Data och Informationsteknik / Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
Magnus Myréen

Modelsvar för
Tentamen för Objektorienterad programvaruutveckling, TDA545

Fredag, 2014-10-31, 08:30-12:30, byggnad M

Ansvarig lärare:	Magnus Myréen, besöker byggnad M kl 09:00 och 11:30
Vitsords gränser:	3=28p, 4=38p, 5=48p, max 60p.
Hjälpmedel:	på sista sidan av dokument finns ett utdrag ur Javas API
Resultat:	skickas per epost från Ladok
Lösningar:	kommer att finnas på kurshemsidan
Granskning av rättning:	tider för detta kommer att finnas på kurshemsidan

---

Kom ihåg att inte fastna på en uppgift. Bestäm i förväg din egen tidsgräns per uppgift. ***Lycka till!***

Uppgift 1: [4 poäng totalt] *while-loopar, boolska uttryck, primitiva typer*

```
int x = 0;
int y = 1;
while (0 < y) {
    System.out.println(x);
    System.out.println(y);
    x = x + y;
    y = x + y;
}
System.out.println("slut");
```

a) Skriv ner de första 8 raderna av programmets utskrift. [2 poäng]

De första 8 raderna av programmets utskrift är:

```
0
1
1
2
3
5
8
13
```

Poäng: man mister inte poäng för små fel i additionerna här, dvs siffrorna behöver inte vara exakt rätt.

b) Skriver programmet någonsin `slut`? *Förklara ditt svar.* [2 poäng]

Ja, värdet av x och y ökar tills det blir för stort för `int` typen. Då blir värdet på y (och x) negativt och while loopen slutar, och `"slut"` skrivs ut.

Poäng: man får 1 poäng för svaret "ja", och 1 poäng till om man förklarar orsaken klart och tydligt.

Uppgift 2: [4 poäng totalt] *referensvärden, parameteröverföring, metodanrop*

Skriv ner vad programmet skriver ut när man kör `java A`. [4 poäng]

```
public class A {
    public int n = 0;
    public static A fun(A t, A u) {
        u.n = u.n + 1;
        t = new A();
        t.n = 4;
        return u;
    }
    public static void main(String[] args) {
        A x = new A();
        A y = new A();
        x.n = 10;
        A z = fun(x,y);
        z.n = y.n + 2;
        System.out.println("x.n = " + x.n);
        System.out.println("y.n = " + y.n);
        System.out.println("z.n = " + z.n);
    }
}
```

Tips: Var noggrann! Bra att rita "minnesplatser" och "pilar" som vi gjorde på föreläsningarna.

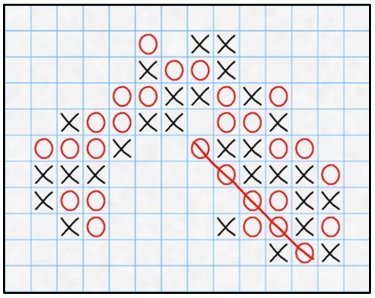
Följande skrivs följande ut när programmet `java A` körs.

```
x.n = 10
y.n = 3
z.n = 3
```

Poäng: man får 1 poäng om endast en av raderna är rätt *och det inte finns förklaringar hur man kom fram till resultatet.* Ifall svaret visar att studenten förstår hur metodanropet kopierar referensvärden så blir det 1 eller 2 poäng till. För fulla 4 poäng måste man ha alla tre rader av utskriften rätt.

Uppgift 3: [18 poäng totalt] *fält, for-loopar, exceptions*

Här implementerar vi en klass som representerar ett luffarschack spel (ibland kallas det fem-i-rad).



Luffarschack spelas på ett plan av rutor som antingen kan vara tomma, eller ha ett `X` eller ett `O` ritat i sig. Spelet tar slut när det finns fem celler med samma markering (`X` eller `O`) i rad, antingen lodrätt, vågrätt eller diagonalt.

Nedan finns det en skiss av klassen `Luffarschack`. Uppgiften är att fylla i koden som saknas, enligt punkterna (a), (b), (c) nedanför koden.

```
public class Luffarschack {

    public enum Cell { X, O, EMPTY };

    int width;
    int height;
    Cell[][] board;
    boolean xTurnNext;

    public Luffarschack(int width, int height) {
        ... svar till del (a) här ...
    }

    public void choose(int i, int j) throws IllegalArgumentException {
        ... svar till del (b) här ...
    }

    public boolean hasFiveInARow() {
        ... svar till del (c) här ...
    }

    ... eventuella hjälpmetoder här för del (a), (b), (c) ...
}
```

a) Skriv koden för konstruktorn. Den bör skapa ett tomt spel plan. [3 poäng]

```
this.width = width;
this.height = height;
board = new Cell[width][height];
for(int j=0; j<height; j++) {
    for(int i=0; i<width; i++) {
        board[i][j] = Cell.EMPTY;
    }
}
```

Poäng: De två första raderna är värda 1 poäng. Den tredje raden är värd 1 poäng. Loopen som skriver `Cell.EMPTY` i `board` är värd 1 poäng.

b) Skriv koden för `choose`-metoden. Den bör skriva ett `X` i rutan med koordinater `i` och `j` ifall `xTurnNext` är sant, annars skriver den `O` i den rutan. I båda fallen skall den byta värdet på `xTurnNext`. Den bör kasta en exception ifall något är fel, t.ex. rutan inte finns eller inte är tom. [5 poäng]

Kod för choose-metoden:

```
if (i < 0 || w <= i ||
    j < 0 || h <= j ||
    board[i][j] != Cell.EMPTY) {
    throw new IllegalArgumentException("bad i,j for write");
}
if (xTurn) {
    board[i][j] = Cell.X;
} else {
    board[i][j] = Cell.O;
}
xTurn = !xTurn;
```

Poäng: Koden bör kasta `IllegalArgumentException` ifall koordinaterna är utanför spelplan. Det är inte viktigt att kasta exception ifall rutan inte är `Cell.EMPTY`, men i det fallet skall man också inte skriva på eller byta `xTurn`.

c) Skriv koden för `hasFiveInARow`-metoden. Skriv kod som returnerar sant ifall spelet har tagit slut, annars skall den returnera falskt. Den får *inte* kasta exception. [10 poäng]

Tips: Del (c) är kanske stiligast löst med en (eller flera) hjälpmetoder, t.ex. en metod som kollar om det finns fem av samma i en given riktning från givna koordinater `i` och `j`.

Det finns flera sätt att lösa denna uppgift. Här är en lösning:

```
private boolean hasCell(int i, int j, Cell c) {
    return 0 <= i && i < width &&
           0 <= j && j < height &&
           board[i][j] == c;
}

private boolean hasFiveInARowFromDir(int i, int j, Cell c, int id, int jd) {
    return hasCell(i+0*id,j+0*jd,c) &&
           hasCell(i+1*id,j+1*jd,c) &&
           hasCell(i+2*id,j+2*jd,c) &&
           hasCell(i+3*id,j+3*jd,c) &&
           hasCell(i+4*id,j+4*jd,c);
}

private boolean hasFiveInARowFrom(int i, int j, Cell c) {
    return hasFiveInARowFromDir(i,j,c,1,0)   // right
        || hasFiveInARowFromDir(i,j,c,0,1)   // down
        || hasFiveInARowFromDir(i,j,c,1,1)   // right-down diagonally
        || hasFiveInARowFromDir(i,j,c,1,-1); // right-up diagonally
}

public boolean hasFiveInARow() {
    for(int j=0; j<height; j++) {
        for(int i=0; i<width; i++) {
            if (hasFiveInARowFrom(i,j,Cell.O) ||
                hasFiveInARowFrom(i,j,Cell.X)) {
              return true;
            }
        }
    }
    return false;
}
```

Poäng: För fulla poäng är det viktigt att koden kollar om det finns fem av samma (`X` eller `O`, men inte `EMPTY`) i rad *någonstans* i `board` fältet. Koden får inte släppa ut exceptions, t.ex. `ArrayIndexOutOfBounds`. Här mister man inte poäng för långsam eller ful kod.

Uppgift 4: [9 poäng totalt] *deklarationer, arv, gränssnitt*

a) Förklara alla delar av deklarationen (declaration) nedan. [2 poäng]

```
private static int numberOfCars = 0;
```

`private` betyder att namnet `numberOfCars` inte syns utanför klassen.
`static` betyder att variabeln tillhör klassen, inte instanserna.
`int` är typen av variabeln `numberOfCars`.
`numberOfCars` är namnet på variabeln som deklareras.
" `= 0`" betyder att variabeln har utgångsvärdet 0.

b) Ge *ett exempel* på överskuggning (overriding). [2 poäng]

```
public class A {
    public int foo() { return 5; }
}

public class B extends A {
    public int foo() { return 6; }
}
```

Här överskuggar definitionen av `foo` i `B`, definitionen av `foo` i `A`.

c) Förklara *med ett exempel* vad abstrakta (abstract) metoder och klasser är. [3 poäng]

```
public abstract class D {
    public abstract int foo();
}
```

Här är metoden `foo` en abstrakt metod; vi skriver ingen kod för abstrakta metoder. Klassen D innehåller en abstrakt metod och därför måste den klassen också vara abstrakt. Abstrakta klasser kan man inte direkt göra insanser av, dvs det går inte att skriva `new D()`.

d) Förklara *med ett exempel* vad ett gränssnitt (interface) är. [2 poäng]

```
public interface I {
    public abstract int foo();
}
```

I Java är gränssnitt samlingar av metodsignaturer (metoddeklarationer). Gränssnittet `I` ovan innehåller en metodsignatur. Varje klass som säger att de implementerar detta gränsnitt måste implementera metoden `foo` enligt signaturen som är given i gränsnittet.

Poäng: Det blir *inte* poäng avdrag för små fel i terminologin.

Uppgift 5: [12 poäng totalt] *arv, händelsehantering, timer, swing*

a) Beskriv kort den inbyggda `JFrame` klassen (dvs. `javax.swing.JFrame`). [2 poäng]

`JFrame` klassen är en klass för fönster. I `JFrame` fönster kan man kan sätta knappar, textrutor och andra delar av ett grafiskt användargränssnitt (GUI). Man får fram `JFrame` fönstret med att anropa `setVisible` med indata `true`.

b) Beskriv kort den inbyggda `Timer` klassen (dvs. `javax.swing.Timer`). Beskriv vad en lyssnare är och hur `Timer` klassen använder sig av lyssnare, i detta fall en `ActionListener`. [3 poäng]

`Timer` klassen kan producera metodanrop till en given lyssnarmetod, antingen en gång eller repeterat. Timer klassen garanterar att det kommer att finnas en viss tid mellan anropen till lyssnaremetoden.

I Java är en lyssnare ett objekt som implementerar ett lyssnargränssnitt, t.ex. ActionListener. Lyssnargränssittet har metoder som ska köras när någonting händer, t.ex. ett knapptryck har skett.

För `Timer` klassen lyssnar lyssnaren på timer klockans ”tickningar”.

c) Skriv kod för en ny klass med namnet `SnapFrame`. Den nya klassen `SnapFrame` bör ärva `JFrame` och vara på nästan alla sätt precis samma som en `JFrame`. Klassen `SnapFrame` bör avvika från klassen `JFrame` när det gäller synlighet:

- En `SnapFrame` får endast vara synlig *en gång*, och då i *max* 5 sekunder.
- Efter att man anropar `setVisible(true)`, för första gången, skall `SnapFramen` *automatiskt* gömma sig efter 5 sekunder.
- Därefter bör det vara omöjligt att få fönstret synligt igen.

[7 poäng]

Obs: Det står max 5 sekunder. Se till att det är möjligt att gömma fönstret (med att kalla på `setVisible(false)`) innan det gått 5 sekunder.

Tips: Överskugga `setVisible` metoden och använd dig av superklassens `setVisible` metod.

```
public class SnapFrame extends JFrame implements ActionListener {

    private boolean hasBeenVisible = false;

    public void setVisible(boolean b) {
        if (hasBeenVisible) {
            super.setVisible(false);
        } else {
            super.setVisible(b);
            if (b) {
                hasBeenVisible = true;
                Timer t = new Timer(5000,this);
                t.setRepeats(false);
                t.start();
            }
        }
    }
    public void actionPerformed(ActionEvent e) {
        setVisible(false);
    }
}
```

Alternativt, kan man använda innerklasser eller `new ActionListener{ … }`.

Poäng: Det blir *inte* poäng avdrag ifall man glömmer att stänga av timern.

Uppgift 6: [13 poäng totalt] *grafik, rekursion, swing*

Skriv kod som ritar, med Javas `Graphics` klass, rektanglar på följande sätt:

- Varje rektangel innehåller två mindre rektanglar.
- De inre rektanglarna har också två (ännu mindre) rektanglar i sig.
- Dessa små rektanglar har igen rektanglar i sig, dvs samma upprepas.
- Detta mönster upprepas tills de minsta rektanglarna är för små för att ritas.

Rektanglarna ska ritas i en `JPanel` som sitter i ett `JFrame` fönster, se exemplen nedan.

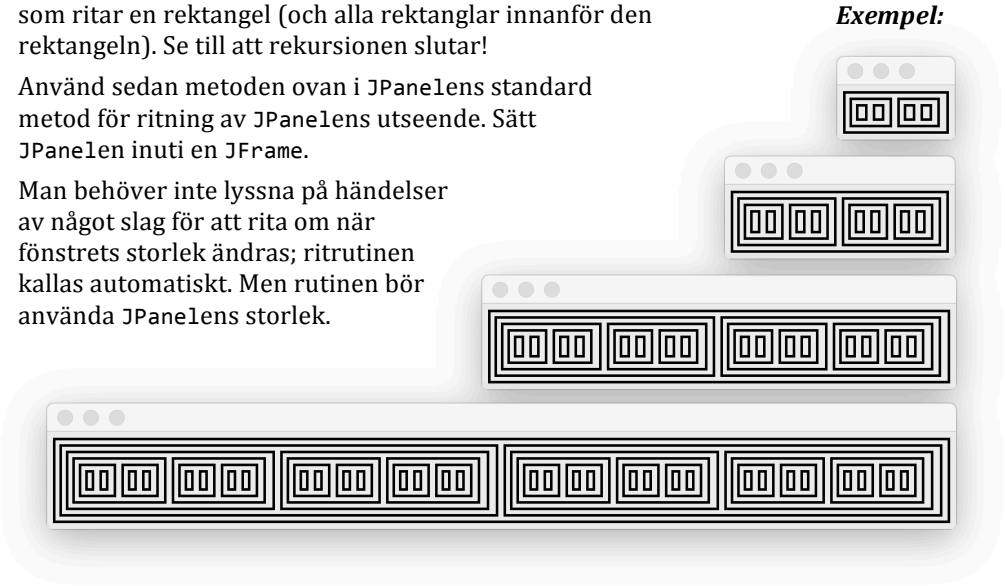
Obs. Rektanglarna bör fylla hela fönstret, även när användaren ändrar på fönstrets storlek.

Obs. Rektanglarnas linjer får inte kollidera eller ligga på varandra. *Förklara* varför de inte kolliderar / ligger på varandra med den koden som du har skrivit.

Tips:

- Det är kanske lättast att börja med att skriva den metoden som ritar en rektangel (och alla rektanglar innanför den rektangeln). Se till att rekursionen slutar!
- Använd sedan metoden ovan i `JPanel`ens standard metod för ritning av `JPanel`ens utseende. Sätt `JPanel`en inuti en `JFrame`.
- Man behöver inte lyssna på händelser av något slag för att rita om när fönstrets storlek ändras; ritrutinen kallas automatiskt. Men rutinen bör använda `JPanel`ens storlek.

Exempel:



```
public class RecDraw extends JPanel {

    private void drawMany(int x, int y, int width, int height, Graphics g) {
        g.drawRect(x,y,width,height);
        if (0 < width && 0 < height) {
            drawMany(x+5,y+5,width / 2 - 8,height - 10,g);
            drawMany(x+2 + width / 2,y+5,width / 2 - 8,height - 10,g);
        }
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        drawMany(5,5,getWidth()-10,getHeight()-10,g);
    }

    public static void main(String[] args) {
        JFrame r = new JFrame();
        JPanel p = new RecDraw();
        r.add(p);
        r.setVisible(true);
    }

}
```

Poäng: Max 7 poäng för den rekursiva ritmetoden (den behöver inte vara rekursiv), 3 poäng för korrekt överskuggning av `paintComponent` metoden; och 3 poäng för `JFrame` koden.

Utdrag ur Javas API. ***Obs.*** Man behöver inte använda alla dessa klasser. Man får också använda annat från Javas API.

`Interface ActionListener`

`void actionPerformed(ActionEvent e)`
Invoked when an action occurs.

`Class ActionEvent extends AWTEvent extends EventObject extends Object`

`ActionEvent(Object source, int id, String command)`
Constructs an ActionEvent object.
`String getActionCommand()`
Returns the command string associated with this action.

`Class Color extends Object`

`static Color BLACK`
The color black.

`Class Component extends Object`

`int getHeight()`
Returns the current height of this component.
`int getWidth()`
Returns the current width of this component.

`Class Container extends Component extends Object`

`Component add(Component comp)`
Appends the specified component to the end of this container.

`Class FlowLayout extends Object, implements LayoutManager`

`FlowLayout()`
Constructs a new FlowLayout with a centered alignment and a default 5-unit horizontal and vertical gap.

`Class Graphics extends Object`

`void drawRect(int x, int y, int width, int height)`
Draws the outline of the specified rectangle.
`abstract void fillRect(int x, int y, int width, int height)`
Fills the specified rectangle.
`abstract void setColor(Color c)`
Sets this graphics context's current color to the specified color.

`Class JComponent extends Container extends Component extends Object`

`protected void paintComponent(Graphics g)`
Calls the UI delegate's paint method, if the UI delegate is non-null.

`Class JFrame extends Frame extends Window extends Container extends Component extends Object`

`Container getContentPane()`
Returns the contentPane object for this frame.

`Class JPanel extends JComponent extends Container extends Component extends Object`

`JPanel()`
Creates a new JPanel with a double buffer and a flow layout.
`JPanel(LayoutManager layout)`
Create a new buffered JPanel with the specified layout manager

`Interface LayoutManager`

`Class Object`

`boolean equals(Object obj)`
Indicates whether some other object is "equal to" this one.
`String toString()`
Returns a string representation of the object.

`Class Timer extends Object`

`Timer(int delay, ActionListener listener)`
Creates a Timer and initializes both the initial delay and between-event delay to delay milliseconds.
`void setActionCommand(String command)`
Sets the string that will be delivered as the action command in ActionEvents fired by this timer.
`void setRepeats(boolean flag)`
If flag is false, instructs the Timer to send only one action event to its listeners.
`void start()`
Starts the Timer, causing it to start sending action events to its listeners.
`void stop()`
Stops the Timer, causing it to stop sending action events to its listeners.

`Class Window extends Container extends Component extends Object`

`void setVisible(boolean b)`
Shows or hides this Window depending on the value of parameter b.