# Tentamen

DD2385 Programutvecklingsteknik vt 2011

Tisdagen den 24 maj 2011 kl 14.00 – 17.00

Hjälpmedel: penna, suddgummi, linjal

Tentan har två delar om vardera 30 poäng

Maximala betygsgränser (gränserna kan bli lägre men inte högre):

| | |
|---|---|
| Betyg FX: | 22-23 poäng på del I |
| Betyg E: | minst 24 poäng på del I |
| Betyg C: | Godkänd del I och minst 44 poäng på hela tentan |
| Betyg A: | Godkänd del I och minst 56 poäng på hela tentan |

Betyg FX får kompletteras till C eller A om
totalpoängen efter komplettering räcker till C respektive A.

Bonuspoäng från labbarna (max 6) läggs till del II-resultatet innan betyget sätts.

Skriv gärna flera uppgifter på samma papper

*men tag nytt papper när du börjar på del II !*

# del I

(5p) **1.** Nedan följer **sju** namn på designmönster och därefter **fem** korta beskrivningar av designmönster. Välj det rätta mönsternamnet till varje beskrivning. Två mönsternamn blir alltså över!

| | | | |
|---|---|---|---|
| **Iterator** | **Mediator** | **Observer** | **Proxy** |
| **Facade** | **Mock Object** | **Singleton** | |

**A)** Ett objekt kontrollerar åtkomsten till ett annat objekt. De båda implementerar samma gränssnitt.

**B)** En enda klass ger ett enkelt gränssnitt till ett komplext system.

**C)** Det går bara att skapa *ett enda* objekt av en viss klass.

**D)** När tillståndet i ett objekt ändras så uppdateras ett antal andra objekt som "prenumererar" på uppdateringsinformation.

**E)** Man får tillgång till objekt i sekvens, utan att känna till hur de egentligen är organiserade.

(9p) **2.** Rita ett UML-klassdiagram som åskådliggör följande klasser och interface. Alla relationer ska vara med. För full poäng måste öppna och slutna pilspetsar samt streckade och heldragna linjer användas rätt. Fyllda romber behöver *inte* användas. Instansvariabler och metoder behöver *inte* vara med i diagrammet men ev. relationer som ges av variabler och metoder ska ritas ut. Alla klasser och interface som nämns i koden *utom* **ArrayList** ska vara med i diagrammet. Långa namn får förkortas om det är helt klart vilket namn som avses, t.ex. `AAA` för `AbstractAnimalAnimator`.

```
public interface Animal {
    public void move();
}

class Butterfly implements Animal { ... }

class BlueMorpho extends Butterfly { ... }

class LeopardLacewing extends Butterfly { ... }

abstract class AbstractAnimalAnimator {
    ArrayList<Animal> alist;

    abstract void initAnimator(int n);

    AbstractAnimalAnimator (int n) {
        alist = new ArrayList<Animal>();
        initAnimator(n);
    }

    public void moveAllAnimals() {
        for (Animal a : alist)
            a.move();
    }

    // ... fler metoder ...
}

class ButterflyAnimator extends AbstractAnimalAnimator {

    ButterflyAnimator(int n) {
        super(n);
    }

    void initAnimator(int n) {
        for (int i=0; i<n; i++){
            if (Math.random() > 0.5)
                alist.add(new BlueMorpho());
            else
                alist.add(new LeopardLacewing());
        }
    }
}
```

(1p) **3.** Vilket designmönster används av klasserna `AbstractAnimalAnimator` och `ButterflyAnimator` i uppgift 2 ?

(4p) **4.** Vad är ett *ramverk* och vad är en *komponent* i programutvecklinggsammanhang? Ge ett exempel på Java-ramverk och ett exempel på en Java-komponent.

(2p) **5. a** Ange minst två karakteristiska drag för *agila/lättviktiga* metodiker för programutveckling eller ange fyra av XP-reglerna som används inom *eXtreme Programming*.

(2p) **5. b** Ange två egenskaper hos *Vattenfallsmetoden* för programutveckling där denna metod skiljer sig från de lättviktiga metoderna.

(2p) **6. a** Skriv om följande kodavsnitt kortare och tydligare.

```
for (int i=0; i<5; i++) {
    if (i==2)
        c();
    else if (i==0)
        a();
    else if (i==1)
        b();
    else
        d();
}
```

(1p) **6. b** Inom programutveckling, vad kallas det för när man skriver om och förbättrar ett programavsnitt utan att funktionen ändras?

**A)** Testdriven programutveckling **B)** Refactoring **C)** Profilering

**7.** Antag att följande deklarationer i Java är gjorda:

```
class A {...}

class B extends A { ... }
```

(1p) **a.** Vilken eller vilka typer av referenser kan referera till objekt av klassen `A`?

**A)** endast typen `A`
**B)** endast typerna `A` och `Object`
**C)** endast typerna `A`, `B` och `Object`

(1p) **b.** Vilken eller vilka typer av referenser kan referera till objekt av klassen `B`?

**A)** endast typen `B`
**B)** endast typerna `B` och `A`
**C)** endast typerna `B`, `A`, och `Object`

(2p) **8.** Om variabeln `a` refererar till en Java-objektsamling, t.ex. en `ArrayList` eller `LinkedList`, så kan man under vissa förutsättningar få samlingen sorterad genom att anropa metoden `Collections.sort(a)`. Vad måste gälla för att sorteringen ska gå att genomföra?

# del II

(4p) **9.** Begreppet **static** är svårt. Markera för **A** resp. **B** om den eller de angivna metoderna är **static** eller ej. Svara på frågan **C**.

**A)** Metoderna `sin` och `cos` i klassen `Math` (1p).

**B)** Metoden `adjustmentValueChanged` i interfacet `AdjustmentListener`. Metoden implementeras av lyssnare som lyssnar efter händelser i t.ex. `JScrollBar`s. (1p)

**C)** Förklara noga varför följande program inte går att kompilera: (2p)

```
class Fel {
    int finttal = 1729;

    public static void main(String[] u) {
        System.out.println("Fina talet = " + finttal);
    }
}
```

(3p) **10.** Antag att klassen `Damspelsmodell` är stor och komplicerad och har ca 20 metoder. Klassen `Speltestare` använder `Damspelsmodell` vilket antyds i kodsnutten nedan men endast fyra av metoderna i `Damspelsmodell` anropas av `Speltestare`.

Beskriv begreppet *lös koppling* mellan klasser. Visa hur man kan ändra på beroendet mellan `Speltestare` och `Damspelsmodell` så att de blir just *löst kopplade*. Använd gärna UML-klassdiagram och/eller programkod i svaret. Ett tydligt svar med bara ord kan dock ge full poäng.

```
class Speltestare {
    Damspelsmodell modellen;

    Speltestare (Damspelsmodell dm) {
        modellen = dm;
        ...
    }
    ...
}
```

**11.** Ett filsystem består av *filer* och *kataloger*. Kataloger kan innehålla filer och andra kataloger som i sin tur kan innehålla filer och kataloger o.s.v. enligt mönstret *Composite*. Filer representeras av objekt av klassen `File` som finns här nedanför. Kataloger representeras av objekt av `Directory`. Både filer och kataloger har namn som lagras i instansvariabeln `name` i den gemensamma superklassen `FileElement`. Observera att `File` här inte är samma klass som i Java-API:n. Det är inte "riktig" filhantering i uppgiften och inga referenser till "riktiga" `File` ska införas. Använd bara den givna `File`.

```
abstract class FileElement {

    String name;
    Directory parent = null;

    FileElement (String n) {
        name = n;
    }

    public String toString() {
        return name;
    }

    String path() {
        // metodkroppen är uppgift 11a
    }

    abstract void add(FileElement fe);
    abstract void remove(FileElement fe);
    abstract void printXML(); //lägg ev. till en parameter !!!
}

class File extends FileElement {

    File (String n) {
        super(n);
    }

    void add(FileElement fe) {}
    void remove(FileElement fe) {}

    void printXML() { // lägg ev. till en parameter !!!
        System.out.println("<FILE>" + name + "</FILE>");
    }
}
```

(3p) **a.** Skriv metoden `path()` i klassen `FileElement`. Metoden ska returnera en `String` med sökvägen från trädets rot till filen eller katalogen. Om filen med namnet "Citronsås" ligger i katalogen "recept" som ligger i katalogen "hem" så ska `cs.path()` returnera `''hem/recept/Citronsås''`, förutsatt att `cs` refererar till filobjektet med name-attributet "Citronsås".

(12p) **b.** Skriv klassen `Directory` för filkataloger enligt mönstret *Composite*. Ett `Directory` ska kunna innehålla objekt av `Directory` och objekt av `File`. Följande metoder ska finnas i `Directory`:

- `add` för att lägga till ett `FileElement`–objekt (2p)
- `remove` för att ta bort ett `FileElement`–objekt (1p)
- `printXML` för att skriva ut en XML-version av filsystemet (4p)
  Om indenteringen (indraget av vänstermarginalen vid utskriften) speglar trädjupet så ges ytterligare (2p). Se exempel på nästa sida!

För resten av klassen `Directory` (instansvariabler, konstruktor) ges (3p).

(5p) **c.** Komplettera klasserna med en metod som söker efter objekt (filer eller kataloger) med ett givet namn. En lista med sökvägar ska returneras. T.ex. om en katalog med namn `elevprog` finns både under `hem/kurser/grupdat/knepigt/` och under `hem/kurser/prutt/` så ska metoden returnera en lista :
`['hem/kurser/grupdat/knepigt/elevprog', 'hem/kurser/prutt/elevprog']`
Om det sökta namnet bara finns på ett ställe returneras en lista med ett element. Om namnet inte finns alls så returneras en tom lista. Fler exempel finns på nästa sida!

Metodhuvud:

```
ArrayList<String> findFiles(String target)
```

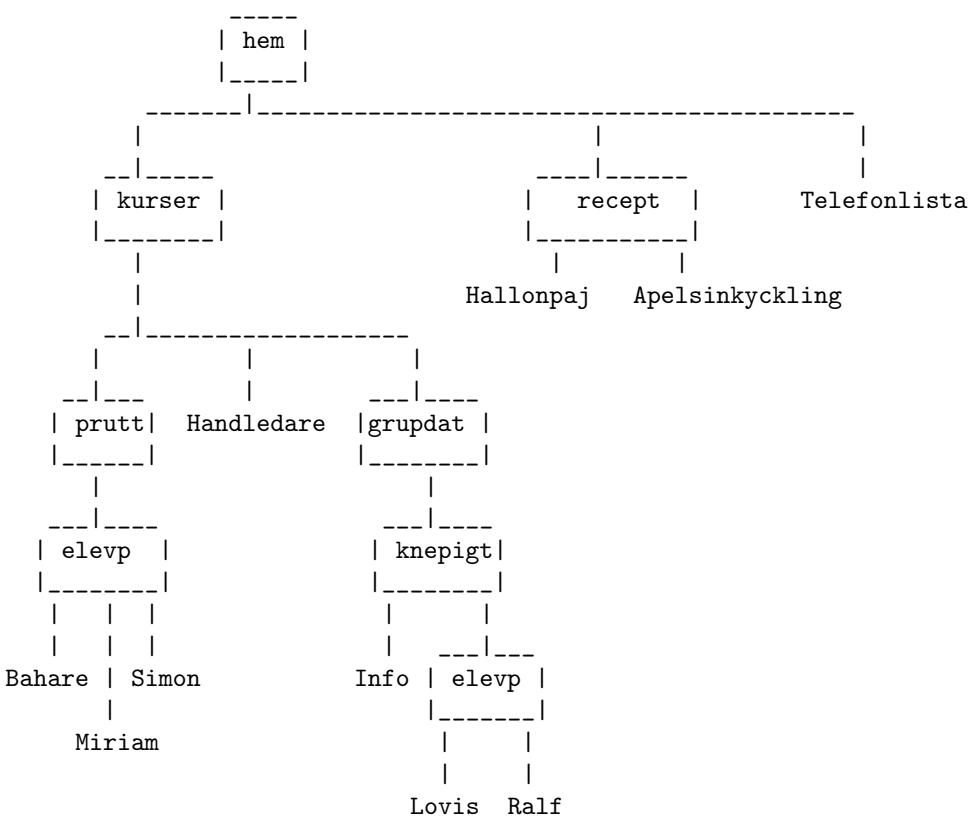
Metoden måste gå att anropa för `Directory`–objekt men välj själv i vilka klasser den ska implementeras! Flera möjligheter finns.

Några av dessa hjälpmedel kan behövas i **b** och **c**:
`coll.remove(obj)` – tar bort `obj` ur samlingen `coll`
`coll.add(obj)` – lägger till `obj` till samlingen `coll`
`coll.addAll(collny)` – lägger till alla element från `collny` till samlingen `coll`
Operatorn `instanceof` används så här: `obj instanceof A` ger `true` om objektet `obj` har typen `A`

Perfekt Java-syntas krävs *inte* för full poäng på någon uppgift!

(3p) **d.** Sökningen i uppgift **c** söker endast efter exakt överensstämmelse på namnet. Nu vill vi kunna variera sökkriteriet, t.ex. söka efter namn som börjar på, slutar på eller innehåller en viss textsträng eller har någon annan egenskap. Hur kan man ordna så att en enda sökmetod `findFiles(...)` använder olika jämförelsekriterier vid olika anrop? Fullständig Java-kod behövs inte. Ett svar i bara ord eller lite kod + ord duger bra om det är tydligt. Ange gärna namnet på ett passande designmönster också! Mönsternamnet krävs *inte* för full poäng men kan kompensera *i viss mån* för en otydlig beskrivning. Det går bra att svara på den här frågan även om du inte löst något annat på uppgift **11**.

Exempel till uppgift 11.



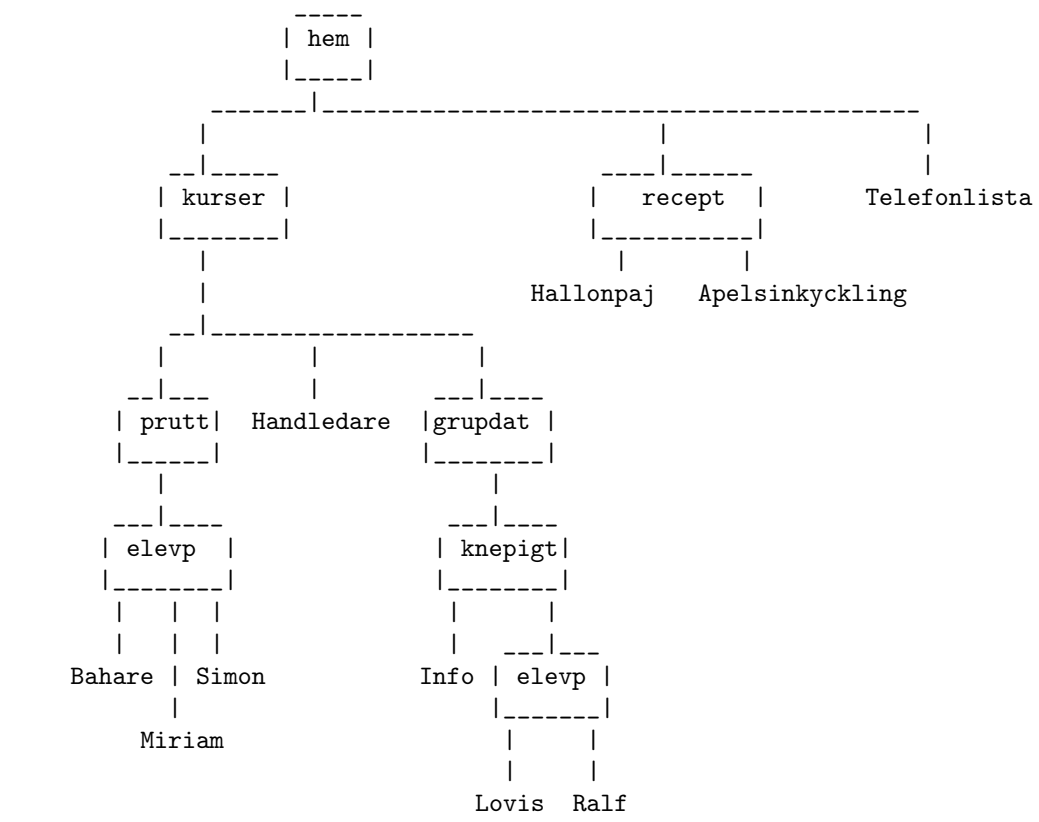

Anrop av `path` för Ralf ska ge
`hem/kurser/grupdat/knepigt/elevp/Ralf`

Sökning efter ‘‘`elevp`’’ på katalogen `hem` ska ge listan
[‘‘`hem/kurser/grupdat/knepigt/elevp`’’, ‘‘`hem/kurser/prutt/elevp`’’]
Ordningen inom listan spelar ingen roll.

Sökning efter ‘‘`elevp`’’ på katalogen `grupdat` ska ge listan
[‘‘`grupdat/knepigt/elevp`’’] och samma sökning på katalogen `recept` ger tom lista.

Resultatet av `printXML()` på katalogen `grupdat` visas här. XML för hela katalogen `hem` finns på nästa sida.

```
<DIRECTORY>grupdat
  <DIRECTORY>knepigt
    <FILE>Info</FILE>
    <DIRECTORY>elevp
      <FILE>Lovis</FILE>
      <FILE>Ralf</FILE>
    </DIRECTORY>
  </DIRECTORY>
</DIRECTORY>
```




```
<DIRECTORY>hem
  <DIRECTORY>kurser
    <DIRECTORY>prutt
      <DIRECTORY>elevp
        <FILE>Bahare</FILE>
        <FILE>Miriam</FILE>
        <FILE>Simon</FILE>
      </DIRECTORY>
    </DIRECTORY>
    <FILE>Handledare</FILE>
    <DIRECTORY>grupdat
      <DIRECTORY>knepigt
        <FILE>Info</FILE>
        <DIRECTORY>elevp
          <FILE>Lovis</FILE>
          <FILE>Ralf</FILE>
        </DIRECTORY>
      </DIRECTORY>
    </DIRECTORY>
  </DIRECTORY>
  <DIRECTORY>recept
    <FILE>Hallonpaj</FILE>
    <FILE>Apelsinkyckling</FILE>
  </DIRECTORY>
  <FILE>Telefonlista</FILE>
</DIRECTORY>
```