Tentamen

DD2385 Programutvecklingsteknik vt 2012

Lördagen den 2 juni 2012 kl 9.00 – 12.00

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

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

del I

Skriv helst flera uppgifter på samma blad på 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** **Strategy**

A) En klass ger ett enkelt gränssnitt till ett komplext system (som kan bestå av många klasser).

B) En del av en algoritm är utbytbar medan programmet kör, ingen omkompilering behövs för att byta. Den utbytbara delen är inkapslad i ett objekt.

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

D) Ett objekt har hand om kommunikationen mellan n st andra objekt. De kommunicerande objekten är oberoende.

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 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 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 RedLacewing extends Butterfly { ... }

class BluePansy extends Butterfly { ... }

abstract class AbstractAnimalAnimator {
    ArrayList<Animal> alist;

    abstract void init(int n);

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

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

    // ... fler metoder ...
}

class ButterflyAnimator extends AbstractAnimalAnimator {

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

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

(2p) **3a.** Klasserna `AbstractAnimalAnimator` och `ButterflyAnimator` i uppgift 2 använder mönstret *Template Method*. Förklara hur det syns i koden att det är just detta mönster.

(1p) **3b.** Ange namnet på *en* metod som definitivt måste definieras i klassen `Butterfly` om klassen ska gå att kompilera.

(2p) **4a.** Ange två egenskaper hos *Vattenfallsmetoden* för programutveckling där denna metod skiljer sig från de *lättviktiga/agila* metoderna.

(2p) **4b.** Ange minst två karakteristiska drag för *lättviktiga/agila* metodiker för programutveckling. Det måste vara andra egenskaper än de som är motsatser till svaren i 4a. Svar som gäller för XP (*eXtreme Programming*) godtas även om de inte tillämpas inom alla lättviktiga metodiker!

(2p) **4c.** Beskriv utvecklingsmetodiken *Rapid Prototyping*.

(3p) **5.** I programutveckling står **MVC** för Model-View-Control. Förklara kortfattat hur ett program är uppbyggt som följer MVC-principen. Det kan räcka med 1-2 meningar för vardera Model, View och Control.

6. Klassen `RationalNumber` representerar rationella tal.

```
class RationalNumber {
    int t, n;
    RationalNumber(int t, int n) {
        this.t = t;
        this.n = n;
    }

    public String toString() {
        return t + "/" + n;
    }
}
```

Följande kodsnutt skapar rationella tal (nödvändiga `import` kan antas ha gjorts):

```
ArrayList<RationalNumber> lista1 = new ArrayList<RationalNumber>();
ArrayList<RationalNumber> lista2 = new ArrayList<RationalNumber>();
Random random = new Random();
int n = 11;
for (int i=0; i<n; i++){
    RationalNumber
        r = new RationalNumber(random.nextInt(20), 1+random.nextInt(19));
    lista1.add(r);
    lista2.add(r);
}
```

(2p) **6a.** Hur många objekt av `RationalNumber` skapas i koden ovan? Rätt svarsalternativ ger 1p och en korrekt motivering av svaret ger ytterligare 1p.

A) 1 **B)** 11 **C)** 20 **D)** 22

(2p) **6b.** Nu vill man sortera listorna med rationella tal och försöker med `Collections.sort(lista1);` men anropet går inte att kompilera.
Klassen `Collections` finns i paketet `java.util` och antas vara importerad och tillgänglig. Vilket krav måste listelementen uppfylla för att det ska gå att sortera dem med det angivna metodanropet? Hur ska `RationalNumber` ändras för att sorteringen ska gå att utföra? Fullständig javakod behövs inte!

del II

Skriv gärna alla svar på uppgift 7 på samma blad men tag nytt blad för uppgift 8!

7. Studera skissen av klassen `Person` nedan. Klassens s.k. fabriksmetod ska skapa ett objekt av någon av subklasserna `Hon, Hen` eller `Han`. Vilken subklass som väljs beror på parametern `info` men vi bryr oss inte om hur det går till. Subklasserna antas korrekta och tillgängliga på rätt katalog.

```
public class Person {

    public Person createPerson(String info) {
        if (...)              // ... står för ett villkor för Hon
            return new Hon();
        else if (...)         // ... står för villkor för Hen
            return new Hen();
        else
            return new Han();
    }
}
```

Så här vill vi anropa fabriksmetoden (standaranrop av sådan):

```
Person unknown = Person.createPerson("ACTGGGADG")
```

Anropet sker i en `main`-metod och ger kompileringsfel. Det spelar ingen roll om `main`-metoden ligger i `Person` eller i en annan klass.

(2p) **a.** Förklara noga varför metodanropet ovan inte går att utföra. Svara särskilt på följande: Vilket slags metoder kan anropas som i exemplet? Vilket slags metod är den definierade `createPerson()`?

(1p) **b.** Vad ska ändras för att metoden `createPerson` ska fungera som det är tänkt?

(1p) **c.** Om felet åtgärdas så går det att skapa objekt av `Person` med `new Person()`. Hur gör man för att säkerställa att `new Person()` inte går att göra någonstans *utanför* klassen `Person` men går att göra *inuti* klassen `Person`?

(1p) **d.** Hur gör man för att se till att `new Person()` inte är möjligt någonstans, varken utanför eller inuti klassen. Alltså, hur definierar man en klass som inte går att instansiera överhuvudtaget?

(2p) **e.** Klasser som inte går att instansiera (skapa objekt av) används i flera sammanhang. Ett exempel på användning som är vanligt i objektorienterad (OO) programmering involverar s.k dynamisk bindning. Vad är dynamisk bindning?

(1p) **f.** Klasser som inte ska instansieras har en tillämpning som inte alls är "objektorienterad", inte behöver innehålla några som helst objekt. Vilken tillämpning är det? Svar i form av ett exempel från Javabiblioteket räcker. En allmän beskrivning är ett bra alternativ förstås.

Frågorna 7a – 7f kan besvaras oberoende av varandra

8. 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; // alla FileElement utom trädets rot har en parent

    FileElement (String n) {
        name = n;
    }

    public String toString() {
        return name;
    }


    abstract void add(FileElement fe);
    abstract void remove(FileElement fe);
    abstract File largestFile(); // returns a reference to its largest File
    abstract void addPrefix();
  }

class File extends FileElement {
    int size;

    File (String n, int s) {
        super(n);
        size = s;
    }

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

    File largestFile() {
        // Uppgift a
    }

    void addPrefix() {
        // Uppgift a
    }
}
```

(3p) **8a.** Komplettera klassen `File`. Metoden `largestFile()` ska returnera en referens till filobjektet självt, den enda tänkbara filen. Metoden `addPrefix` ska ändra filens namn genom att lägga till `"F:"` före filnamnet. Filen med namn `"Miriam"` byter namn till `"F:Miriam"`.

(14p) **8b.** Skriv klassen `Directory` för filkataloger enligt mönstret *Composite*. Ett `Directory` ska kunna innehålla objekt av `Directory` och objekt av `File`. Följande metoder ska skrivas i `Directory`:

- `add()` för att lägga till ett `FileElement`–objekt. Glöm inte `parent`! (2p)
- `addPrefix()` som lägger till `"D:"` före namnet på en katalog och alla dess underkataloger och `"F:"` före namnet i katalogens filer och i underkatalogernas filer o.s.v (3p)
- `largestFile()` som ger referens till största filen i katalogen. Alla katalognivåer ska genomsökas. Använd attributet `size`! Returnera `null` om katalogen inte innehåller några filer. Se exempel på nästa sida. (6p)

För resten av klassen `Directory` (instansvariabler, konstruktor m.m) ges (3p).
Att skriva `remove()` eller metoder för utskrift ingår inte i uppgiften.

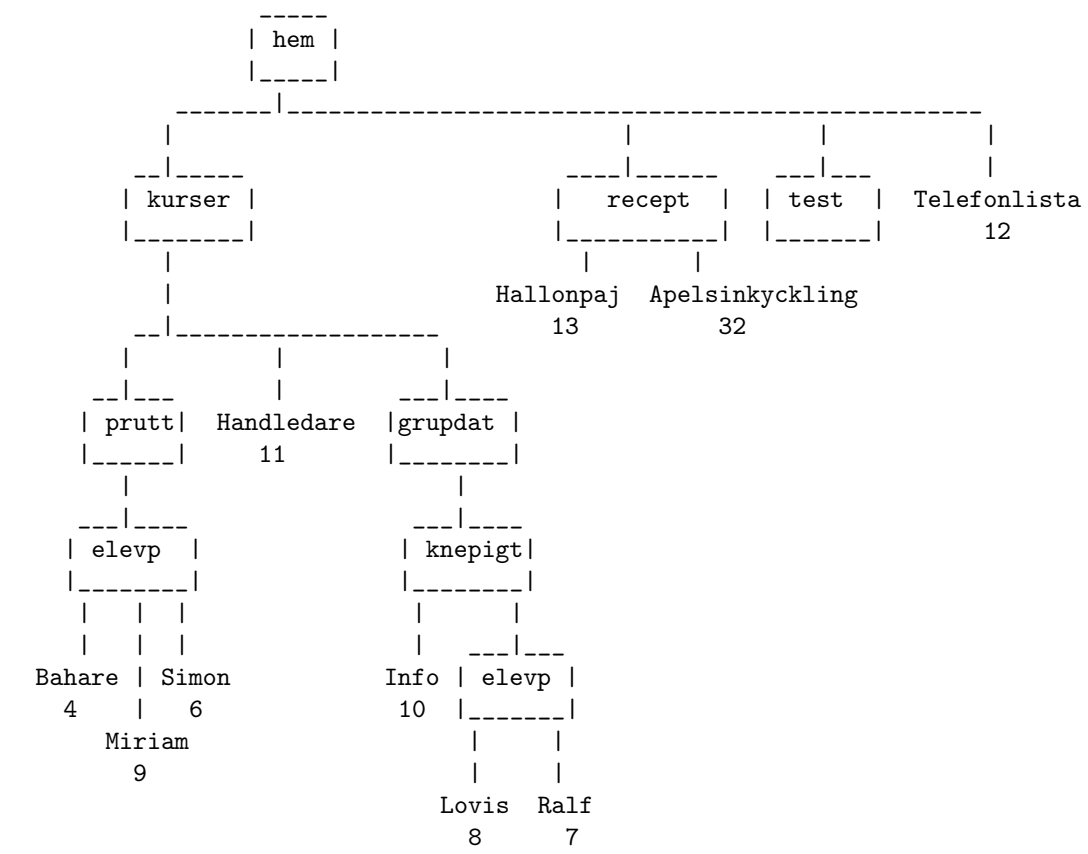
Metoderna `addPrefix()` och `largestFile()` ska implementeras enligt mönstret `Composite`.

För full poäng får operatorn `instanceof` *inte* användas.
Perfekt Java-syntax krävs **INTE** för full poäng på någon uppgift.

Sista delfrågan (8c) går bra att svara på även om du inte svarat på 8a eller 8b.

(5p) **8c.** Antag att objekt av två olika slag, ett grafiskt och ett icke-grafiskt, ska upprätthålla aktuella bilder av hur en filkatalog (`Directory`) ser ut. Dessa objekt ska uppdateras när något läggs till eller tas bort ur filkatalogen. Vilket känt designmönster passar för att implementera detta? Beskriv mönstret och hur det ska tillämpas just här. Ange särskilt vilken roll `Directory` har i mönstret och vilken roll de andra klasserna/objekten har? Visa i UML-diagram hur det kan se ut och beskriv med ord hur mönstrets deltagare kommunicerar. Någon Javakod behöver inte skrivas men det är tillåtet att göra det.

Exempel på objektstruktur i uppgift 8.

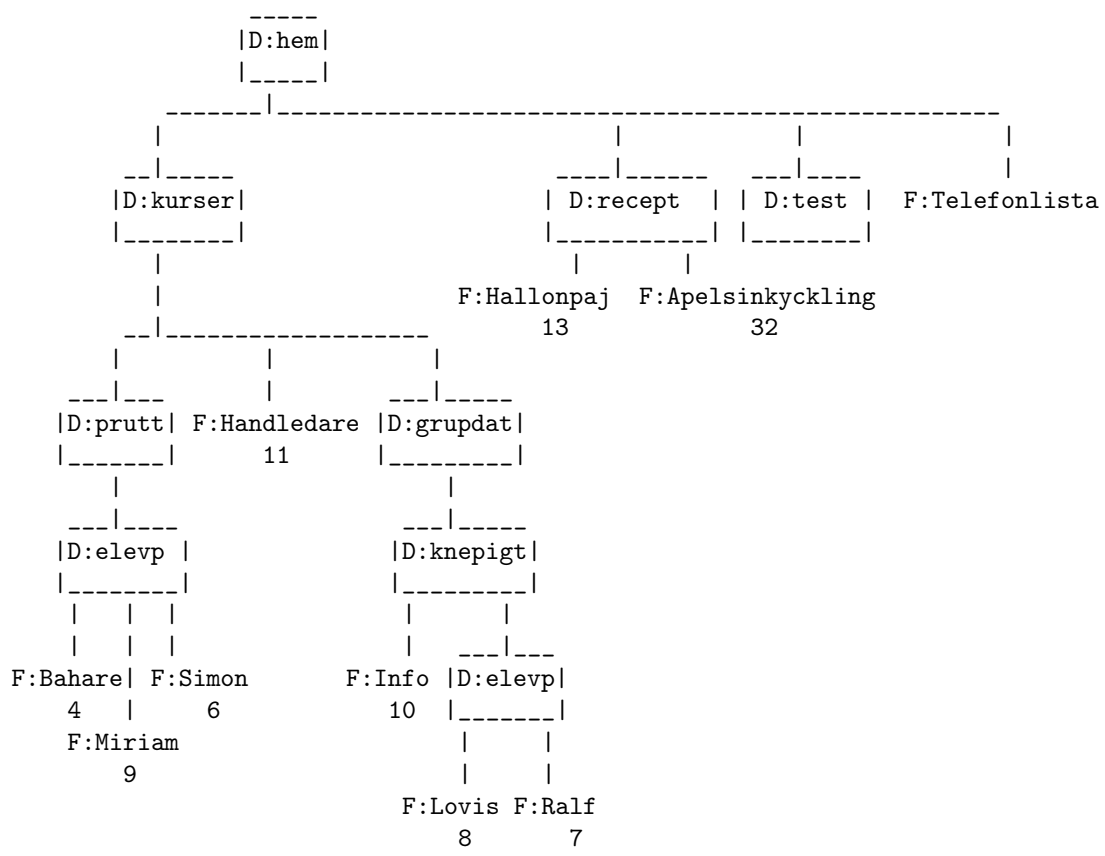


Anrop av `largestFile()` för katalogen `hem` ska ge en referens till `Apelsinkyckling` då det är den största filen, `size = 32`.

Anrop av `largestFile()` för katalogen `grupdat` ska ge referens till `Info` som har storlek 10.

Anrop av `largestFile()` för katalogen `test` ska ge `null` eftersom katalogen är tom.

Exempel på objektstruktur i uppgift 8.



Så här ser det ut efter att metoden `addPrefix()` har anropats på katalogen `hem`.
Alla namn är uppdaterade med `"D:"` respektive `"F:"`.