CHALMERS

Institutionen för data- och informationsteknik

TENTAMEN

KURSNAMN	**Objektorienterad programmering, 7.5p**
PROGRAM:	**TKIEK-2, TKTFY-3, TKTEM-3** **2017/2018, lp 2**
KURSBETECKNING	**TDA550**
EXAMINATOR	**Uno Holmer**
TID FÖR TENTAMEN	**Onsdagen den 22/8 2018, 08.30-12.30**
HJÄLPMEDEL	**Java API** (delas ut av skrivningsvakten)
ANSV LÄRARE	**Uno Holmer** **tel. 772 5730** **besöker tentamen ca kl. 9.30 samt ca 11.30**
DATUM FÖR RESULTAT	**Senast den 12/9 2018** **Granskningsdatum meddelas på kursens hemsida**
ÖVRIG INFORM.	**Betygsgränser: 3 - 24p, 4 - 36p, 5 - 48p. (max 60p)**

TENTAMEN: Objektorienterad programutveckling, fk

Läs detta!

- *Uppgifterna är inte ordnade efter svårighetsgrad.*
- Börja varje hel uppgift på ett nytt blad. Skriv inte i tesen.
- Skriv **inte** med rödpenna.
- Ordna bladen i uppgiftsordning.
- Skriv din tentamenskod på varje blad (så att vi inte slarvar bort dem).
- **Skriv rent dina svar. Oläsliga svar r ä t t a s e j!**
- Programkod som finns i tentamenstesen behöver ej upprepas.
- Programkod skall skrivas i Java 5, eller senare version, och vara indenterad och renskriven.
- Onödigt komplicerade lösningar ger poängavdrag.
- Givna deklarationer, parameterlistor etc. får ej ändras.
- Läs igenom tentamenstesen och förbered ev. frågor.

I en uppgift som består av flera delar får du använda dig av funktioner klasser etc. från tidigare deluppgifter, även om du inte löst dessa.

Lycka till!

Uppgift 1

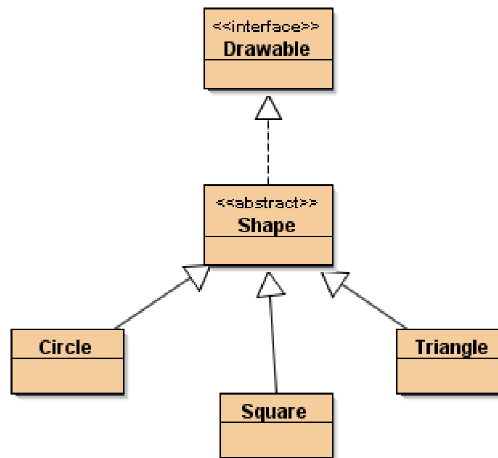
Klassen nedan strider mot *Open-Closed Principle*. Refaktorera klassen enligt designmönstret *Strategy*.

```
public class PersonalTrainer {
    private int trainee;
    public PersonalTrainer(int trainee) {
        this.trainee = trainee;
    }
    public void activate() {
        switch (trainee) {
            case 1: walk();
            break;
            case 2: run();
            break;
            case 3: swim();
        }
    }
    private void walk() {
        System.out.println("Walking");
    }
    private void run() {
        System.out.println("Running");
    }
    private void swim(){
        System.out.println("Swimming");
    }
}
```

(7 p)

Uppgift 2

Antag att vi har en klasshierarki för ritning av geometriska figurer:



Gränssnittet `Drawable` definieras

```
public interface Drawable {
    void move(int deltaX,int deltaY);
    void moveTo(int x,int y);
    void draw();
    void erase();
}
```

Vi skall lägga till en möjlighet att animera figurer. När en figur animeras skall den ”åka” i en viss riktning med en given hastighet. En sådan rörelse kan simuleras genom att upprepade gånger rita om figuren i olika positioner efter den tänkta linjen.

Definiera klassen `Animation` genom att tillämpa designmönstret *Decorator*. Man skall t.ex. kunna skriva så här:

```
Square sq = new Square(50,50,”blue”,100);
sq.draw();
Animation a = new Animation(new Circle(150,150,”red”,50));
a.animate(3,1,40,20);
```

Metoden `animate` skall ha fyra heltalsparametrar. De två första anger förflyttningen, mätt i bildpunkter, i x- resp. y-led i varje animationssteg. Den tredje anger antalet omritningar som skall ske. Den fjärde anger antalet omritningar per sekund. I exemplet ovan skall den röda cirkeln åka snett ned åt höger genom att rita den 41 gånger med tidsfördröjningen 50 ms mellan varje bild. Du kan utgå från att ingen av subklasserna ovan ritar om figuren när den flyttas med `move` eller `moveTo`. *Tips:* En paus i exekveringen i *t* ms kan fås med anropet `Thread.sleep(t);`

(8 p)

Uppgift 3

Betrakta nedanstående gränssnitt och klasser:

```
public interface IA {
   public void doA();
}
```

```
public interface IX {
   public void doX();
}
```

```
abstract public class A
implements IA {
   public void doA() {
      System.out.println("A doA");
   }
   abstract public void doC();
}
```

```
public class B extends A {
   public void doC() {
      System.out.println("B doC");
   }
}
```

```
public class C extends B
implements IX {
   public void doA() {
      System.out.println("C doA");
   }
   public void doX() {
      System.out.println("C doX");
   }
   public void doC() {
      System.out.println("C doC");
   }
}
```

```
public class X implements IX {
   public void doX() {
      System.out.println("X doX");
   }
   public void doA() {
      System.out.println("X doA");
   }
}
```

a) Rita ett UML-diagram över klasserna och gränssnitten.

(1 p)

b) Vad blir resultatet för vart och ett av följande kodavsnitt (ger kompileringsfel, ger exekveringsfel, skriver ut xxx, etc.)?

i) B b = **new** B(); b.doA();	ii) IA a = **new** X(); a.doA();	iii) C c = **new** B(); c.doC();	iv) B b = **new** C(); b.doX();

v) IX x = **new** C();
X x1 = (X) x;
x1.doA();
	vi) IX x = **new** C();
	B b = (B) x;
	b.doC();
		vii) A a = **new** B();
		a.doA();
			viii) IA a = **new** A();
			a.doA();

(8 p)

Uppgift 4

CSV-filer (colon separated values) är textfiler med fält separerade av t.ex. ':'. De används ofta för att spara konfigurationsdata för program, mätvärden m.m. Fördelen är att de lätt kan redigeras i en vanlig texteditor, importeras till kalkylblad, osv.
Ex. En CSV-fil med numeriska mätvärden för olika givare skulle kunna se ut så här

```
sensor1:1.0:20.7:4.3
sensor5:7.1:9.4
sensor1:42.3:2.8:3.9:4.1
```

Första kolumnen i varje rad är givarens identifikation, därefter kommer mätvärdena, separerade av ':'. Data för en givare kan finnas på flera olika rader i filen.

a) Ange lämpliga samlingsklasser i Javas API för att lagra information enligt ovan. Sensornamnen lagras som strängar och värdena med typen `Double`.

(1 p)

b) Konstruera en metod som läser en CSV-fil enligt ovan och sparar innehållet i en lämplig datasamling enligt a och som ges tillbaka som returvärde:

```
publicn typ loadCSVData(String fileName)
```

(6 p)

Uppgift 5

Betrakta nedanstående fyra specifikationer för metoden

```
double sqrt(double x)
```

som returnerar kvadratroten för argumentet `x`.

I) @requires x >= 0
@return y such that |y*y – x| <= 0.0001

II) @requires x >= 0
@return y such that |y*y – x| <= 0.01

III) @return y such that |y*y – x| <= 0.0001
@throws IllegalArgumentException if x < 0

IV @requires x > 0
@return y such that |y*y – x| <= 0.0001

Ange för vart och ett av nedanstående par av specifikationer, vilken av specifikationerna som är starkast. Om det inte går att avgöra vilken specifikation som är starkast skall detta anges. Motiveringar krävs!

a) I och II
b) I och III
c) II och III
d) II och IV

(4 p)

Uppgift 6

En klassisk fransk kortlek har fyra färger (suit) och 13 valörer (rank) av varje kort. Följande gränssnitt beskriver ett minimum av metoder:

```
public interface FrenchCard {
    enum Suit {HEARTS,SPADES,CLUBS,DIAMONDS}
    int getRank();
    Suit getSuit();
}
```

Vidare finns följande klass, som vi inte beskriver närmare:

```
public class Card implements FrenchCard,Cloneable,Comparable<Card>
```

Uppgiften är nu att konstruera två metoder (se nedan) i klassen `CardHand` för att representera en *hand* av kort. En hand är en samling kort ur en eller flera kortlekar. Klassen har bl.a. följande metoder:

`public void add(Card c)`	adderar kortet `c` till handen
`public int size()`	returnerar antalet kort i handen
`public Card look(int i)` `throws IndexOutOfBoundsException`	returnerar det i:te kortet i handen
`public boolean discard(int i)` `throws IndexOutOfBoundsException`	tar bort det i:te korten ur handen

Dessutom skall klassen ha metoderna `clone` som returnerar en djup kopia av handen, samt `equals` som avgör om två händer är lika, samt `hashCode`. Två händer är lika om de innehåller lika många kort, samt lika många av varje kort. Den interna ordningen bland korten spelar ingen roll för om två händer är lika eller ej.

Ex.

```
Card c1 = new Card(4,FrenchCard.Suit.DIAMONDS);
Card c2 = new Card(12,FrenchCard.Suit.SPADES);
Card c3 = new Card(9,FrenchCard.Suit.HEARTS);
CardHand h1 = new CardHand();
h1.add(c1); h1.add(c2); h1.add(c3);
CardHand h2 = new CardHand();
h2.add(c3); h2.add(c1); h2.add(c2);
System.out.println(h1.equals(h2));   // true
System.out.println(h2.equals(h1));   // true
System.out.println(h1.hashCode() == h2.hashCode());    // true
```

Du kan anta att handen är representerad som en lista av kort. Implementera metoderna `clone` och `equals` och `hashCode` i klassen `CardHand`. Ingen av metoderna får ändra kortens inbördes ordning. *Tips*: Utnyttja `clone` och `Collections.sort` i `equals`.

(9 p)

Uppgift 7

a) Nedanstående kod bryter mot minst tre designprinciper som behandlats i kursen. Nämn dessa och motivera varför koden bryter mot dem!

(3 p)

```
public class AirCondition {
    public static final double COMFORT_TEMPERATURE = 22.0;
    private double temperature;
    private Cooler cooler = new Cooler();
    public double getTemperature() { return temperature; }
    public Cooler getCooler() { return cooler; }
    // Other members omitted.
}
```

```
public class Cooler {
    public void on() {…}
    public void off() {…}
}
```

```
public class ClimateControl {
    private AirCondition airCondition;
    public ClimateControl(AirCondition airCondition) {
        this.airCondition = airCondition;
    }
    public void adjustTemperature() {
        double delta = airCondition.getTemperature() -
                       AirCondition.COMFORT_TEMPERATURE;
        if ( delta > 1.0 )
            decreaseTemperature();
    }
    private void decreaseTemperature() {
        airCondition.getCooler().on();
        while ( airCondition.getTemperature() >
                AirCondition.COMFORT_TEMPERATURE )
        {
            sleep(100); // ms
        }
        airCondition.getCooler().off();
    }
    // Other members omitted.
}
```

b) Refaktorera klasserna `Aircondition` och `ClimateControl` så att designprinciperna i a följs.

(5 p)

Uppgift 8 GUI, Observer

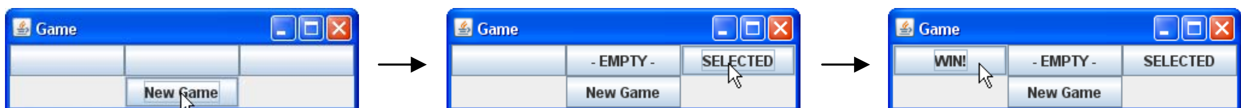
I ett klassiskt publiknummer får en person chansen att erhålla en vinst som är gömd i en av tre ogenomskinliga lådor. De två andra är tomma. Det går till så här. Först gissar man på en låda, men man får inte se innehållet. Därefter visar programledaren upp en tom låda – det finns ju minst en till. Man får nu chansen att byta till den tredje lådan, eller att stanna kvar vid sitt första val. Det visar sig att ett av valen i längden ger betydligt bättre utdelning än det andra. Den som läst lite sannolikhetsteori kan räkna ut vilket, men det skall vi inte göra här. Uppgiften blir i stället att konstruera ett grafiskt användargränssnitt till ett program som simulerar spelet. Logikdelen är redan klar och implementeras av klasserna `Box` och `GameEngine`. Designmönstret Observer utnyttjas för att de båda klasserna inte skall behöva veta något om det grafiska gränssnittet. `Box` håller reda på tillståndet i en låda i spelet, d.v.s. om lådan är stängd, vald, öppen, tom, full, etc. Vi går inte närmare in på denna klass eftersom objekten helt sköts av `GameEngine`.

```
public class Box extends Observable { ... }

public class GameEngine {
    public GameEngine() { ... }
    public void newGame() { ... }
    public void select(int i) { ... }
    public void addBoxObserver(int i,Observer obs) { ... }
}
```

När metoden `select` anropas med ett värde mellan 0 och 2 uppdaterar `GameEngine` tillståndet i motsvarande låda. Även en annan låda kan påverkas. Om det var första anropet av `select` skall t.ex. en annan tom stängd låda öppnas. Metoden `addBoxObserver` adderar en observatör till angiven låda; i detta fallet en GUI-komponent av typen `BoxView` som visar lådan i ett fönster.

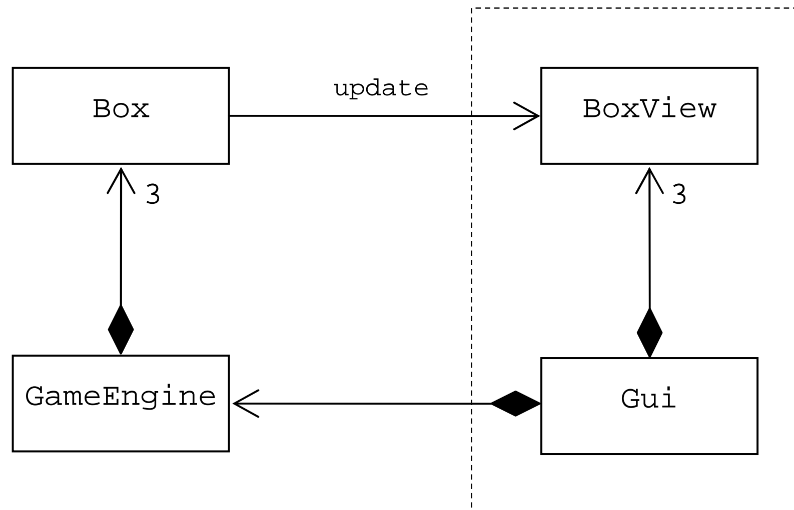
Gränssnittet skall ha fyra knappar. De tre översta motsvarar lådorna. Den nedre skall starta ett nytt spel. Alla lådknappar är från början blanka. I en lådknapp kan tre olika texter visas: SELECTED, EMPTY, eller WIN! Så här kan två spelomgångar se ut. I den första väljer spelaren den högra lådan genom att klicka på den varefter den mittersta visas upp. Spelaren väljer nu i sitt andra drag den vänstra lådan och vinner.



I denna omgång väljer spelaren att behålla sitt val genom att klicka på mittknappen en gång till. Tyvärr visar den sig vara tom.

v.g.v.

Uppgiften går ut på att implementera klasserna i den streckade rutan:



a) Implementera klassen `BoxView` som representerar en låda visuellt i Gui:t. Klassen skall ärva från `JButton` och vara observatör på `Box`. När klassens `update`-metod anropas kommer den andra parametern att innehålla en sträng med följande format: `s;b` där s är tillståndet `SELECTED` eller `OPEN`, medan b anger om lådan är tom eller inte. Ex. på sådana strängar är `"SELECTED;false"`, `"SELECTED;true"`, `"OPEN;true"`, `"OPEN;false"`. I de två första fallen skall texten ”`SELECTED`” visas på knappen, i det tredje ”`WIN!`”, och i det sista ”`- EMPTY -`”.

(3 p)

b) Implementera klassen `Gui` som visar ett fönster enligt exemplen ovan. Klassen skall skapa ett objekt av klassen `GameEngine` samt tre `BoxView`-objekt. Dessa skall visas i fönstret tillsammans med en knapp med texten ”`New Game`”. `BoxView`-objekten skall registreras som observatörer hos spelmotorn med lämplig metod. Vid knapptryck skall `select` anropas i spelmotorn med knappens position som parameter.

(5 p)