Tentamen

DD2385 Programutvecklingsteknik vt 2016

Tisdag 7 juni 2016 kl 14.00-17.00

Tillåtna hjälpmedel:
Penna, Suddgummi och linjal

Maximala betygsgränser:
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

Lycka till,

Ann och Vahid

Del I

1. (5p) 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 State Proxy Mediator Decorator Template Method Strategy Observer

   a. I en superklass definieras en algoritms huvuddrag. Delar av algoritmen definieras om i subklasser.

   b. Ett objekts beteende ändras när dess tillstånd ändras.

   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. När tillståndet i ett objekt ändras så uppdateras ett antal andra objekt som prenumererar på uppdateringsinfo.

2. (8p) 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.

```
interface Spectator{
    public void alert();
}

class MeasureDevice{
    ArrayList<Spectator> spectators=new ArrayList<Spectator>();
    public void addSpectator(Spectator s){spectators.add(s);}
    public void removeSpectator(Spectator s){...}
    public void alertSpectators(){
        for (Spectator s: spectators)
            s.alert();
    }
    ...
}

class Thermometer extends MeasureDevice{
    double temp;

    public void setTempAndAlert(double newTemp){
```

```
        temp=newTemp;
        alertSpectators();
    }

    public double getTemp(){...}
    ...
}

class Speaker implements Spectator{
    Thermometer t;
    String lastTemp;
    Speaker(Thermometer t){this.t=t;}
    private String convertDigit2Word(double temp){...}
    public void alert(){lastTemp = convertDigit2Word(t.getTemp());}
    ...
}

class DigitalDisplay implements Spectator{
    Thermometer t;
    double lastTemp;
    DigitalDisplay(Thermometer t){this.t=t;}
    public void alert(){lastTemp = t.getTemp();}
    ...
}
```

(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?

4. Betrakta följande kod:

```
class Car{
    static int amount;
    String mark;

    Car(String m){
       mark=m;
       amount++;
    }


    public static void main(String[] args){
         Car c1 = new Car("BMW");
         Car c2 = new Car("Volvo");
         Car[] list1=new Car[]{c1};
         Car[] list2=new Car[]{c1};
         list1[0].mark = "Tesla";
         System.out.println(list1[0]+" "+list2[0]);
    }
```

```
    public String toString(){
         return mark+":"+amount;
    }
}
```

(2p) a. Vad skrivs ut av programmet?

(2p) b. Vad är det för skillnad mellan variablerna `mark` och `amount`?

(2p) 5. a. Vilka skillnader finns mellan abstrakta klasser och interface?

(2p) b. Vad är enum i java? Ge ett exempel.

6. Antag att följande klasser definieras

```
class Person {
   String name;
   Person(){System.out.println("UNNAMED");}
   Person(String name){
       this.name=name;
       System.out.println("PERSON");
   }

   private void activity(){ System.out.println("REST");}
}

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

class Main{
   public static void main(String[] args) {
       Person miranda = new Athlete("Miranda");
       miranda.activity();
   }
}
```

a. `Main` ger kompileringsfel. Vilka 2 saker ska ändras i klasserna `Person` och `Athlete` så att klassen `Main` kan kompileras utan fel. Koden i `Main`

(2p) får inte modifieras.

(2p) b. Åtgärda felen i klasserna `Person` och `Athlete` (koden i `Main` får inte modifieras) så att man får följande utskrift när man kör programmet. Du får inte lägga till eller modifiera några utskriftsatser.

```
PERSON
RUN
```

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

   a. En metod med modifieraren `protected` ärvs inte.

   b. Konstrukturer ärvs inte.

   c. Klassen A kan implementera klass B endast om klassen B är en abstrakt klass.

   ```
   class A implements B{...}
   abstract class B{...}
   ```

   d. `interface X extends Y{}` är korrekt om `Y` är interface.

   e. `class X extends Y{}` är korrekt om `Y` är interface.

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

   g. Anta att `p` och `q` är referensvariabler och att `(p == q)` är `false`. Påstående: Detta innebär alltid att `p.equals(q)` också är `false`.

   h. En fil som innehåller klassdefinitionen `public class A{}` måste kallas `A.java` för att den ska kunna kompilera.

   i. Ett ramverk i Java innehåller INTE några interface.

   j. Om man vill kunna köra ett program med kommandot `java A` då måste klassen `A` innehålla en metod som heter `main` definerad med signaturen `public static void` och har ett argument av typen `String[]`.

Del II

8. En tråd kan ha något av tillstånden: under körning, färdig för körning, blockerad och död.

   a. (2p) Man får ett fel av typen `IllegalThreadStateException` när man anropar metoden `start` hos en tråd som befinner sig i död-tillstånd. Vad är anledningen till detta? Vilka möjliga tillstånd kan en tråd som just befinner sig död-tillstånd byta till?

   b. (2p) En tråd har blockerande tillstånd p.g.a. att tråden väntar på data som ska matas in via tangentbordet. Vilket tillstånd är trådens nästa tillstånd direkt efter att en godtycklig data har matats in via tangentbordet?

   c. (2p) En tråd befinner sig i tillståndet död. Vad händer när man anropar objektets run-metod?

9. (4p) Beskriv LSP (Liskov Substitution Principle) genom ett exempel.

10. (5p) Betrakta följande kod:

```
class CentralFuseBox{
    CentralFuseBox theOnlyOne;
    CentralFuseBox(){...}

    CentralFuseBox getInstance(){
        if (theOnlyOne==null)
               theOnlyOne=new CentralFuseBox();
        return theOnlyOne;
    }
}
```

Klassdefinition `CentralFuseBox` kompilerar utan några fel men är ett misslyckat försök till designmönstret Singleton. Vilka krav ställs på en Singletonklass? Utför minimala modifieringar i koden så att `CentralFuseBox` svarar mot de krav som ställs på Singleton. Koden behöver inte vara trådsäker.

11. Inför resan när man packar in sina saker i en resväska är det praktiskt att gruppera saker av samma slag i mindre väskor och förpackningspåsar så att man snabbare finner det man söker i resväskan när man väl behöver det.

(15p) Väskor kan innehålla saker och andra mindre väskor som i sin tur kan innehålla saker och väskor o.s.v. enligt mönstret *Composite*. Saker representeras av objekt av klassen `Thing` som finns nedan. Väskor representeras av objekt av klassen `Bag`. Både saker och väskor har namn och behållare som lagras i instansvariabelerna `name` respektive `container` i den gemensamma superklassen `PackItem`. Instansvariabeln `container` ska ange i vilken behållare (`Bag`) denna `PackItem` befinner sig. `container` ska vara `null` om objektet inte ligger i en `Bag`.

```
abstract class PackItem {
    String name;
    Bag container;  //bag som innehåller denna PackItem.

    PackItem (String n) {
        name = n;
    }
    ...
}

class Thing extends PackItem {

    Thing(String n) {
        super(n);
    }
    ...
}
```

Skriv klassen `Bag` och komplettera klasserna `PackItem` och `Thing` enligt mönstret *Composite*. Ett objekt av `Bag` ska kunna innehålla objekt av `Bag` och `Thing`. Följande metoder ska du definiera som abstrakta eller/och konkreta beroende på vilken klass de placeras enligt mönstret *Composite*.

- `add()` för att lägga till ett objekt av `PackItem`.
- `remove()` för att ta bort ett objekt av `PackItem`.
- `print()` som skriver ut resväskans innehåll med indenteringar som anger packnivån. Se exempel på sidan 7. Metoden ska skrivas enligt mönstret *Composite*.
- `putInBag()` som lägger varje `Thing` i en extra `Bag` (kallad Safetybag), överallt där flera objekt av klassen `Thing` ligger bredvid ett eller flera `PackItem` (`Bag` eller `Thing`) i samma `Bag`. Se exempel på sidan 7. Metoden ska skrivas enligt mönstret *Composite*.

Enda tillåtna ändringen av klassen `PackItem` är att lägga till abstrakta metoder. Du får självklart skriva fler hjälpmetoder i klassen `Bag` än de som krävs i uppgiften.

`x instanceof A` ger true om objektet som `x` refererar till har typen `A`.

`instanceof` kan behövas för att lösa uppgifterna men ska användas med stor försiktighet. Överanvändning av `instanceof` kan medföra poängavdrag.

EXEMPEL

Before putting safeteybags

```
Suitcase
  Jeans
  Socks
  Toiletbag
    Shampoo
    Smallbag
      Medicine
  Emptybag
  Plasticbag
    comb
```

After putting safeteybags

```
Suitcase
  Safetybag
    Jeans
  Safetybag
    Socks
  Toiletbag
    Safetybag
      Shampoo
    Smallbag
      Medicine
  Emptybag
  Plasticbag
    comb
```

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