Tentamen

DD2385 Programutvecklingsteknik vt 2014

Måndagen den 2 juni 2014 kl 10.00 – 13.00

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 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 **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**	**State**	**Proxy**
Facade	**Template Method**	**Strategy**	

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

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

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

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

(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 Doer {
    ArrayList<Wit> wlist = new ArrayList<Wit>();

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

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

    void tell() {
        for (Wit w:wlist)
            w.refresh();
    }
}

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

public interface Wit {
    public void refresh();
}

class Looker implements Wit {
    Head myDoer;
    ...
    Looker (Head d){ myDoer = d; }
    public void refresh() { ... }
    ...
}

class Gazer implements Wit {
    Head myDoer;
    ...
    Gazer (Head d) { myDoer = d; }
    public void refresh() { ... }
    ...
}
```

(1p+2p) **3.** Klassernas namn i uppgift **2** är valda för det ska antydas men inte vara uppenbart vilket designmöster som används. Vilket mönster är det? 1p ges för rätt mönsternamn och 2p för en bra motivering med hänvisning till koden och/eller UML-diagrammet.

(2p) **4a.** Beskriv två egenskaper/regler hos utvecklingsmetodiken *XP (eXtreme Programming)* som skiljer sig mycket från hur man gör inom Vattenfallsmetodiken.

(2p) **4b.** Ange ytterligare två egenskaper/regler hos *XP*

(2p) **5a.** Vad blir utskriften från följande program? Svarsalternativ finns efter klassen `Athlete`.

```
class Question5 {
    public static void main(String[] u) {
        Athlete jenny = new Athlete();
        Person luke = new Person();
        Person ellie = new Athlete();

        jenny.activity();
        luke.activity();
        ellie.activity();
        System.out.println();
    }
}

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

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

A) `REST RUN RUN`
B) `RUN REST RUN`
C) `RUN REST REST`
D) `REST RUN REST`

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

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

Vilket av följande påståenden är FEL?

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()`

6. En klass `Point` för att representera punkter i XY-planet skrivs.

(2p) **a.** Vi vill att två punkter som har samma x-koordinat och samma y-koordinat ska betraktas som *samma* punkt. T.ex. om man skapar två objekt

```
p1 = new Point(7, 11);    p2 = new Point(7, 11);
```

och sätter in båda i en mängd, t.ex. en `HashSet` så ska antalet objekt i mängden vara *ett* efter insättningen av de *två* punktobjekten.

(2p) **b.** Dessutom vill vi kunna sortera punkterna. En lista

`ArrayList<Point> thePoints` ska vid ett anrop

```
Collections.sort(thePoints)
```

sorteras m.a.p. x-koordinaten. Hur ska `Point` skrivas för att anropet ovan ska kunna kompileras och sorteringen utföras korrekt?

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

(3p) **7.** Vilka *tre* av påståendena nedan är korrekta? De påståenden som rör programmering avser förstås Java. Ge exakt *tre* svar! Om du ger fler än tre svar så räknas de tre första.

A) I ett `interface` kan instansvariabler deklareras.

B) `JUnit` är en Java-utvecklingsmiljö anpassad för att lära små barn programmera.

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

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

E) En klass kan ärva från en klass som i sin tur har ärvt från en klass:
`class A{}  class B extends A {}  class C extends B {}`

F) 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`.

G) 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`.

H) Ramverk är ett samlingsnamn för den kategori designmönster som beskriver objektstrukturer.

del II

Skriv gärna svaren på frågorna 8-10 på samma papper men tag nytt för fråga 11!

8. Studera klassen `Mystery` och utskriften från programmet.

```
class Mystery {
    static int spell;

    Mystery(int s) {
        spell = s;
    }

    public String toString() {
        return "" + spell; // "" + is used to enforce the String type,
    }                      // this is not part of the question

    public static void main(String[] u) {
        Mystery m1 = new Mystery(1729);
        Mystery m2 = new Mystery(666);  // *** fråga 8b) ***
        Mystery m3 = new Mystery(42);
        System.out.print(m1 + " ");
        System.out.print(m2 + " ");
        System.out.print(m3);
        System.out.println();
        // System.out.println(spell);   // *** fråga 8c) ***
    }
}

Utskriften blir: 42 42 42
```

(2p) **8a.** Tre objekt av `Mystery` med olika indata skapas men alla skrivs ut som `42`. Förklara detta! Visa hur klassen ska ändras för att utskriften ska bli `1729 666 42` utan att print–satserna ändras.

(1p) **8b.** Om utskrift av objektet `m1` görs vid kommentaren i mainmetoden i koden ovan, vilket värde skrivs då ut?

(1p) **8c.** Om mainmetodens sista print–sats avkommenteras, kommer programmet ovan att kompileras utan fel? Frågan avser programmet som det står på tentan, inte efter ev. ändring enligt **8a**.

(4p) **9.** Vad är polymorfism och vad är dynamisk bindning i objektorienterad programmering? Vilken nytta kan man som programmerare ha av dessa mekanismer?

10. En metod som filtrerar en lista ska skrivas. Från en lista med objekt ska en annan lista där elementen uppfyller ett visst villkor skapas. T.ex.

– Från en lista med heltal, skapa en ny lista med alla udda tal
– Från en lista med personobjekt skapa en ny lista med dem som är i en viss ålder
– Från en lista med ord, skapa en ny lista där alla ord är kortare än ett gränsvärde

(3p) **10a.** Visa hur problemet löses generellt, *inte* för de specifika exemplen!

(3p) **10b.** Beskriv hur man gör för att tillämpa lösningen från **a** på följande specifika fall: Listan innehåller `String` - objekt och filtreringsvillkoret är att textsträngarna ska vara kortare än `n`.

11. *Mönstret Decorator, dekorera en kö*

En kö kan beksrivas med följande `interface`.

```
interface IQueue<E> {
    public boolean isEmpty();
    public void put(E obj);
    public E get();
}
```

Antag att det finns en klass `Queue` som implementerar `IQueue<String>`. Du behöver *inte skriva* `Queue`. En abstrakt `Decorator`–klass och två konkreta klasser som dekorerar kön ska skrivas.

- **Dekorering 1:** Håll reda på antalet element som finns i kön. Antalet ska kunna läsas av med en metod. Tänk på att kön inte behöver vara tom när den dekoreras.
- **Dekorering 2:** Gör det möjligt att
  - Tömma kön
  - Ta bort alla de element ur kön som uppfyller något villkor. Villkoret ska anges som en metodparameter.

Köinterfacet `IQueue<E>` definierar köns basoperationer och får inte ändras. Kön kan bara hanteras genom sina basoperationer.

(4p) **11a.** Välj egna namn på dekoratorerna och rita ett UML-diagram över `IQueue`, `Queue`, den abstrakta `Decorator`–klassen och de två konkreta dekoratorerna. Diagrammet ska förstås följa standardbeskrivningen av *Decorator*.

(4p) **11b.** Skriv den abstrakta *Decorator*–klassen enligt mönstret *Decorator*.

(4p) **11c.** Skriv klassen för konkret Dekorering 1 enligt mönstret *Decorator*.

(4p) **11d.** Skriv klassen för konkret Dekorering 2 enligt mönstret *Decorator*.

- Viktigt för att få full poäng är att mönsteregenskaperna framgår i lösningarna.
- Men, några poäng kan ges även om du inte behärskar mönstret.
- Perfekt Java-syntax krävs inte.
- *Tips:* Köns basoperationer ska fungera på objekt av dekoratorklasserna!