Tentamen

DD2385 Programutvecklingsteknik vt 2013

Onsdagen den 22 maj 2013 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 43 poäng på hela tentan
Betyg A:	Godkänd del I och minst 55 poäng på hela tentan

Bonuspoäng från labbarna (max 5) 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	**Template Method**	**Strategy**	

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

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

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

(7p) **2.** Rita ett UML-klassdiagram som åskådliggör följande klasser. För full poäng måste öppna och slutna pilspetsar användas rätt. För full poäng måste 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 String ska vara med i diagrammet.*

```
abstract class Currer {
    String info;

    Currer (String s) {
        info = s;
    }

    String info () {
        return info;
    }
}

class Acton extends Currer {
    Acton (String s) {
        super(s);
    }
}

class Ellis extends Currer {
    ArrayList<Currer> bell = new ArrayList<Currer>();

    Ellis (String s) {
        super(s);
    }

    void add(Currer c) {
        bell.add(c);
    }

    void remove(Currer c) {
        bell.remove(c);
    }

    String info () {
        String tmp = info;
        for (Currer c : bell) {
            tmp += " " + c.info();
        }
        return tmp;
    }
}
```

(1p) **3a.** Klassernas namn i uppgift **2** är valda för att dölja vilket designmöster som antyds av klasserna. Vilket designmönster är det?

(1p) **3b.** Vad innebär satsen `super(s)` som står i konstruktorn till två av klasserna?

4. *Uppkopplad förbindelse mellan program i Java*

(5p) Ett program som tillåter att andra program kopplar upp sig till det kallas **A** – program. **A** – programmet lyssnar på en viss **B** på den dator där det kör. Om flera program ska kunna vara uppkopplade samtidigt så skapar **A** – programmet en **C** för varje anslutare. I varje ände av en uppkopplad förbindelse finns en **D**. En **D** har två **E**–objekt och genom (eller på) dem kan information sändas respektive tas emot.

Var och en av **A, B, C, D, E** i texten ovan ska ersättas med ett av orden här nedanför. Välj rätt ord för varje bokstav!

tråd	**exception**	**ip-adress**	**klient**
socket	**konsument**	**observer**	**port**
runnable	**server**	**subklass**	**ström**
superklass	**actionListener**	**unicode**	

(1p) **5a.** Vad är *testdriven programutveckling*? Svara kort!

(1p) **5b.** Hur används testning inom *eXtreme Programming*?
Svara kortfattat på båda frågorna. 1-2 meningar per fråga kan räcka.

(1p) **6.** Vad är *Refactoring* inom programutveckling? Svara med högst tre meningar!

(2p) **7.** Beskriv utvecklingsmetodiken *Rapid Prototyping*.

(2p) **8.** Antag att följande deklarationer är gjorda (och är fullständiga)

```
interface I {...}
class A {...}
class B extends A implements I {...}
```

Tre metoder deklareras i klassen Test

```
static void X(I x) {...}
static void Y(A y) {...}
static void Z(B z) {...}
```

Här följer fyra påståenden om aktuella parametrar till metoderna. Ange SANT eller FALSKT för varje påstående

A) Objekt av `A` kan ges som aktuell parameter till `X`
B) Objekt av `B` kan ges som aktuell parameter till `X`
C) Objekt av `B` kan ges som aktuell parameter till `Y`
D) Objekt av `A` kan ges som aktuell parameter till `Z`

(2p) **8.** Vad är en fabriksmetod? Ange ett exempel på situation då det passar att använda en fabriksmetod.

(2p) **9.** Beskriv *kort* (2-3 meningar bör räcka) syftet med mönstret *Observer*. Ange någon tillämpning (t.ex ur Java-API:n eller från kursens föreläsningar) där mönstret är lämpligt att använda. UML behövs **inte**.

del II

Skriv gärna alla svar på frågorna 10-12 på samma blad men tag nytt blad för fråga 13!

10. Klassen `Treasure` nedan är tänkt att följa mönstret Singleton.

```
public class Treasure {

    private static Treasure theTreasure = new Treasure();

    Treasure () {}

    public Treasure getTreasure() {
        return theTreasure;
    }
}
```

(6p) **10a.** `Treasure` innehåller två fel. För varje fel, rätta felet och förklara *tydligt* varför det inte kan bli en `Singleton` så som klassen är skriven och varför det fungerar efter rättelsen.

(2p) **10b.** I Java finns det ett enklare och säkrare sätt att göra Singletonobjekt än genom att definiera en klass enligt ovan (den rättade). Hur gör man? Javakod behövs inte i svaret.

(2p) **11.** Vilket av föjande påståenden stämmer för en instansmetod i klassen A vars deklaration inleds med `synchronized void syncm()`? Antag att `a` refererar till ett objekt av klassen `A`.

A) Under exekvering av `a.syncm()` kan inga andra metoder i klassen A märkta synchronized anropas.

B) Under exekvering av `a.syncm()` sker en synkroniserad skräpsamling.

C) Under exekvering av `a.syncm()` är objektet som a refererar till låst så att andra trådar inte kan anropa metoder märkta synchronized i objektet a.

D) Under exekvering av `a.syncm()` är objektet som a refererar till låst så att andra trådar inte kan anropa metoden `syncm()` i a. Ev. andra synchronized-metoder får anropas under låsningen.

(4p) **12.** Vad är polymorfism i objektorienterad programmering?
Vad är dynamisk bindning? Ett exempel och UML kan hjälpa till att göra svaret tydligare men krävs inte.

13. *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:** Gör det möjligt att sätta in en lista av objekt i kön och ta ut ett antal objekt på en gång ur kön genom metoderna

  ```
  void putSequence (List<String> list)
  List<String> getSequence(int n)
  ```

- **Dekorering 2:** Håll reda på hur lång tid som gått sedan senaste `get()` eller `put()` gjordes på kön. Tiden ska kunna läsas av genom en metod och ska skrivas ut i samband med att `get()` respektive `put()` utförs. Metoden
  `System.currentTimeMillis()`
  ger antalet millisekunder sedan midnatt 1 januari 1970.

(4p) **13b.** 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) **13b.** Skriv den abstrakta *Decorator*–klassen enligt mönstret *Decorator*.

(3p) **13c.** Skriv klassen för konkret Dekorering 1 enligt mönstret *Decorator*.

(5p) **13d.** Skriv klassen för konkret Dekorering 2 enligt mönstret *Decorator*.

• Det viktiga för att få full poäng på **13b-d** är att mönsteregenskaperna framgår i lösningarna.

• Java-syntaxen behöver inte alls vara perfekt.

• *Tips:* Tänk på att basoperationerna ska fungera på objekt av dekoratorklasserna!