FLUM

Ohjelmistotuotanto, kurssikoe 9.5.2015

Kirjoita jokaiseen palauttamaasi konseptiin kurssin nimi, kokeen päivämäärä, nimesi ja opiskelijanumerosi.

Kaikkien tehtävien vastaukset tulee tehdä omille konsepteille.

Vastaa tehtäviin ytimekkäästi. Sopiva vastauksen pituus kuhunkin tehtävistä on noin 1-2 sivua. **Täydet pisteet voi saada ainoastaan kirjoittamalla järkevästi muotoillun esseen, ranskalaiset viivat eivät riitä.**

1. (4p) Kerro mitä tarkoitetaan product backlogilla ja sprint backlogilla. Minkälainen on hyvä product backlog? Entä hyvä sprint backlog? Miten scrum-tiimin eri rooleissa olevat henkilöt hyödyntävät backlogeja?

2. (2+1.5+1.5p)

   (a) Ketterien menetelmien yhteydessä puhutaan usein käsitteistä definition of done ja hyväksymiskriteerit (engl. acceptace criteria). Selitä mitä käsitteillä tarkoitetaan. Anna konkreettinen esimerkki molemmista

   (b) Mitä tarkoitetaan regressiotestaamisella? Miten regressiotestaus yleensä hoidetaan kun ohjelmistoja kehitetään ketterien periaatteiden mukaan?

   (c) Mitä tarkoitetaan tutkivalla testaamisella? Minkälaisiin tilanteisiin tutkiva testaus sopii ja mihin se ei sovi?

3. (5p) Alla on lueteltu kuusi kurssilla vastaan tullutta termiä. Selitä lyhyesti **viiden** termin merkitys, missä niitä käytetään ja mikä niiden hyödyt hyödyt ovat?

   - minimal viable product
   - work in progress limit
   - velositeetti
   - staging-palvelin tai staging-ympäristö
   - stub- ja mock-oliot
   - mikropalvelmalli/arkkitehtuuri (microservice)

   HUOM: jätä yhden termin merkitys selittämättä. Jos kerrot jokaisesta termistä, arvostellaan viisi huonointa vastausta.

4. (1+2+3p)

   (a) Mitä tarkoitetaan ohjelman ulkoisella ja sisäisellä laadulla?

   (b) Mitä sisäisen laadun kannalta ongelmallisia asioita tehtäväpaperin lopun koodissa on?

   (c) Selitä miten refaktoroisit tehtäväpaperin lopun koodia soveltaen suunnittelumalleja tai muita tilanteeseen sopivia ratkaisuja. Vastaukseen tulee liittää mukaan luonnosmaista koodia, pelkkä suunnittelumallien nimien luetteleminen ei riitä. Älä kuitenkaan kirjoita "javaboilerplatea" (luokka+näkyvyysmäärittelyt ym.) kokonaisuudessaan

```
public class Main {
    public static void main(String[] args) {
        Company ibm = new Company("database");
        // adding employees etc...

        // print hour workers in csv-format
        for (Employee hourWorker : ibm.hourWorkes()) {
            System.out.println(hourWorker.formatAs("csv"));
        }

        // print emplyees with age less than 50 in json-format
        for (Employee youngEmployee : ibm.ageLessThan(50)) {
            System.out.println(youngEmployee.formatAs("json"));
        }
    }
}
```

```
public class Company {
    private List<Employee> employees;
    private String source;

    public Company(String source) {
        this.source = source;
        if ( source.equals("file")) {
            readEmployeesFromFile();
        } else if ( source.equals("database")) {
            readEmployeesFromDatabase();
        }
    }

    public List<Employee> hourWorkes(){
        List<Employee> result = new ArrayList<>();

        for (Employee employee : employees) {
            if ( employee.isHourWorker() ) {
                result.add(employee);
            }
        }

        return result;
    }

    public List<Employee> withMonthSalaryAtLeast(int amount){
        List<Employee> result = new ArrayList<>();

        for (Employee employee : employees) {
            if ( !employee.isHourWorker() && employee.getSalary()>amount ) {
                result.add(employee);
            }
        }

        return result;
    }

    public List<Employee> ageLessThan(int limit){
        List<Employee> result = new ArrayList<>();
```

```
        for (Employee employee : employees) {
            if ( employee.getAge() < limit ) {
                result.add(employee);
            }
        }

        return result;
    }

    // also many more simillar finder methods...

    public void setNewSallary(String name, int salary){
        for (Employee employee : employees) {
            if ( employee.getName().equals(name)) {
                employee.setSalary(salary);
                return;
            }
        }
        throw new IllegalStateException(name+" was not found");
    }

    public void newEmployee(Employee e){
        for (Employee employee : employees) {
            if ( employee.getName().equals(e.getName())) {
                throw new IllegalStateException(e.getName()+" was already added");
            }
        }

        employees.add(e);
        if ( source.equals("file")) {
            saveEmplyeeToFile(e);
        } else if ( source.equals("database")) {
            saveEmplyeeToDatabase(e);
        }
    }

    // details of the following not shown
    private void readEmployeesFromFile() { ... }
    private void readEmployeesFromDatabase() { ... }
    private void saveEmplyeeToFile(Employee e) { ... }
    private void saveEmplyeeToDatabase(Employee e) { ... }
}

public class Employee {
    private String name;
    private int age;
    private String sex;
    private int salary;
    private int hoursWorked;
    private boolean hourWorker;

    public Employee(String name, int age, String sex, int salary, boolean monthly) {
        this.name = name;
        this.age = age;
        this.sex = sex;
        this.salary = salary;
        this.hourWorker = monthly;
    }
```

interface

```
    public int salaryToPay(){
        int amount = 0;

        if ( hourWorker ) {
            amount = hoursWorked*salary;
            hoursWorked = 0;
        } else {
            amount = salary;
        }

        return amount;
    }

    public void addHours(int hours ){
        if ( hourWorker ) {
            hoursWorked += hours;
        }
    }

    public String formatAs(String format){
        if ( format.equals("csv")) {
            String payment = "";
            if ( hourWorker ) {
               payment = salary+"/hour";
            } else {
               payment = salary+"/month";
            }

            return  name+";"+age+";"+sex+";"+payment;
        } else if ( format.equals("json")) {
            String payment = "";
            if ( hourWorker ) {
               payment = salary+"/hour";
            } else {
               payment = salary+"/month";
            }
            return  "{ name:"+name+", age:"+age+", sex: "+sex+", payment:"+payment+" }";
        } else if ( format.equals("xml")) {
            // details not shown
            return "";
        }
        // more formats will be added later

        throw new IllegalStateException("format "+format+" not supported");
    }

    // getters and setters
    public int getAge() { return age; }
    public String getName() { return name; }
    public String getSex() { return sex; }
    public boolean isHourWorker() { return hourWorker; }
    public int getSalary() { return salary; }
    public void setSalary(int salary) { this.salary = salary; }

}
```