Lösningsförslag till tentamen

Kurs	**Objektorienterad programmering**
Kursbeteckning	**DAT042**
Tentamensdatum	**2013-01-17**
Program	**D2, TKDAT**
Läsår	**2012/2013, lp 1**
Examinator	**Uno Holmer**

Uppgift 1 (10 p)

Ingen lösning ges. Se kurslitteraturen.

Uppgift 2 (12 p)

```
public class Game implements ActionListener {
    private JFrame frame;
    private JButton leftButton;
    private JButton centerButton;
    private JButton rightButton;
    private JLabel gameCounts;
    private JLabel score;
    private GameState gameState;

    public Game() {
        gameState = new GameState();
        makeFrame();
    }

    private void score( int buttonId ) {
        gameState.score( buttonId );
        score.setText(Integer.toString(gameState.getScore()));
        gameCounts.setText(Integer.toString(gameState.gamesLeft()));
        if ( gameState.isNewGame() ) {
            leftButton.setEnabled(true);
            centerButton.setEnabled(true);
            rightButton.setEnabled(true);
        }
    }

    public void actionPerformed(ActionEvent e) {
        if ( e.getSource() == leftButton )  {
             leftButton.setEnabled(false);
             score( 1 );
        } else if ( e.getSource() == centerButton ) {
             centerButton.setEnabled(false);
             score( 2 );
        } else if ( e.getSource() == rightButton ) {
             rightButton.setEnabled(false);
             score( 3 );
        }
    }
```

```
    private void makeFrame() {
        setTitle("Game");
        setLayout(new GridLayout(2,1));

        // Buttons
        JPanel buttons = new JPanel();
        buttons.setLayout(new GridLayout(1,3));
        leftButton = new JButton("Left");
        leftButton.addActionListener(this);
        centerButton = new JButton("Center");
        centerButton.addActionListener(this);
        rightButton = new JButton("Right");
        rightButton.addActionListener(this);
        buttons.add(leftButton);
        buttons.add(centerButton);
        buttons.add(rightButton);
        add(buttons);

        // Text labels
        JPanel outputFields = new JPanel();
        outputFields.setLayout(new GridLayout(1,2));
        gameCounts = new JLabel();
        gameCounts.setHorizontalAlignment(SwingConstants.CENTER);
        score = new JLabel();
        score.setHorizontalAlignment(SwingConstants.CENTER);
        score.setText(Integer.toString(gameState.getScore()));
        gameCounts.setText(Integer.toString(gameState.gamesLeft()));
        outputFields.add(score);
        outputFields.add(gameCounts);
        add(outputFields);
        pack();
        setVisible(true);
    }
```

Uppgift 3 (8 p)

```
public class ObjectCounter {
    private static HashMap<String,Integer> objectCounts =
        new HashMap<String,Integer>();

    public ObjectCounter() {
        String thisSubClass = getClass().getName();
        Integer c = objectCounts.get(thisSubClass);
        if ( c == null )
            objectCounts.put(thisSubClass,1);
        else
            objectCounts.put(thisSubClass,c+1);
    }

    public int getCount() {
        return objectCounts.get(getClass().getName());
    }
}
```

Uppgift 4 (7+8 p)

a)
`1-3:` obj har statisk typ Base och dynamisk typ Sub

`1: Sub.f1`	`f1` definieras om i `Sub`
`2: Base.f2++Sub.f2`	`f2` definieras om i `Sub` och den börjar med att anropa `Base.f2` explicit
`3: Base.f3`	`f3` definieras ej om i `Sub` utan ärvs från `Base`

`4-7: obj2` har statisk och dynamisk typ `Sub`. Subklassens metoder anropas utom i fall 6 där `f3` ärvs från basklassen.

```
4: Sub.f1
5: Base.f2++Sub.f2
6: Base.f3
7: Sub.f4
```

b)
`Int obj1 = new Int();`
Gränssnitt är abstrakta klasser och sådana får ej instansieras.

`Base obj2 = new Base();`	`Base` är abstrakt.
`Sub1 obj3 = new Sub1();`	Se nedan.
`Sub2 obj4 = new Sub2();`	Korrekt.

Klassen `Sub1` kan ej kompileras eftersom den abstrakta metoden `h` ej är implementerad vare sig i `Base` eller i `Sub1`. Notera att h ej måste definieras i `Base`. Eftersom klassen är abstrakt kan definitionen av `h` skjutas upp till någon subklass längre ner i hierarkin. Alla mellanliggande klasser blir då abstrakta.

Uppgift 5 (8 p)

```
public class BackupDaemon extends Thread {
    private Serializable object = null;
    private String backupFile;
    private int backupInterval;     // seconds

    public BackupDaemon( Serializable object, String backupFile,
                         int backupInterval )
    {
        this.object = object;
        this.backupFile = backupFile;
        this.backupInterval = backupInterval;
        start();
    }

    public void run() {
        while ( true ) {
            try {
                Thread.sleep( backupInterval*1000 );
                saveObject();
                System.out.println("Object written to "+backupFile );
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    private void saveObject() {
         try {
             ObjectOutputStream out =
                 new ObjectOutputStream(
                     new FileOutputStream(backupFile));
             out.writeObject(object);
             out.close();
         }
         catch ( IOException e){
             System.out.println("Cannot write to: " + backupFile);
         }
    }
}
```

Uppgift 6 (7 p)

```
public class A extends Observable {
    private int value = 0;

    public void compute(int x) {
        value += x;
        setChanged();
        notifyObservers();
    }

    public int getValue() {
        return value;
    }
}

public class B implements Observer {
    private A theAObject;

    public B(A anAObject) {
        theAObject = anAObject;
        theAObject.addObserver(this);
    }
    public void update(Observable o, Object arg) {
        if ( o instanceof A ) {
            System.out.println(theAObject.getValue());
        }
    }
}
```