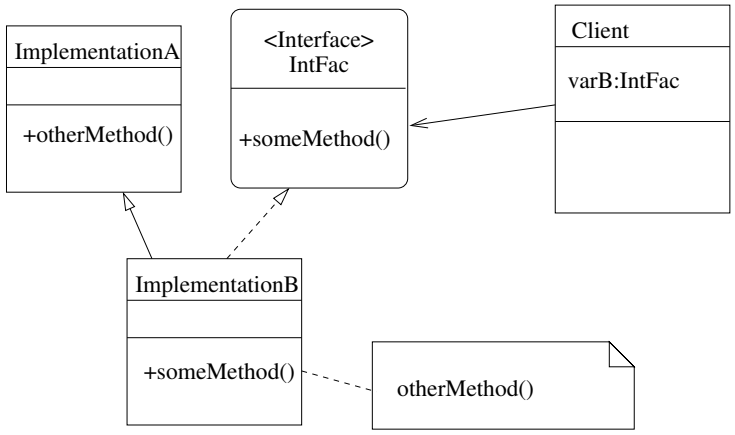
Del I - flervalsfrågor

1. I denna uppgift finns 4 namn på designmönster, åtföljda av 3 UML-diagram vardera. *Generiska namn används i stället för standardiserade namn.* För varje mönster, ange det UML-diagram som bäst beskriver en implementation av det. Varje korrekt angivet UML-diagram ger 1 p. (4 p)

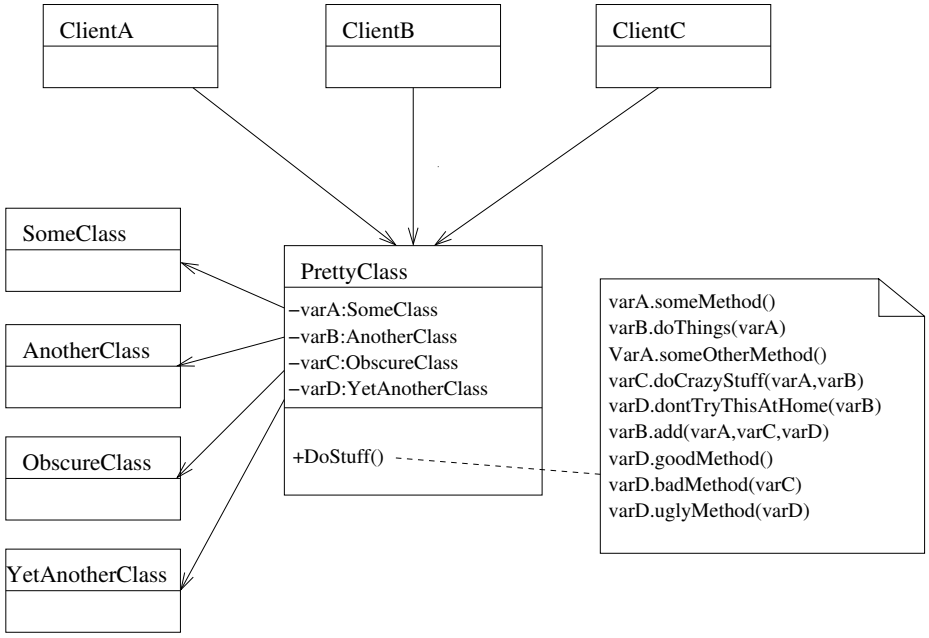
a) Adapter

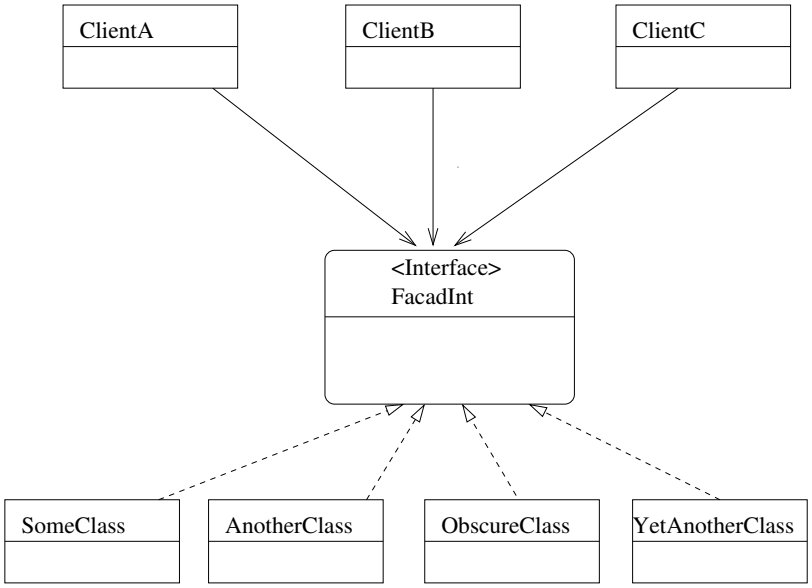


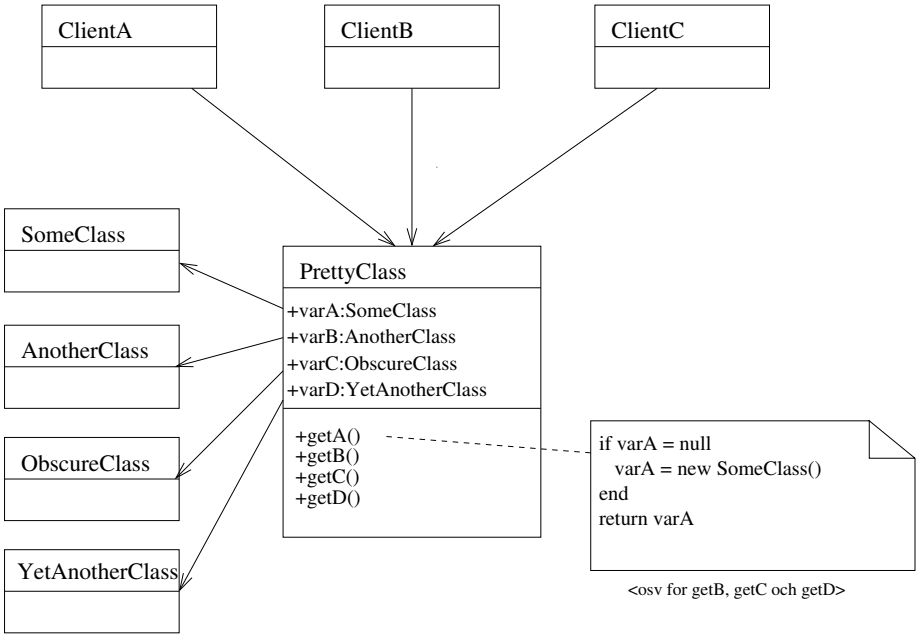




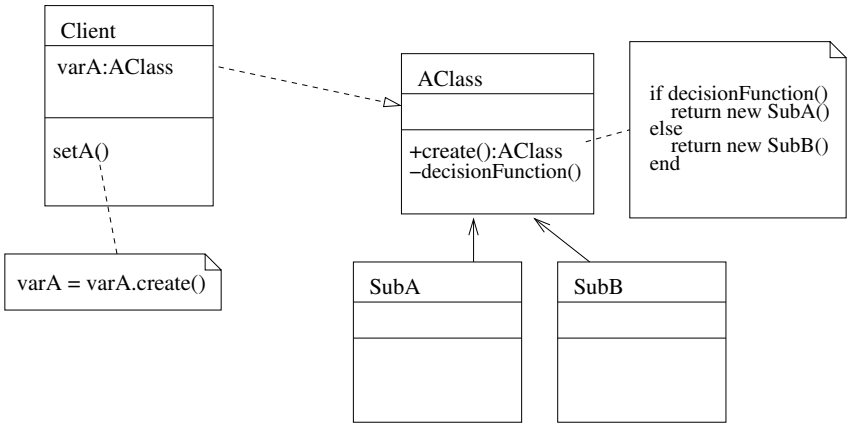
b) Facade

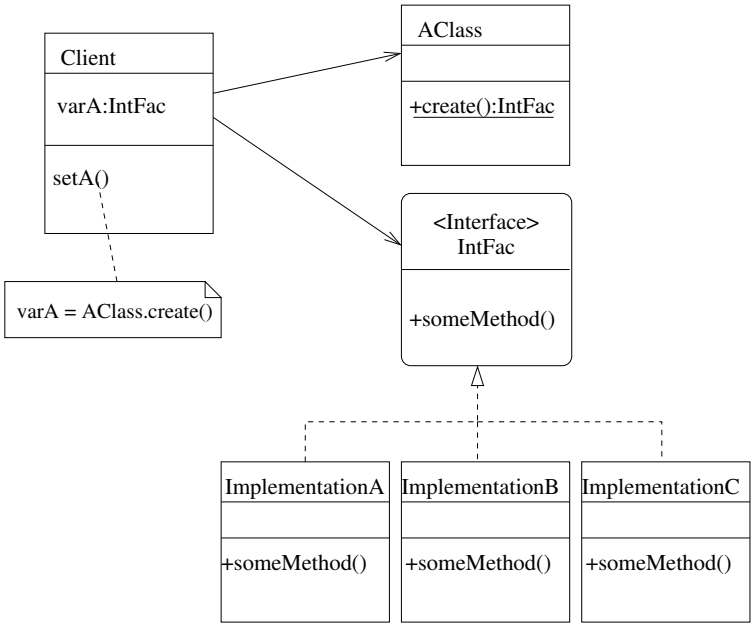

ClientA
ClientB
ClientC
SomeClass
AnotherClass
ObscureClass
YetAnotherClass
PrettyClass
−varA:SomeClass
−varB:AnotherClass
−varC:ObscureClass
−varD:YetAnotherClass
+DoStuff()
varA.someMethod()
varB.doThings(varA)
VarA.someOtherMethod()
varC.doCrazyStuff(varA,varB)
varD.dontTryThisAtHome(varB)
varB.add(varA,varC,varD)
varD.goodMethod()
varD.badMethod(varC)
varD.uglyMethod(varD)

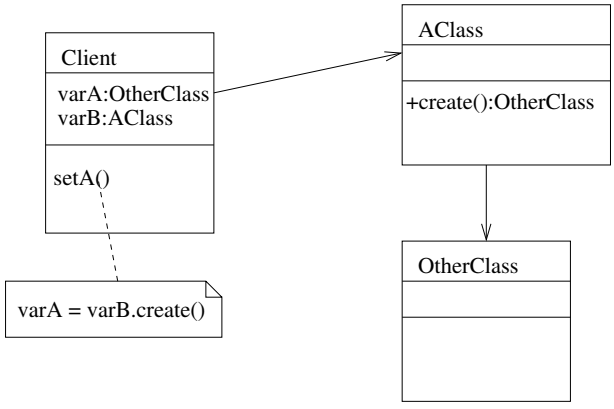

ClientA
ClientB
ClientC
<Interface>
FacadInt
SomeClass
AnotherClass
ObscureClass
YetAnotherClass


ClientA
ClientB
ClientC
SomeClass
AnotherClass
ObscureClass
YetAnotherClass
PrettyClass
+varA:SomeClass
+varB:AnotherClass
+varC:ObscureClass
+varD:YetAnotherClass
+getA()
+getB()
+getC()
+getD()
if varA = null
varA = new SomeClass()
end
return varA
<osv for getB, getC och getD>

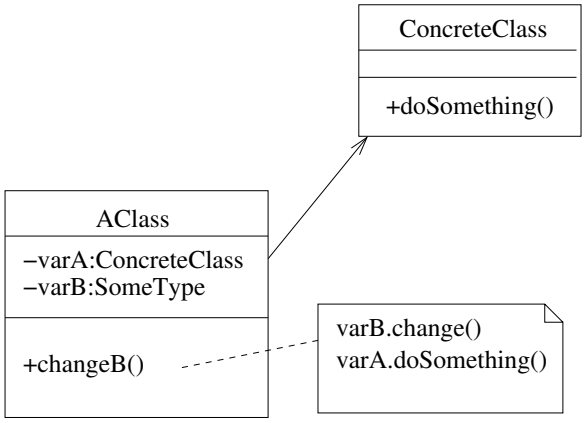
c) Factory

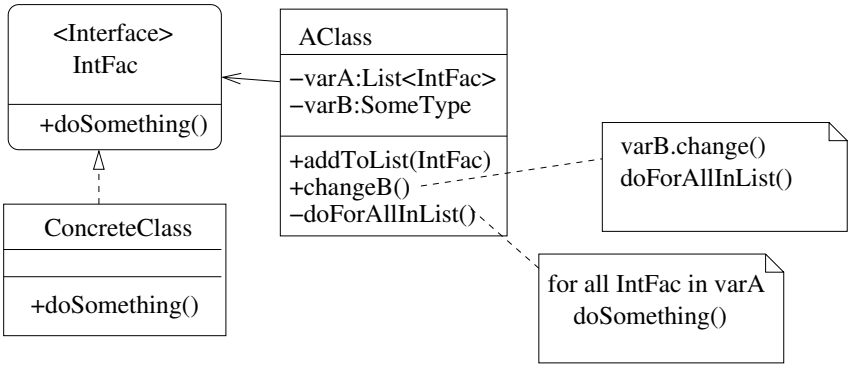

Client
varA:AClass
setA()
varA = varA.create()
AClass
+create():AClass
−decisionFunction()
if decisionFunction()
return new SubA()
else
return new SubB()
end
SubA
SubB

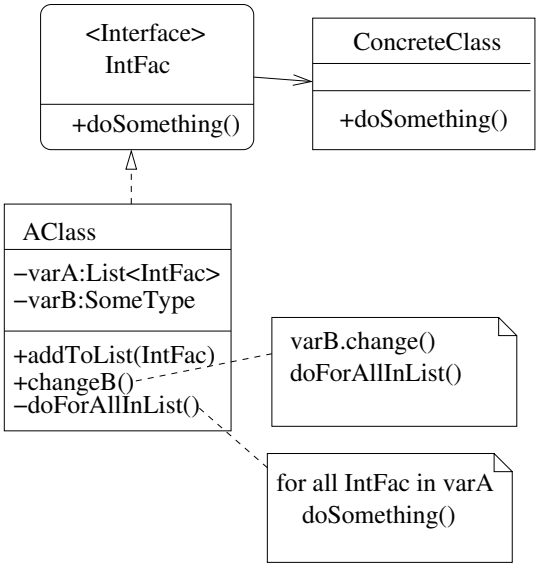

Client
varA:IntFac
setA()
varA = AClass.create()
AClass
+create():IntFac
<Interface>
IntFac
+someMethod()
ImplementationA
+someMethod()
ImplementationB
+someMethod()
ImplementationC
+someMethod()


Client
varA:OtherClass
varB:AClass
setA()
varA = varB.create()
AClass
+create():OtherClass
OtherClass

d) Observer


ConcreteClass
+doSomething()
AClass
−varA:ConcreteClass
−varB:SomeType
+changeB()
varB.change()
varA.doSomething()


<Interface>
IntFac
+doSomething()
ConcreteClass
+doSomething()
AClass
−varA:List<IntFac>
−varB:SomeType
+addToList(IntFac)
+changeB()
−doForAllInList()
varB.change()
doForAllInList()
for all IntFac in varA
doSomething()


<Interface>
IntFac
+doSomething()
ConcreteClass
+doSomething()
AClass
−varA:List<IntFac>
−varB:SomeType
+addToList(IntFac)
+changeB()
−doForAllInList()
varB.change()
doForAllInList()
for all IntFac in varA
doSomething()

2. I denna uppgift finns 4 namn på designmönster, åtföljda av 3 kodlistningar vardera. *Generiska namn används i stället för standardiserade namn.* För varje mönster, ange den kodlistning som bäst implementerar det. Varje korrekt angiven kodlistning ger 1 p. (4 p)

a) Prototype

```
public class Prototype1 implements Cloneable{

    private static Prototype1 myPrototype = null;
    private SomeTypeA myA;
    private SomeTypeB myB;
    private SomeTypeC myC;

    private Prototype1(){
        myA = new SomeTypeA();
        myB = new SomeTypeB();
        myC = new SomeTypeC();

        myA.doHeavyStuff();
        myB.doHeavierStuff();
        myC.doReallyHeavyStuff();
    }

    public Prototype1 clone() throws CloneNotSupportedException{
        Prototype1 output = (Prototype1)super.clone();
        output.myA = myA.clone();
        output.myB = myB.clone();
        output.myC = myC.clone();
        return output;
    }

    public static Prototype1 getClone() throws CloneNotSupportedException{
        if(myPrototype == null){
            myPrototype = new Prototype1();
        }
        return myPrototype.clone();
    }
}
```

```
public class Prototype2 implements Cloneable{

    private static SomeTypeA myA = new SomeTypeA();
    private static SomeTypeB myB = new SomeTypeB();
    private static SomeTypeC myC = new SomeTypeC();

    private Prototype2(){
        myA.doHeavyStuff();
        myB.doHeavierStuff();
        myC.doReallyHeavyStuff();
    }

    public Prototype2 clone() throws CloneNotSupportedException{
        return (Prototype2)super.clone();
    }

    public static Prototype2 getClone(){
         return this.clone();
    }
}



public class Prototype3 implements Cloneable{

    private static Prototype3 myPrototype = new Prototype3();
    private SomeTypeA myA;
    private SomeTypeB myB;
    private SomeTypeC myC;

    private Prototype3(){
        myA = new SomeTypeA();
        myB = new SomeTypeB();
        myC = new SomeTypeC();
        myA.doHeavyStuff();
        myB.doHeavierStuff();
        myC.doReallyHeavyStuff();
    }

    public static Prototype3 clone() throws CloneNotSupportedException{
        return (Prototype3)super.clone();
    }
}
```

b) Singleton

```
public class Singleton1{

    private Singleton1 theInstance = new Singleton1();

    private Singleton1(){
    }

    public Singleton1 getInstance(){
        return theInstance;
    }

}


public class Singleton2{

    private static Singleton2 theInstance = new Singleton2();

    private Singleton2(){
    }

    public static Singleton2 getInstance(){
        return theInstance;
    }

}


public class Singleton3{

    private static Singleton3 theInstance = new Singleton3();

    private Singleton3(){
    }

    public Singleton3 getInstance(){
        return theInstance;
    }

}
```

c) Lock

```
public class Lock1{

    private static Object myLock = new Object();
    private int a = 0;

    public void incrementA(){
        synchronized(myLock){
            a++;
        }
    }

    public void decrementA(){
        synchronized(myLock){
            a--;
        }
    }

    public int getA(){
        synchronized(myLock){
            return a;
        }
    }
}


public class Lock2{

    private int a = 0;

    public synchronized void incrementA(){
        a++;
    }

    public synchronized void decrementA(){
        a--;
    }

    public synchronized int getA(){
        return a;
    }
}
```

```
public class Lock3{

    private Object myLock = new Object();

    private static int a = 0;

    public void incrementA(){
        synchronized(myLock){
            a++;
        }
    }

    public void decrementA(){
        synchronized(myLock){
            a--;
        }
    }

    public int getA(){
        synchronized(myLock){
            return a;
        }
    }
}
```

d) Flyweight

```
public class Flyweight1{

    private HeavyType myHeavy;

    private Flyweight1(){
        myHeavy = new HeavyType();
    }

    public Flyweight1 getNewFly(){
        Flyweight1 newFly = new Flyweight1();
        newFly.myHeavy = this.myHeavy;
        return newFly;
    }
}
```

```
public class Flyweight2 implements Cloneable{

    private HeavyType myHeavy;

    private Flyweight2(){
        myHeavy = new HeavyType();
    }

    public Flyweight2 clone(){
        Flyweight2 newFly = super.clone();
        newFly.myHeavy = this.myHeavy;
        return newFly;
    }

    public Flyweight2 getNewFly(){
        return this.clone();
    }
}
```

```
public class Flyweight3{

    private static final HeavyType myHeavy = new HeavyType();

    public Flyweight3(){
    }

}
```

3. Medan många designmönster har som främsta ändamål att göra det lättare för programmeraren, genom att t.ex underlätta kodåteranvändning eller göra det lättare att förstå hur olika klasser ska användas, syftar andra mönster främst till att förbättra det färdiga programmets egenskaper, så att det kan köras effektivare och med färre exekveringsfel. För varje designmönster i den följande listan, ange om det främst syftar till att underlätta vid programmering (**P**) eller till att förbättra körningen (**K**). 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) `Facade`
   b) `Lock`
   c) `Builder`
   d) `Observer`
   e) `Flyweight`

4. För varje påstående om nyckelord 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) Om ett fält deklareras med nyckelordet `static` så kan dess innehåll inte ändras efter initiering.
   b) En metod som deklareras som `protected` kan bara anropas från klassen självt.
   c) Om en klass deklareras med nyckelordet `final` så kan den inte ärvas av någon annan klass.
   d) Fält kan deklareras med högst två nyckelord per fält.
   e) Metoder som deklareras med nyckelordet `synchronized` får inte samtidigt deklareras som `private`.

5. För varje påstående om klasser och instanser 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) Det går att skapa klasser som inte går att instansiera.
   b) Klassmetoder kan anropas utan att man explicit skapat någon instans av klassen.
   c) En instansmetod får bara använda instansvariabler.
   d) En klassmetod får bara använda klassvariabler.
   e) Man instansierar en klass genom att ärva från den.

6. 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`'. Notera att metoden `sleep()` kan kasta en `InterruptedException`.

(5 p)

a)
```
public class ClassA{

    public int a = 1;

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

b)
```
public class ClassB{

    private int a = 1;
    static ClassB myB = new ClassB();

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


}
```

[1]För tydlighets skull anses här Java 7, som finns i skolans datasalar

c)
```
public class ClassC extends Thread{

    private static int a = 1;

    public static void main(String[] args){
        for(int i = 0; i < 1000; i++){
            ClassC myC = new ClassC();
            myC.start();
        }

        printResult();

    }

    public void run(){
        try {
            sleep(500);
        } catch(InterruptedException e){}
        incrementA();
    }

    private synchronized void incrementA(){
        a++;
    }

    private static synchronized void printResult(){
        try {
            sleep(500);
        } catch(InterruptedException e){}
        System.out.println(a);
    }
}
```

d)
```
private class ClassD{

    static int a = 1;

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

e)
```
public class ClassE implements Runnable{

    private int a = 1;
    private static ClassE myClassE = new ClassE();

    public static void main(String[] args){
        for(int i = 0; i< 1000; i++){
            Thread myThread = new Thread(new ClassE());
            myThread.start();
        }

        System.out.println(myClassE.a);
    }

    public void run(){
        try {
            Thread.sleep(500);
        } catch(InterruptedException e){}
        a++;
    }
}
```

Del II - fördjupningsfrågor

Följande uppgifter besvaras på separat papper.

7. Vad menas med *lazy initialisation*? Ge exempel på hur det kan användas i två olika designmönster. (3 p)

8. Förklara designmönstret *Iterator*. Vad är dess syfte, hur implementerar man det, och vilka fördelar har det (eller nackdelar, om dessa är mer relevanta)? (2 p)

9. Du har just fått ett spännande och utvecklande sommarjobb på Centrum för Autonoma System (CAS). CAS är ett forskningscenter på KTH som också ger kurser, bl. a i robotprogrammering. Din arbetsuppgift är att skapa en grundstruktur i Java som studenterna kan använda för att programmera robotar i en sådan kurs. Robotarna har två motordrivna hjul som de ska balansera på, accelerometrar, sonar och andra enkla sensorer. Datorn som styr varje robot är jämförbar med en modern laptop.

   Den kod som finns på robotarna i dagsläget består av ett antal programmoduler (troligtvis skrivna i C, men källkod saknas) som körs i bakgrunden och som kan kommunicera över UDP-sockets. Varje modul styr en bit hårdvara, t.ex en motor, eller en sensor. Modulerna lyssnar efter anslutningar på en egen port, och kan efter anslutning skicka eller ta emot enkla XML-meddelanden. Med dessa meddelanden kan man t. ex stänga av eller sätta på sensorer, be dem att skicka sitt senaste mätvärde, eller be en hjulmotor att rotera med en viss hastighet. Modulerna kan typiskt klara av kommunikationshastigheter upp mot ett par tusen meddelanden i sekunden, vilket är tillräckligt för att köra avancerade reglersystem.

   Kursdeltagarna ska skriva enklare kod får att få robotarna att balansera, åka runt i labyrinter, och lösa olika enklare uppgifter. För att debugga sin kod måste de kunna logga all information som genereras i olika delar av programmet på ett bra sätt. De förväntar sig ett antal lättanvända Java-klasser som de kan använda till att skriva den kod som gör allt de vill kunna göra med robotarna, och ska inte behöva lägga tid på sånt som inte direkt har med koden för robotstyrning att göra.

   Beskriv hur du bygger upp Javastrukturen som studenterna ska programmera mot. 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! Du får använda UML eller (pseudo-)kod i ditt svar om du tycker att det förenklar presentationen, men det är inte ett krav. (8 p)

10. Ange tre olika designmönster som ger *lös koppling*, och förklara hur för vart och ett av dem. (6 p)

11. En av dina vänner har skrivit ett program för interaktiva bakgrundsbilder på smartphones. En användare **A** kan installera en bakgrundsbild på sin smartphone, som sedan en annan användare **B** kan ändra på, på sin telefon. T.ex kan bilden föreställa ett cafe, där **B** kan andra på inredningen, besökarna, belysning, vädret utanför, mm. **A** ser då alltid den senaste versionen av bilden som bakgrund på sin telefon. Innan din vän släpper denna app till försäljning vill hen ha din hjälp att lösa ett smärre problem.

    Det är nämligen så att det går väldigt långsamt att uppdatera bilden över nätverket. För **A** spelar det ingen roll, eftersom hen troligtvis inte sitter och tittar på bakgrundsbilden hela tiden, utan kan stå ut med att den uppdateras långsamt eller sporadiskt, men för **B** blir det direkt plågsamt att varje liten förändring eller justering tar en eller flera sekunder att genomföra, för att inte tala om att det inte går alls om **A** tillfälligt saknar nätverkstäckning.

    Beskriv konkret hur din vän lämpligast löser sina problem, och namnge korrekt alla designmönster som används i lösningen. (3 p)

12. Du behöver integrera ett tredjepartsbibliotek för grafik i en befintlig javaapplikation som ditt företag utvecklar. Tidigare har ni använt ett egetutvecklat bibliotek, men för att få bättre prestanda vill ni byta till det nya tredjepartsbiblioteket. Ditt företag hyrde in en dyr konsult som skissade upp nedanstående lösning i UML. Vilket mönster är det som föreslås? Hur implementerar man konsultens förslag? Kommer det att fungera som tänkt, eller behövs det några ändringar? Förklara! (3 p)