Institutionen för Data- och informationsteknik
Chalmers Tekniska Högskola
Ansvarig lärare: Johannes Åman Pohjola

DAT050 Objektorienterad Programmering

Tenta, 2024-10-31, 08:30-12:30

Betygsgränser: U (0-23 poäng); 3 (24-35 poäng); 4 (36-47 poäng); 5 (48-60 poäng)

Vänligen notera:

- Frågor kan besvaras på svenska eller engelska.
- Skriv tydligt.
- När du gör antaganden som inte finns med i frågeställningen, var tydlig med det.

Lycka till!

Fråga 1: Begreppsförståelse (14 poäng)

Besvara följande frågor kortfattat (någon mening räcker) och med egna ord:

(a) Vad är relationen mellan en *klass* och ett *objekt*? (2 poäng)

(b) Vad gör nyckelordet **private** för någonting? (2 poäng)

(c) Vad är det för skillnad mellan ett *interface* och en *abstrakt klass*? (2 poäng)

(d) Vad innebär *låg koppling*, och varför är det önskvärt i objektorienterad programmering? (2 poäng)

(e) Vad skiljer en *referenstyp* från en *primitiv typ*? (2 poäng)

(f) Vad skiljer en *statisk metod* från en *instansmetod*? (2 poäng)

(g) Vad är *modellens* roll i designmönstret MVC? (2 poäng)

Fråga 2: Specifikationer (6 poäng)

(a)
```
/** Compute x distance between two points
    @param p1 a point
    @param p2 another point
    @return the distance along the x-axis between points p1 and
     p2
 */
static int distX(Point p1, Point p2) {
    return Math.abs(p1.x - p2.x);
}
```

Vilket potentiellt körfel har specifikationen av `distX` glömt nämna? Föreslå en ändring av specifikationen för att lösa problemet. (3 poäng)

(b) Skriv en specifikation i javadocformat som beskriver följande metods beteende:

```
static List<Integer> sub(List<Integer> l1, List<Integer> l2) {
    List<Integer> output = new LinkedList<>();
    for(int i = 0; i < l1.size(); i++) {
        if(i >= l2.size())
            return output;
        else
            output.add(l1.get(i) - l2.get(i));
    }
    return output;
}
```

(3 poäng)

Fråga 3: Programmera datastrukturer (10 poäng)

Betrakta följande implementation av klassen `Bag`, en enkel datastruktur som endast stödjer tre operationer: lägg till ett tal i påsen, returnera summan av alla heltal i påsen, och returnera medelvärdet av alla tal i påsen.

```
public class Bag {
    private LinkedList<Integer> elements;

    public Bag() {
        elements = new LinkedList<>();
    }

    public void add(int i) {
        elements.add(i);
    }

    public int sum() {
        int sum = 0;
        for(int i : elements)
            sum += i;
        return sum;
    }

    public int average() {
        return sum() / elements.length();
    }
}
```

Följande uppgifter ska lösas **fristående från varandra** om inte annat anges.

(a) Skriv en *kopieringskonstruktor* för `Bag` med följande signatur:

```
public Bag(Bag bag)
```

Konstruktorn bör se till att inga aliasingproblem kan uppstå mellan den nya och gamla påsen. (3 poäng)

(b) Skriv en *immutable* version av `Bag`. Du får använda konstruktorn från tidigare uppgift (oavsett om du löst uppgiften eller ej). Gränssnittet bör vara samma, förutom att `add` ska ha returtype `Bag`. (3 poäng)

(c) `Bag`s interna representation kan förenklas. Skriv en alternativ implementation som har samma beteende utåt sett, men har två instansvariabler av typen **int** istället för en instansvariabel av typen `LinkedList<Integer>`. (4 poäng)

Fråga 4: Programmera spelkort (15 poäng)

(a) Skriv en klass (och eventuella hjälpklasser) med signaturen

```
public class Card implements Comparable<Card> { ... }
```

som representerar ett spelkort i en vanlig 52-korts kortlek.[1]

Intern datarepresentation kan göras efter eget huvud, men klassen bör implementera följande funktionalitet:

- Konstruktor(er) så att alla kort kan skapas, och som ser till att inga nonsenskort kan skapas (t ex, ingen Spader 452).
- En `compareTo` som jämför om ett kort är mer värt än ett annat. [2]
- En metod som returnerar **true** om två kort har samma färg, med följande signatur: **public boolean** `sameSuite(Card other)`
- En implementation av `equals` som returnerar **true** om argumentet är ett `Card`-objekt med samma färg och valör, och **false** annars.

(9 poäng)

(b) Skriv en klass `Deck` som representerar en draghög av kort. En draghög består av noll eller flera kort i en viss ordning; använd `Card` från tidigare uppgift för att representera enstaka kort.[3] Skriv metoder för att:

- Konstruera en draghög med ett av varje kort i.
- Dra det översta kortet ur en draghög. Metoden bör returnera ett `Card`, och ta bort samma kort ur draghögen. Om draghögen är tom ska en (valfri) exception kastas.
- Blanda en draghög. Detta kan göras på olika sätt, men använd `Random` så att blandningen inte blir samma varje gång.

(6 poäng)

[1] Ett kort har en *valör* och en *färg*. Färgen är antingen *hjärter*, *klöver*, *ruter* eller *spader*. Valörerna är 2, 3, 4, 5, 6, 7, 8, 9, 10, Knekt, Dam, Kung, Ess.

[2] Kort med högre valör är mer värda än kort med lägre valör (2 är lägst, Ess är högst). Om korten har samma valör är hjärter värt mer än spader, som är värt mer än ruter, som är värt mer än klöver.

[3] om du inte löst tidigare uppgift, gör antaganden om hur `Card` fungerar och redovisa dem.

Fråga 5: GUI-programmering (15 poäng)

(a) Skriv ett program som ritar upp ett tomt fönster (`JFrame`). Fönstret ska ändra storlek varje gång muspekaren flyttas, så att muspekaren alltid befinner sig i mitten av fönstret. Notera dock följande undandtag:

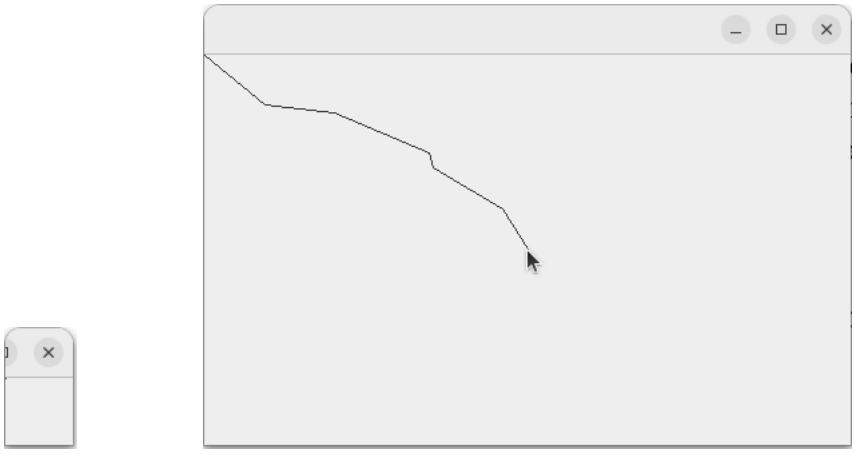
- Fönstret ska aldrig vara mindre än 50 pixlar brett i x-led.
- Fönstret ska aldrig vara mindre än 50 pixlar högt i y-led.

(5 poäng)

(b) Utöka programmet från föregående del så att ett spår ritas ut efter muspekarens rörelser. En gång per sekund, spara koordinaterna som muspekaren senast siktade på. När fönstret ritas upp, rita linjer som följer spåret av koordinater som sparats hittills.

(10 poäng)

Följande screenshots är ett exempel på hur en körning av programmet kan se ut. Till vänster visas fönstret från början (innan pekaren flyttats in i fönstret). Till höger visas fönstret efter att pekaren flyttats inom fönstret i några sekunder. Notera att fönstret förstorats så att muspekaren är mitten för (a)-uppgiften, och spåren som visar musrörelser hittills för (b)-uppgiften.



Om du löser båda delarna behöver inte (a)-delen redovisas separat.

(totalt 60p)

Utdrag ur Javas API

public class `LinkedList<E>`

`LinkedList()`
Constructs an empty list.

boolean `add(E e)`
Appends the specified element to the end of this list.

`E remove()`
Retrieves and removes the head (first element) of this list.

`E get(`**int** `index)`
Returns the element at the specified position in this list.

int `size()`
Returns the number of elements in this list.

public interface `Comparable<T>`

int `compareTo(T o)`
Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

public class `Random`

`Random()`
Creates a new random number generator.

int `nextInt()`
Returns the next pseudorandom, uniformly distributed int value from this random number generator's sequence.

public class `JPanel` **extends** `JComponent` **extends** `Container`

`JPanel()`
Creates a new JPanel with a double buffer and a flow layout.

public class `JFrame` **extends** `Frame` **extends** `Window` **extends** `Container`

`JFrame()`
Constructs a new frame that is initially invisible.

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

public abstract class `JComponent`

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

public class `Container` **extends** `Component`

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

public abstract class `Component`

public void `repaint()`
Repaints this component.

void `setSize(`**int** `width, `**int** `height)`
Resizes this component so that it has width width and height height.

public class `Graphics`

abstract void `drawLine(int x1, int y1, int x2, int y2)`
Draws a line, using the current color, between the points (x1, y1) and (x2, y2) in this graphics context's coordinate system.

public interface `MouseMotionListener`

void `mouseDragged(MouseEvent e)`
Invoked when a mouse button is pressed on a component and then dragged.

void `mouseMoved(MouseEvent e)`
Invoked when the mouse cursor has been moved onto a component but no buttons have been pushed.

public interface `ActionListener`

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

public class `Timer`

`Timer(int delay, ActionListener listener)`
Creates a Timer and initializes both the initial delay and between-event delay to delay milliseconds.

void `start()`
Starts the Timer, causing it to start sending action events to its listeners.

public class `MouseEvent`

int `getX()`
Returns the horizontal x position of the event relative to the source component.

int `getY()`
Returns the vertical y position of the event relative to the source component.

public class `Point`

`Point(int x, int y)`
Constructs and initializes a point at the specified (x,y) location in the coordinate space.

int `x`
The X coordinate of this Point.

int `y`
The Y coordinate of this Point.