Skolan för datavetenskap och kommunikation

Objektorienterad Programkonstruktion, DD1346

Tentamen 2014–03–20, 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 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)

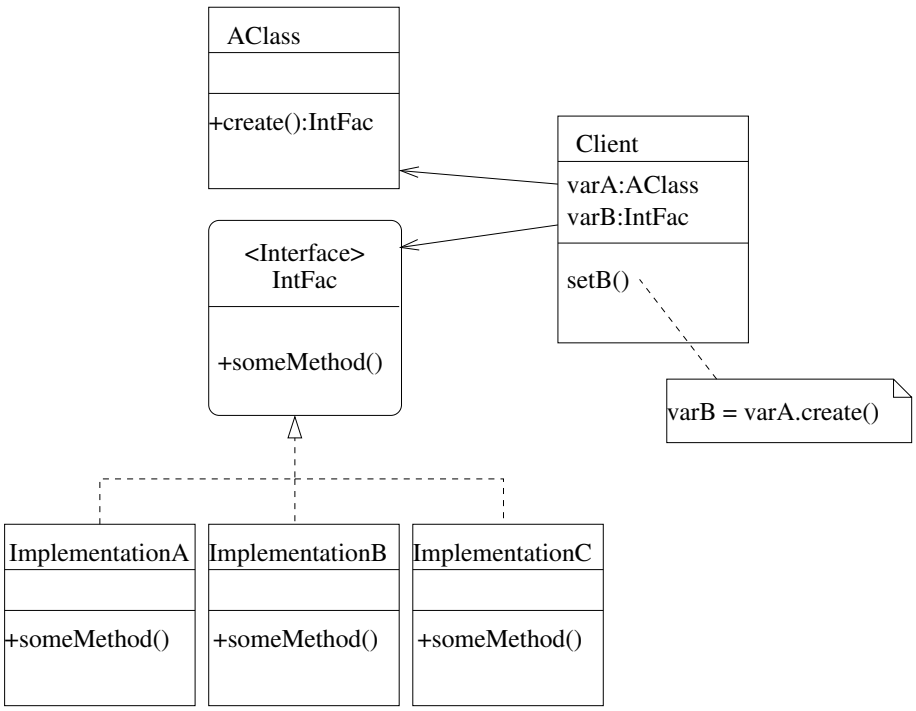
Lycka till!

Del I - flervalsfrågor

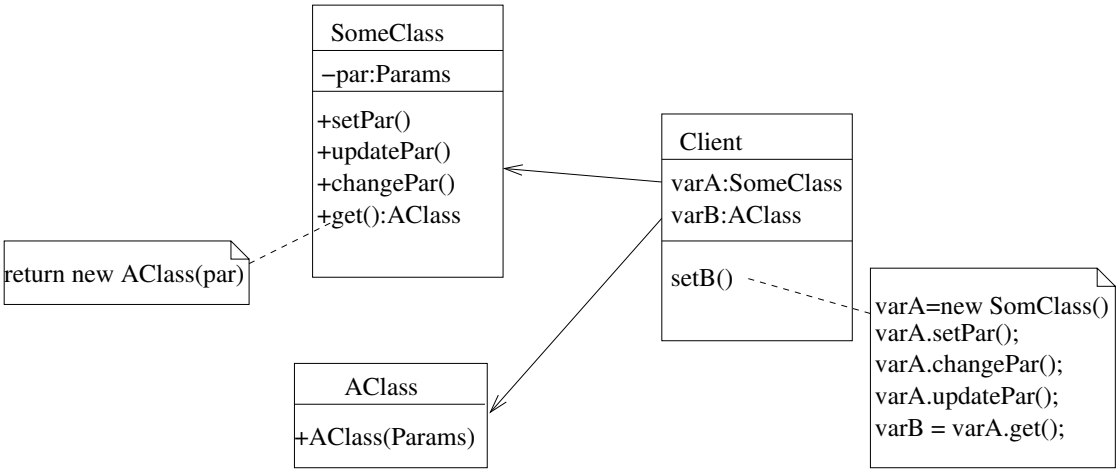
1. I denna uppgift finns 5 uppsättningar UML-diagram. *Generiska namn används i stället för de mer vanliga namn som avslöjar vilket mönster det är.* För varje diagram, ange det designmönster som bäst beskriver det. Välj från listan nedan. Varje korrekt angivet designmönster ger 1 p. (5 p)

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

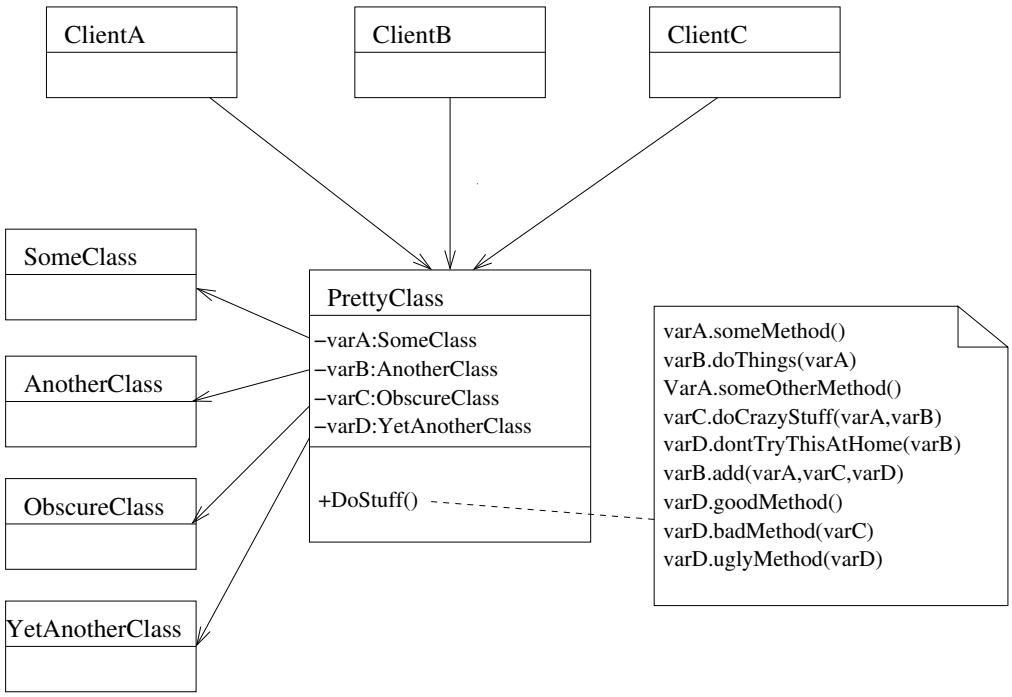
a)



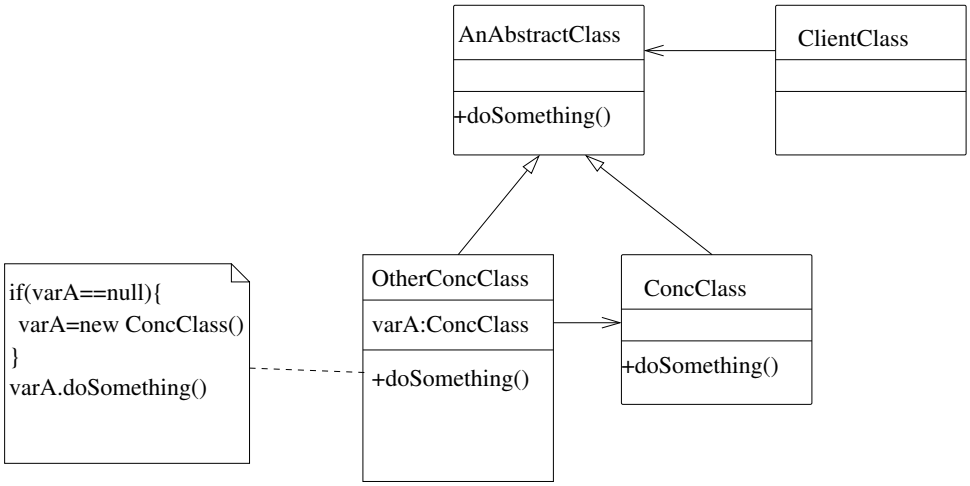
b)


SomeClass
−par:Params
+setPar()
+updatePar()
+changePar()
+get():AClass
return new AClass(par)
Client
varA:SomeClass
varB:AClass
setB()
varA=new SomClass()
varA.setPar();
varA.changePar();
varA.updatePar();
varB = varA.get();
AClass
+AClass(Params)

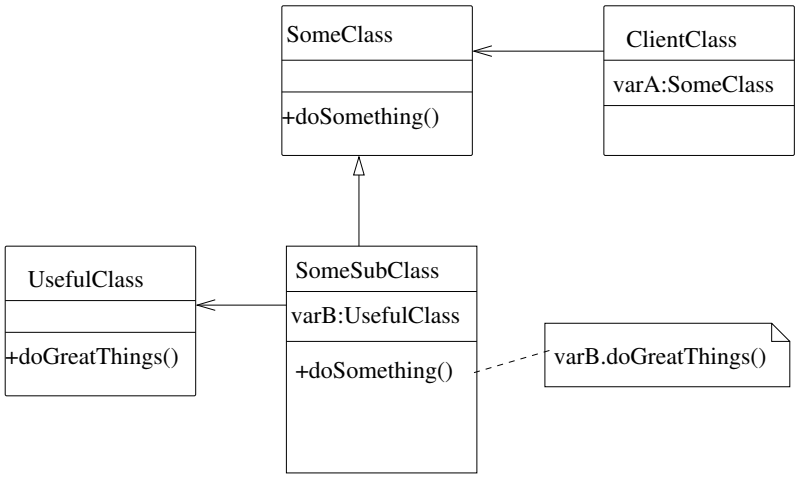
c)


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)

d)


AnAbstractClass
+doSomething()
ClientClass
if(varA==null){
varA=new ConcClass()
}
varA.doSomething()
OtherConcClass
varA:ConcClass
+doSomething()
ConcClass
+doSomething()

e)


SomeClass
+doSomething()
ClientClass
varA:SomeClass
UsefulClass
+doGreatThings()
SomeSubClass
varB:UsefulClass
+doSomething()
varB.doGreatThings()

2. Nedan följer 3 exempel på Java-program. För varje program, ange om programmet ger en korrekt implementation av en **singleton**. Varje korrekt analyserat program ger 1 p. (3 p).

a)
```
public class SingletonA{

    private static SingletonA theInstance = new SingletonA();

    private SingletonA(){}

    public static SingletonA getInstance(){
        return theInstance;
    }
}
```

b)
```
public class SingletonB{

    private static SingletonB theInstance = null;

    private SingletonB(){
        if(theInstance == null){
            theInstance = new SingletonB();
        }
    }

    public static SingletonB getInstance(){
        return new SingletonB();
    }
}
```

c)
```
public class SingletonC{

    private static SingletonC theInstance = null;

    private SingletonC(){}

    public static SingletonC getInstance(){
        if(theInstance == null){
            theInstance = new SingletonC();
        }
        return theInstance;
    }

}
```

3. Nedan följer 3 exempel på Java-program. För varje program, ange om programmet ger fungerande **trådsäker variabelåtkomst**. Varje korrekt analyserat program ger 1 p. (3 p).

a)
```
public class ThreadSafeA{

    private static volatile int a = 0;

    public void incA(){
        a++;
    }

    public void decA(){
        a--;
    }

    public int getA(){
      return a;
    }

}
```

b)
```
public class ThreadSafeB{

    private static int a = 0;

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

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

    public synchronized int getA(){
        return a;
    }


}
```

c)
```
public class ThreadSafeC{

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

    public void incA(){
        synchronized(lock){
            a++;
        }
    }

    public void decA(){
        synchronized(lock){
            a--;
        }
    }

    public int getA(){
        return a;
    }

}
```

4. För varje påstående om text och strängar 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) `String` är en av Javas inbyggda primitiva datatyper.

   b) En fördel med att använda strängkonkatenering med '+'-operatorn är att det ger snabbare kod om man hanterar långa strängar med många sammanslagningar.

   c) En `Scanner` är en *iterator* för `String`-objekt.

   d) Ett `String`-objekt i Java går inte att ändra i efterhand, dvs den är vad man kallar *immutable*.

   e) Eftersom en `String` är implementerad som en array av `char` så kan den bara innehålla text kodad som `ascii`.

5. För varje påstående om XML, 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) XML är ett utrymmeseffektivt sätt att lagra komplexa datastrukturer, som t.ex mätdataloggar från experiment.

   b) XML passar bra för att lagra data med en explicit trädstruktur.

   c) En anledning till att lagra data i XML-format kan vara att man vill att det ska vara lättfattligt för en mänsklig läsare.

   d) Alla XML-taggar måste alltid förekomma i par, med en start-tag och en slut-tag.

6. För varje påstående om designmönster nedan, 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 fämsta syftet med `Flyweight` är att minska minnesanvändning.

   b) `Composite` används för att minska overhead i parallella program.

   c) *Polymorfism* är ett problem som man ofta kan undvika med `Threadpool`.

   d) Ett vanligt mål med att använda `Observer` är att göra programexekveringen snabbare.

   e) `MVC` är ofta ett lämpligt mönster när man gör program med grafiska användargränssnitt.

7. För varje programlistning nedan, ange om koden går att kompilera eller ej. (4 p)

a)
```
public class ParentClass{

    private int a = 0;

    public class SubClass extends ParentClass{

        private int b;

        public SubClass(){
            b=a;
        }

    }
}
```

b)
```
public class ParentClass{

    public static void main(String[] args){
        SubClass sc = new ParentClass();
    }

    private class SubClass extends ParentClass{
    }

}
```

c)
```
public class ParentClass{

    public static void main(String[] args){
        IntFace ifce = new SubClass();
    }

    private static class SubClass extends ParentClass implements IntFace{
    }

    private interface IntFace{
    }

}
```

d)
```
public final class ParentClass{

    private ParentClass(){
    }

    public static void main(String[] args){
        ParentClass pc = new ParentClass();
        SubClass sc = pc.new SubClass();
    }

    private class SubClass extends ParentClass{
    }

}
```

Del II - fördjupningsfrågor

Följande uppgifter besvaras på separat papper.

8. a) Vad menas med *Lös koppling*? Är det bra eller dåligt? Varför? (2 p)
   b) Ge exempel på hur man kan koda för att åstadkomma respektive undvika lös koppling. (2 p)

9. Förklara skillnaden mellan *abstrakta klasser* och *gränssnitt*, och när man bör använda vilken. (2 p)

10. Förklara vad en **Threadpool** är, och hur och varför man använder den. (2 p)

11. Nedan följer fem olika designmönster. Välj ut tre olika kombinationer av två mönster ur denna lista, och beskriv hur dessa mönsterpar kan kombineras med varandra, och varför man skulle vilja göra det. Notera att det ska handla om en direkt kombination av mönster, inte bara en applikation som råkar innehålla båda mönstren. (6 p)

    Builder Factory Singleton Lock Proxy

12. Du har bestämt dig för att bli rik och berömd och tänker åstadkomma detta genom att ge dig in i spelbranschen och börja producera enkla billiga spel för olika plattformar som stöder Java. Din marknadsresearch säger dig att det är ganska små skillnader mellan framgångsrika och misslyckade spel, men att slumpen verkar spela stor roll. För att komma runt detta tänkte du börja med att programmera ett praktiskt ramverk som du kan återanvända för att göra olika spel, så att du enkelt kan producera ett stort antal olika spel med förhoppningen att något av dem slår igenom. För att minimera risken att någon stämmer dig på alla dina miljoner när framgången väl kommer, har du beslutat att inte använda några tredjepartsbibliotek.

    Du tänker dig att alla dina spel ska vara enkla och actionbetonade, och ska innehålla ett antal olika figurer och föremål som kan interagera med varandra enligt olika regler, i stil med Angry Birds, Cut The Rope, Flappy Bird, Worms eller liknande succéspel.

    Beskriv hur du konstruerar ditt ramverk för att göra dina enkla spel i, så att du får en bra arkitektur med så mycket återandvändbar och lättmodifierad kod som möjligt. Vilka designmönster använder du, hur, och varför? (8 p)

13. Du hjälper en vän att skriva ett enkelt fotoredigeringsprogram i Java, och enligt din rekommendation använder hen färdiga Swingkomponenter till användargränssnittet. I ritprogrammet kan man applicera olika filter på sin bild genom att trycka på någon av ett antal olika knappar. Vissa filter kan dock ta väldigt lång tid att köra, och din vän lägger för säkerhets skull till en "avbryt"-knapp. Tyvärr fungerar det inte som hen hade tänkt sig, eftersom det inte går att trycka på någon knapp, inklusive "avbryt", medan filterkoden körs, utan resultatet av knapptryckningarna kommer först när filtreringen är färdig. Din vän undrar vad som händer och hur hen ska fixa det. Ge ett hjälpsamt svar! (3 p)