Tentamen

DD2385 Programutvecklingsteknik vt 2015

Fredagen den 5 juni 2015 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:	23 poäng på del I
Betyg E:	minst 24 poäng på del I
Betyg C:	Godkänd del I och minst 43 poäng på hela tentan
Betyg A:	Godkänd del I och minst 55 poäng på hela tentan

Högst 3 bonuspoäng från labbarna får räknas på del I.
Alla bonuspoäng räknas när betygen C och A sätts.

del I

Skriv gärna flera uppgifter på samma blad på del I!

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

Iterator	**Mediator**	**State**	**Proxy**
Facade	**Template Method**	**Strategy**	**MVC**

A) I en superklass definieras en algoritms huvuddrag. Delar av algoritmen definieras i subklasser eller definieras *om* i subklasser.

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) Man får tillgång till ett antal objekt i sekvens, utan att behöva känna till hur de egentligen är organiserade.

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

(8p) **2.** Rita ett UML-klassdiagram som åskådliggör följande klasser och interface. För full poäng måste öppna och slutna pilspetsar användas rätt samt multiplicitet anges vid de associationer där det är relevant. Fyllda romber behöver *inte* användas. Instansvariabler och instansmetoder behöver *inte* vara med i diagrammet men ev. relationer som ges av variabler och metoder ska ritas ut. *Alla klasser som nämns i koden utom ArrayList ska vara med.* ... står för utelämnad kod.

```
class Center {
    ArrayList<Spec> wlist = new ArrayList<Spec>();

    void tie(Spec w) {wlist.add(w);}

    void untie(Spec w) {wlist.remove(w);}

    void tell() {
        for (Spec w:wlist)
            w.amend();
    }
}

class Focus extends Center {
    ...
    void changestate(int newstate){
        mystate = newstate;
        tell();
    }
    int getstate() {...}
}

public interface Spec {
    public void amend();
}

class Looker implements Spec {
    Focus myCenter;
    ...
    Looker (Focus c){ myCenter = c; }
    public void amend() { ... }
    ...
}

class Wiewer implements Spec {
    Focus myCenter;
    ...
    Wiewer (Focus c) { myCenter = c; }
    public void amend() { ... }
    ...
}
```

(1p) **3.** Klassernas namn i uppgift **2** är valda för det ska antydas men inte vara uppenbart vilket designmöster som används. Vilket mönster är det?

(2p) **4a.** Vad är testdriven programutveckling?

(1p) **4b.** Vad är *refactoring* inom programutveckling?

(2p) **4c.** Testdriven programutveckling och refactoring används inom *XP (eXtreme Programming)*. Ange ytterligare fyra karakteristiska egenskaper/regler hos *XP*

(1p) **4d.** Beskriv utvecklingsmetodiken *Rapid Prototyping*.

5. Antag att följande klasser definieras

```
class Person {
    void activity() {
        System.out.println("REST ");
    }
}

class Athlete extends Person {
    void activity() {
        System.out.println("RUN ");
    }
}
```

(1p) **5a.** Vi skapar ett objekt så här:

```
Person miranda = new Athlete();
```

Vad blir utskriften från `miranda.activity();` ?
Varför blir utskriften så? Poäng ges för motivering, inte för rätt svar.

(1p) **5b.** Antag att klassdefinitionerna ovan gäller. Om en metod börjar så här

```
static void testm (Person p)
```

Vilket av följande påståenden är FEL? Motivering behövs *inte* här.

A) Objekt av typ `Object` kan ges som parameter till `testm()`
B) Objekt av typ `Person` kan ges som parameter till `testm()`
C) Objekt av typ `Athlete` kan ges som parameter till `testm()`

Formuleringen "Objekt av typ A" innebär att objektet skapats med `new A()`

(2p) **6.** Vad är en abstrakt klass i Java? Varför är sådana användbara?

(2p) **7.** I Javas bibliotek finns metoder som sorterar listor av objekt, t.ex.

```
Collections.sort(myList)
```

Sorteringsmetoderna måste kunna jämföra objekten för att sortera dem. Antag att myList innehåller objekt av en klass A som du själv har definierat. Hur ska du i klassen A ange hur objekt ska jämföras?

För att få full poäng måste du visa att du förstår de principer som används i Java. Exakt rätt namn på metoder m.m. krävs inte. Javakod krävs inte heller men kan underlätta att få svaret tydligt.

(4p) **8.** Vilka *fyra* av påståendena nedan är korrekta? De påståenden som rör programmering avser förstås Java. Ge exakt *fyra* svar! Om du ger fler än fyra svar så räknas de fyra första.

A) Att *instansiera* en klass `A` betyder att man skriver en ny klass som ärver från `A`.

B) En klass kan implementera två interface: `class A implements I1, I2 {...}`

C) En klass kan ärva från två klasser
`class A{}  class B{}  class C extends A, B {...}`

D) *Polymorfism* är ett problem i objektorienterad programmering som åtgärdas genom att man gör metoder privata, dvs markerar dem som `private` i koden.

E) Om man vet att `int` – variablerna `m` och `n` båda har värdet `17` så kan man vara säker på att uttrycket `(n == m)` har värdet `true`.

F) Om man vet att `String` – variablerna `p` och `q` båda har värdet `"PI"` så kan man vara säker på att uttrycket `(p == q)` har värdet `true`.

G) Ett *ramverk* i Java innehåller klasser och/eller interface.

H) Alla klasser i Java ärver från klassen `Object` utan att man behöver skriva det explicit.

I) En tråd i Java startas genom att programmeraren anropar trådobjektets metod som heter `run()`.

del II

9.

```
class Particle {
    int x, y;
    Particle (int x, int y) {
        thix.x = x; this.y = y;
    }
}
```

(4p) Utvidga klassen `Particle` så att klassen själv håller reda på hur många objekt av `Particle` som har skapats. Skriv en metod i `Particle` som returnerar antalet skapade objekt. Ange hur metoden ska anropas. Visa alla ändringar och tillägg som behövs i `Particle` för att lösa uppgiften. Antalet skapade objekt är här de som skapas under en körning av ett program som använder klassen. Antalet ska inte lagras på fil utan allt ska skötas inuti klassen `Particle`. Inga andra klasser får användas eller skrivas.

(4p) **10.** Vad är en "fabriksmetod"? Vad är dess uppgift?
Ge något exempel på situation då fabriksmetod används.
Vilken Java–*modifierare* är absolut nödvändig att sätta på en fabriksmetod?
Vad gäller för konstruktorer i samband med fabriksmetoder?

(4p) **11.** Fråga **6** på del I handlar om sortering i Java. Antag att den uppgiften är löst, dvs objekt av klassen `A` sorteras korrekt genom anropet

```
Collections.sort(myList)
```

Nu vill vi m.h.a. en biblioteksmetod få listan sorterad på ett annat sätt än det som är definierat i klassen `A` men *utan att ändra på* `A`. Hur gör man då? Med "annat sätt" avses t.ex. att den i `A` inbyggda sorteringen görs efter personnummer men den nya ska göras på efternamn.

Den i Java inbyggda mekanismen för "andrasortering" använder ett känt designmönster. Vilket?

Svaret behöver inte innehålla korrekta namn ur biblioteket. Det är principerna som är viktiga. Om du inte kan biblioteken tillräckligt bra för att veta hur det görs så använd dina Java-kunskaper för att beskriva hur det *skulle kunna göras*. Det senare kan också ge poäng.

12. 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 de givna klasserna!

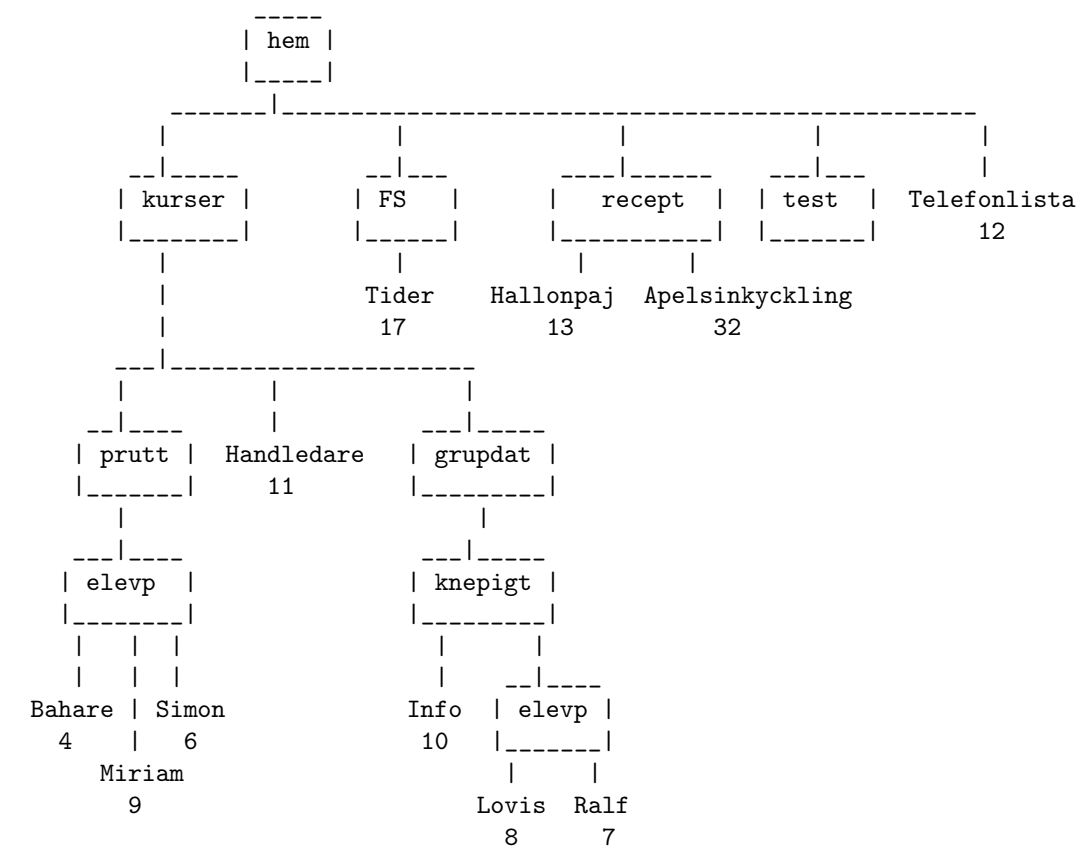
```
abstract class FileElement {

    String name;
    Directory parent = null; // each FileElement except the root must always
                             // have a reference to its parent

    FileElement (String n) {
        name = n;
    }

    public String toString() {
        return name;
    }

    abstract void add(FileElement fe);
    abstract void remove(FileElement fe);
    abstract int size();
  }

class File extends FileElement {
    int size;

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

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

    int size() {
        return size;
    }
}
```

(18p) **12.** Skriv en del av 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.
- `remove(FileElement f)` för att ta bort ett `FileElement`–objekt
- `size()` som beräknar summan av alla filstorlekar i katalogen och alla dess underkataloger enligt mönstret *Composite*.
- `findbigdir()` Metoden ska skriva ut en lista över innehållet i den aktuella katalogen sorterat i storleksordning. Storleken för en katalog är summan av storlekarna för alla filer i katalogen, beräknat på alla nivåer av underkataloger. Se exempel på sidan 8. Det är tillåtet att definiera en hjälpklass här och/eller göra tillägg i de givna klasserna. Du ska dock inte skriva kod för sorteringen själv.
- `moveup()` som flyttar innehållet i katalogen till nivån närmast ovanför samt raderar katalogen. Om det inte finns en nivå ovanför ska metoden inte göra något alls. Se exempel på sidan 9.
- `moveupR()` som flyttar alla filer i katalogen och alla dess underkataloger till nivån ovanför. Katalogen och dess underkataloger ska raderas ur strukturen.

Perfekt Java-syntax krävs **INTE** för full poäng på någon uppgift.

Exempel på objektstruktur i uppgift 12.



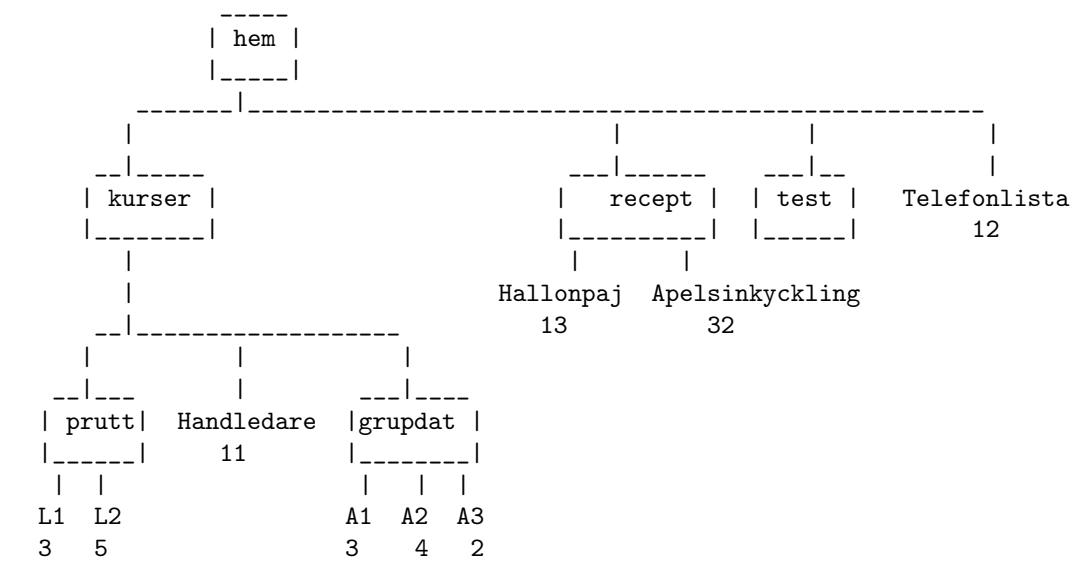
Anrop av `findBigDir()` på katalogen `kurser` ger

```
grupdat     25
prutt       19
Handledare  11
```

För katalogen `hem` blir det

```
kurser       55
recept       45
FS           17
Telefonlista 12
test          0
```

Exempel på objektstruktur i uppgift 12.



Anrop av `moveup()` på katalogen `kurser` i strukturen ovan ska ge en ny struktur enligt bilden nedan.