Lösningsförslag till tentamen

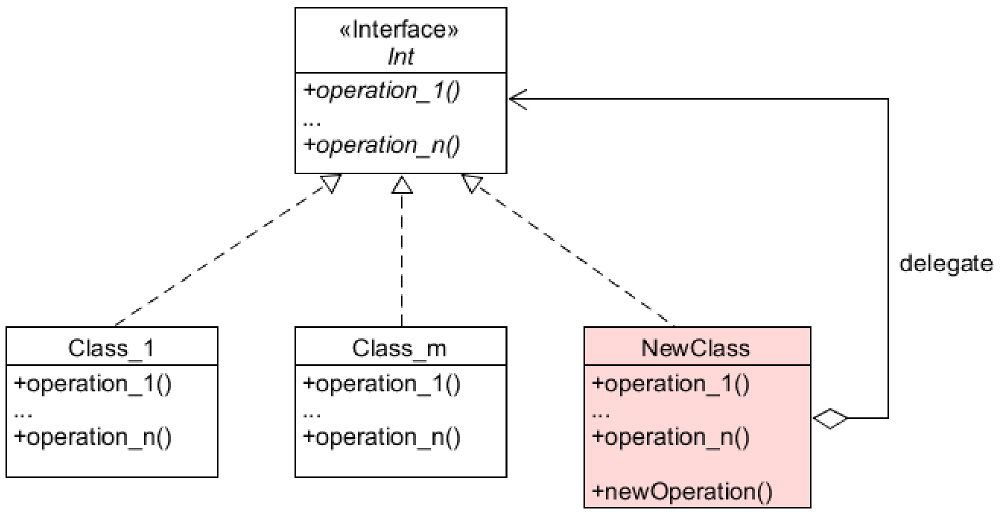
Kursnamn	**Objektorienterade applikationer**
Tentamensdatum	**2018-03-18**
Program	**DAI2**
Läsår	**2017/2018, lp 3**
Examinator	**Uno Holmer**

Uppgift 1

a) (4 p)

```
public class NoRepeatDie implements Die {
    private Die die;
    public NoRepeatDie(Die die) {
        this.die = die;
    }
    @Override
    public int getValue() {
        return die.getValue();
    }
    @Override
    public void roll() {
        int x = getValue();
        do {
            die.roll();
        } while ( getValue() == x );
    }
}
```

b) (2 p)



Uppgift 2

a) (6 p)
```
public class ConfigIO {
    public static void save(Map<String,String> bindings,String fileName)
    throws IOException
    {
        PrintWriter pw = new PrintWriter(new FileWriter(fileName));
        for ( Map.Entry<String,String> e : bindings.entrySet() )
            pw.println(e.getKey() + ":" + e.getValue());
        pw.close();
    }
    public static Map<String,String> load(String fileName) throws IOException {
        Map<String,String> result = new TreeMap<>();
        Scanner sc = new Scanner(new FileReader(fileName));
        while ( sc.hasNextLine() ) {
            String[] row = sc.nextLine().split(":");
            checkRow(row);
            result.put(row[0],row[1]);
        }
        sc.close();
        return result;
    }

    private static void checkRow(String[] row) throws IOException {
        if ( row.length != 2 )
           throw new IOException("Illegal data format found in config file.");
    }
}
```

b) (8 p)

```
public class ConfigData {
    private final static String CONFIGFILE = "config.txt";
    private Map<String,String> configMap;

    public ConfigData() throws IOException {
        configMap = ConfigIO.load(CONFIGFILE);
    }

    public Set<String> getKeys() {
        return configMap.keySet();
    }

    public String getValue(String key) {
        return configMap.get(key);
    }

    public void setValue(String key,String value) throws IOException {
        configMap.put(key,value);
        ConfigIO.save(configMap,CONFIGFILE);
    }
}
```

Uppgift 3 (10 p)

```
public class View extends JFrame {
    private ConfigData configData;

    public View(ConfigData configData) {
        this.configData = configData;
        makeMenu();
        pack();
        setVisible(true);
    }
    private void makeMenu() {
        JMenuBar menuBar = new JMenuBar();
        setJMenuBar(menuBar);
        JMenu configMenu = new JMenu("Config");
        menuBar.add(configMenu);
        for ( String key : configData.getKeys() ) {
            JMenuItem i = new JMenuItem(key);
            i.addActionListener(e -> askValue(key));
            configMenu.add(i);
        }
    }
    private void askValue(String key) {
        String oldValue = configData.getValue(key);
        String newValue = JOptionPane.showInputDialog(null,key,oldValue);
        if ( newValue != null && ! newValue.equals(oldValue) )
            try {
                configData.setValue(key,newValue);
            }
            catch ( IOException e ) {
                JOptionPane.showMessageDialog(null,e.getMessage(),"",
                                              JOptionPane.ERROR_MESSAGE);
            }
    }
}
```

Uppgift 4

a) (4 p)
Klassen `Job` måste imploementera gränssnittet `Serializable`.

```
public static void sendJob(String address,int port, Job job) throws IOException
{
    Socket sock = new Socket(address,port);
    ObjectOutputStream out = new ObjectOutputStream(sock.getOutputStream());
    out.writeObject(job);
    sock.close();
}
```

b) (5 p)

```
public class Server {
    private SimpleQueue<Job> queue = new SimpleQueue<>();
    public Server(int port) {
        serverLoop(port);
    }
    private void serverLoop(int port) {
        try {
            ServerSocket serverSock = new ServerSocket(port);
            while ( true )
                new ClientHandler(serverSock.accept(),queue);
        }
        catch ( Exception e ) {
            e.printStackTrace();
        }
    }
}
```

c) (6 p)

```
public class ClientHandler extends Thread {
    private Socket clientSock;
    private SimpleQueue<Job> queue;
    public ClientHandler(Socket clientSock,SimpleQueue<Job> queue)
    throws Exception
    {
        this.clientSock = clientSock;
        this.queue = queue;
        start();
    }
    @Override
    public void run() {
        try {
            ObjectInputStream in =
                new ObjectInputStream(clientSock.getInputStream());
            while ( true ) {
                Object msg = in.readObject();
                if ( msg instanceof Job )
                    queue.put((Job)msg);
            }
        }
        catch ( SocketException e ) {
            System.out.println("Client disconnected");
        }
        catch ( Exception e ) {
            e.printStackTrace();
        }
    }
}
```

Uppgift 5

a) (3 p)
I klassen `AbstractSignal`:

```
public abstract class AbstractSignal extends Observable implements Signal {
    ...
    public void goOn() {
         ...
            setChanged();
            notifyObservers();
    }
    public void goOff() {
         ...
            setChanged();
            notifyObservers(  );
    }
    ...
}
```

I klassen `SignalGui`:

```
public class SignalGui extends JFrame implements Observer {
    ...
    public void update(Observable obs,Object o) {
        if ( obs instanceof Signal ) {
            Signal s = (Signal)obs;
            if ( s instanceof RedSignal )
                redLamp.switchOnOff(s.isOn());
            else if ( obs instanceof YellowSignal )
                yellowLamp.switchOnOff(s.isOn());
            else if ( obs instanceof GreenSignal )
                greenLamp.switchOnOff(s.isOn());
        }
    }
}
```

Dessutom måste `SignalGui` läggas till som observatör på signalobjekten, vilket lämpligen görs i koden som sätter ihop systemets komponenter. Se deluppgift c.

b) (7 p)

```
public class SignalController extends Thread {
    private final long YELLOW_STOPPING_INTERVAL = 5000;
    private final long YELLOW_PROCEEDING_INTERVAL = 1000;
    private final long SIGNAL_INTERVAL = 20000;
    private enum State {PROCEEDING,STOPPED}
    private State state;
    private Signal redSignal;
    private Signal yellowSignal;
    private Signal greenSignal;
    public SignalController(Signal greenSignal,Signal yellowSignal,
                            Signal redSignal)
    {
        this.greenSignal = greenSignal;
        this.yellowSignal = yellowSignal;
        this.redSignal = redSignal;
        state = State.STOPPED;
    }

    public void proceedSequence() throws InterruptedException {
        if ( state == State.STOPPED ) {
            yellowSignal.goOn();
            sleep(YELLOW_PROCEEDING_INTERVAL);
            yellowSignal.goOff();
            redSignal.goOff();
            greenSignal.goOn();
            state = State.PROCEEDING;
        }
    }

    public void stopSequence() throws InterruptedException {
        if ( state == State.PROCEEDING ) {
            greenSignal.goOff();
            yellowSignal.goOn();
            sleep(YELLOW_STOPPING_INTERVAL);
            yellowSignal.goOff();
            redSignal.goOn();
            state = State.STOPPED;
        }
    }

    public void run() {
        redSignal.goOn();
        try {
            sleep(SIGNAL_INTERVAL);
            while ( ! interrupted() ) {
                proceedSequence();
                sleep(SIGNAL_INTERVAL);
                stopSequence();
                sleep(SIGNAL_INTERVAL);
            }
        }
        catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

c) (5 p)

```
public class TrafficLight {
    private TrafficLight() {
        AbstractSignal greenSignal = new GreenSignal();
        AbstractSignal yellowSignal = new YellowSignal();
        AbstractSignal redSignal = new RedSignal();
        SignalGui gui = new SignalGui();
        greenSignal.addObserver(gui);
        yellowSignal.addObserver(gui);
        redSignal.addObserver(gui);
        (new SignalController(greenSignal,yellowSignal,redSignal)).start();
    }

    public static void main(String[] arg) {
        new TrafficLight();
    }
}
```