Lösningsförslag

Objektorienterad programmering

Uppgift 1 Objektorientering allmänt (12p totalt)

(A) Förklara begreppen låg/hög koppling och kohesion. Beskriv så väl du kan med exempel varför kohesion och koppling i någon mån kan vara både individuellt höga/låga.

– Se föreläsning 11, sid 4–8, 27.

(B) Förklara kort hur ”mutation” kan utföras för icke-muterbara klasser (tänk String, BigInteger).

– Se föreläsning 5, sid 25–26.

(C) Förklara skillnaden mellan specifikation och implementation.

– Se föreläsning 2, sid 18–20, föreläsning 3, sid 9,

(D) Förklara skillnaden mellan klass- och instansvariabler.

– Se föreläsning 3, sid 25–27.

Uppgift 2 Beräkning med memorering (10p)

```
public class Collatz {
   private static final BigInteger THREE = BigInteger.valueOf(3);

   static BigInteger nextCollatz(BigInteger v) {
      if (!v.testBit(0)) {        return v.divide(BigInteger.TWO);      }
      return v.multiply(THREE).add(BigInteger.ONE);
   }

   // Ocachad version, inte så snabb.
   public static double getCollatzMeanNaive(int max) {
      long cAcc = 0;
      for (int i = 1; i <= max; i++) {
         int count = 0;
         BigInteger b = BigInteger.valueOf(i);
         while (!b.equals(BigInteger.ONE)) {
            count++;
            b = nextCollatz(b);
         }
         cAcc += count;
      }
      return cAcc / (double) max;
   }

   // Cachad/memoiserad version, det sökta svaret:
   public static double getCollatzMeanCached(int max) {
      long cAcc = 0;
      final Map<BigInteger, Integer> lookup = new HashMap<>();
      lookup.put(BigInteger.ONE, 0);

      for (int i = 1; i <= max; i++) {
         BigInteger b = BigInteger.valueOf(i);
         int count = 0;
         while (lookup.get(b) == null) {
            count++;
            b = nextCollatz(b);
         }
         count += lookup.get(b);
         lookup.put(BigInteger.valueOf(i), count);
         cAcc += count;
      }
      return cAcc / (double) max;
   }
}

Jämförelse, tidsåtgång:
Naive:  (10^7.0 values) 27.2338444 s
Cached: (10^7.0 values)  5.3229754 s
```

Uppgift 3 Tårtdiagram (15p totalt)

```
public class PiePanel extends JPanel {
   private final double[] vs; // We need to remember the slice sizes.

   // Assume vs != null && vs.length > 0 && all elements of vs are non-negative.
   public PiePanel(double[] vs) {
      double sum = 0;
      for (double e : vs) {
         sum += e;
      }
      this.vs = new double[vs.length];
      for (int i = 0; i < vs.length; i++) {
         this.vs[i] = vs[i] / sum;
      }
   }

   @Override public void paintComponent(Graphics g) {
      super.paintComponent(g);
      // MARGIN could also be 0, not very important when posed as an exam question.
      final int MARGIN = (int) (Math.max(this.getWidth(), this.getHeight()) * 0.02f);
      final int w = this.getWidth() / 2 - MARGIN;
      final int h = this.getHeight() / 2 - MARGIN;
      double prevAngle = 0;

      // Fill all slices.
      for (final double element : this.vs) {
         g.setColor(Color.getHSBColor((float) (prevAngle / 360.0), 0.9f, 0.9f));
         System.out.println(g.getColor() + " prevAngle: " + prevAngle + "\t" + element * 360);
         g.fillArc(MARGIN, MARGIN, w * 2, h * 2, (int) prevAngle, (int) (element * 360 + 0.5));
         prevAngle += element * 360;
      }

      // Draw the spokes.
      g.setColor(Color.black);
      double partSum = 0.0f;
      for (final double element : this.vs) {
         final double theta = partSum * Math.PI * 2;
         g.drawLine(w + MARGIN, h + MARGIN, MARGIN + (int) (w + Math.cos(theta) * w),
               MARGIN + (int) (h - Math.sin(theta) * h));
         partSum += element;
      }
      g.setColor(Color.BLACK);

      // Draw the enclosing circle.
      g.drawOval(MARGIN, MARGIN, this.getWidth() - MARGIN * 2, this.getHeight() - MARGIN * 2);
   }
}
```

Uppgift 4 Memoryspel (23p totalt)

```
public class MemoryModel {
   private final int[][] cards;
   private final int width;
   private final int height;
   private int score;
   private int guesses;

   public MemoryModel(int width, int height) {
      this.cards = new int[height][width];
      this.width = width;
      this.height = height;
      restart();
   }

   public final double getScore() {
      if (this.guesses == 0) {       return 0;    }
      return this.score / (double) this.guesses;
   }

   public final int getHeight() {     return this.height;   }

   public final int getWidth() {      return this.width;    }

   public final boolean isGameOver() {      return this.score == this.width * this.height; }

   public int getValue(int x, int y) {      return this.cards[y][x]; }

   public boolean guess(int x1, int y1, int x2, int y2) {
      this.guesses++;
      if (this.cards[y1][x1] == this.cards[y2][x2]) {
         this.score += 2;
         return true;
      }
      return false;
   }

   public void restart() {
      this.guesses = this.score = 0;
      final List<Integer> cards = new ArrayList<>();
      for (int i = 0; i < this.width * this.height; i++) {       cards.add(i / 2);     }
      Collections.shuffle(cards);
      for (int i = 0; i < this.height; i++) {
         for (int j = 0; j < this.width; j++) {
            this.cards[i][j] = cards.get(i * this.width + j);
         }
      }
   }
}
```

```java
public class MemoryController extends JPanel implements ActionListener {
   private final JLabel scoreLabel;
   private final MemoryModel model;
   private JButton firstButton;
   private JButton secondButton;
   // There are at least four distinct ways of mapping buttons to coordinates.
   // 1: Let this class implement ActionListener and ...
   //    A: Use setActionCommand with a text string that can be parsed
   //       as a cooordinate.
   //    B: Use two maps Map<JButton, Integer>, one for columns and one for rows.
   // 2: Let each button have a unique ActionListener that memorizes the coordinate
   //   through ...
   //    A: An anononymous class
   //    B: A lambda.
   // Note that all four versions are given below (there are other possibilities, too).

   // If chosing solution 1B:
   private final Map<JButton, Integer> rowMap = new HashMap<>();
   private final Map<JButton, Integer> colMap = new HashMap<>();

   public MemoryController(MemoryModel memoryModel) {
      this.model = memoryModel;
      this.setLayout(new BorderLayout());
      final JPanel buttonPanel = new JPanel();
      // We need to remember the buttons to reset them when restarting.
      final List<JButton> buttons = new ArrayList<>();
      buttonPanel.setLayout(new GridLayout(memoryModel.getHeight(), memoryModel.getWidth()));
      for (int r = 0; r < memoryModel.getHeight(); r++) {
         for (int c = 0; c < memoryModel.getWidth(); c++) {
            final JButton b = new JButton();
            // Solution 1A:
            b.setActionCommand(r + " " + c);
            buttonPanel.add(b);
            buttons.add(b);
            b.addActionListener(this);

            // Solution 1B:
            this.colMap.put(b, c);
            this.rowMap.put(b, r);

            // Solution 2A and 2B:
            final int col = c, row = r; // Needs final to use in a lambda or anon. class.

            // Solution 2A:
            b.addActionListener(e -> handleButton(e, col, row));

            // Solution 2B:
            b.addActionListener(new ActionListener() {
               @Override public void actionPerformed(ActionEvent e) {
                  handleButton(e, col, row);
               }
            });
         }
```

```
// constructor, continued.
    }
    resetButtons(buttons);
    this.add(BorderLayout.CENTER, buttonPanel);

    final JPanel statusPanel = new JPanel();
    this.scoreLabel = new JLabel("0");
    updateScoreLabel();
    statusPanel.add(this.scoreLabel);

    final JButton restartButton = new JButton("Restart");
    restartButton.addActionListener(e -> {
        memoryModel.restart();
        MemoryController.this.repaint();
        updateScoreLabel();
        resetButtons(buttons);
    });
    statusPanel.add(restartButton);

    this.add(BorderLayout.SOUTH, statusPanel);
}

private static void resetButtons(List<JButton> buttons) {
    final Font bFont = new Font("Arial", Font.BOLD, 30);
    for (final JButton b : buttons) {
        b.setText("      ");
        b.setFont(bFont);
        b.setBackground(Color.WHITE);
        b.setEnabled(true);
    }
}

private void updateScoreLabel() {
    this.scoreLabel.setText("Score: " + String.format("%.2f", this.model.getScore()));
}

private int getRow(JButton b) {
    // Solution 1A:
    final String[] ac = b.getActionCommand().split(" ");
    return Integer.parseInt(ac[0]);

    // Solution 1B:
    return rowMap.get(b);
}

private int getCol(JButton b) {
    // Solution 1A:
    final String[] ac = b.getActionCommand().split(" ");
    return Integer.parseInt(ac[1]);

    // Solution 1B:
    return colMap.get(b);
}
```

```java
// Solution 1A and 1B:
@Override
public void actionPerformed(ActionEvent e) {
   if (this.firstButton == null) {
      this.firstButton = (JButton) e.getSource();
      showButtonValue(this.firstButton);
   } else if (this.secondButton == null) {
      this.secondButton = (JButton) e.getSource();
      showButtonValue(this.secondButton);
      if (this.model.guess(getCol(this.firstButton), getRow(this.firstButton),
            getCol(this.secondButton), getRow(this.secondButton))) {
         this.firstButton.setBackground(Color.GREEN);
         this.secondButton.setBackground(Color.GREEN);
         this.firstButton.setEnabled(false);
         this.secondButton.setEnabled(false);
         this.firstButton = null;
         this.secondButton = null;
      }
      updateScoreLabel();
   } else { // Two cards already up. Flip them down ...
      this.firstButton.setText("");
      this.secondButton.setText("");
      this.firstButton = this.secondButton = null;
      actionPerformed(e);          // and flip up the new card.
   }
}

// Solution 2A & 2B:
public void handleButton(ActionEvent e, int col, int row) {
   if (this.firstButton == null) {
      this.firstButton = (JButton) e.getSource();
      showButtonValue(this.firstButton, col, row);
   } else if (this.secondButton == null) {
      this.secondButton = (JButton) e.getSource();
      showButtonValue(this.secondButton, col, row);
      if (this.model.guess(getCol(this.firstButton), getRow(this.firstButton),
            getCol(this.secondButton), getRow(this.secondButton))) {
         this.firstButton.setBackground(Color.GREEN);
         this.secondButton.setBackground(Color.GREEN);
         this.firstButton.setEnabled(false);
         this.secondButton.setEnabled(false);
         this.firstButton = this.secondButton = null;
      }
      updateScoreLabel();
   } else { // Two cards already up. Flip them down ...
      this.firstButton.setText("");
      this.secondButton.setText("");
      this.firstButton = null;
      this.secondButton = null;
      handleButton(e, col, row);         // and flip up the new card.
   }
}
```

```
   // Solution 1A and 1B:
   private void showButtonValue(JButton b) {
      final int row = getRow(b);
      final int col = getCol(b);
      b.setText(Integer.toString(this.model.getValue(col, row)));
   }

   // Solution 2A & 2B:
   private void showButtonValue(JButton b, int col, int row) {
      b.setText(Integer.toString(this.model.getValue(col, row)));
   }

   // Demonstration/example main.
   // Give number of columns and rows as command line arguments.
   public static void main(String[] args) {
      if(args.length != 2) {
         System.err.println("Number of columns and rows must be given.");
         System.exit(1);
      }
      int cols = Integer.parseInt(args[0]);
      int rows = Integer.parseInt(args[1]);
      if(cols < 1 || rows < 1 || cols * rows % 2 != 0) {
         System.err.println("Number of columns must be strictly positive and their " +
            " product must be even.");
         System.exit(2);
      }
      final JFrame f = new JFrame("Memory!");
      final JPanel p = new MemoryController(new MemoryModel(cols, rows));
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      f.add(p);
      f.pack();
      f.setVisible(true);
   }
}
```