Skriftlig eksamen for 1. semester datamatiker

Efterår 2018

l18dat1ae | l18dat1ab | l18dat1ac

Prøvens varighed: 4 timer

Dato: 18.12.2018 kl. 8.30 til 12.30

Tilladte hjælpemidler: Alle skriftlige hjælpemidler inkl. online ressourcer.

Det er ikke tilladt at kommunikere med andre under eksamen. Overtrædelse vil medføre bortvisning.

IT afdelingen logger al trafik over skolens netværk, og de studerende skal aflevere deres mobiltelefoner ved prøvens start.

Materiale:

Netbeans projekt
Du skal downloade et Netbeans projekt som zip fil fra Wiseflow, som din besvarelse skal tage udgangspunkt.

Projektet består af:

- Java klasser, som skal implementeres færdigt
- JUnit testklasser, som skal kunne afvikle de implementerede Java klasser succesfuldt

Opgaverne
Eksamenssættet består af en række delopgaver. Vægtning af de enkelte opgaver fremgår af opgavebeskrivelserne nedenfor.

Procenttallene for opgaverne er vejledende. Man kan maksimalt opnå 100 point. Point opnås via succesfuld testafvikling, men den overordnede kodekvalitet påvirker den samlede karakter.

Aflevering: Du skal uploade din besvarelse som et Netbeans projekt eksporteret som .zip-fil på WiseFlow.

Opgave 1 (20 %)

Der skal implementeres tre metoder i klassen **`Strings`**

a. (7 %)

```
public String makeSQLString(String[] columns, String table)
```

Metoden modtager:

- et strengarray **`columns`** til angivelse af kolonnenavne i en tabel
- en streng **`table`** til angivelse af en tabel i en database.

Metoden returnerer et SQL select udtryk bygget op om de modtagne kolonner og tabelnavnet.

Eksempel,
Hvis metoden modtager et strengarray med værdierne **`id`** og **`name`** og tabelnavnet **`person`**, returnerer metoden:

```
select id, name from person;
```

b. (7 %)

```
public boolean isLowerCase(String str)
```

Metoden modtager en streng og returnerer **`true`**, hvis strengen udelukkende indeholder tegn mellem a – z eller er en tom streng.

Tip: Asciiværdien for tegnet 'a' er 97 og 122 for 'z'.

c. (6 %)

```
public boolean isSameBackwards(String str)
```

Metoden modtager en streng og returnerer **`true`**, hvis strengen er ens forfra og bagfra.

Hvis inputstrengen er `null` eller den tomme streng, kastes der en **`IllegalArgumentException`**.

Eksempler:

"abba" returnerer **`true`**
"ab bA" returnerer **`true`**
"bbba" returnerer **`false`**

Opgave 2 (20 %)

Der skal implementeres 2 metoder i klassen **`LabelMaker`**.

a. (6 %)

`public String getCity(String zipcode) throws NotFoundZipcodeException`

Metoden modtager et postnummer, og returnerer den tilknyttede by. Postnumre og byer findes i en datastruktur i klassen `PostalCodes`.

Du får behov for at lave en hjælpemetode i klassen `PostalCodes`.

Der skal kastes en `NotFoundZipcodeException` hvis der ikke findes en by med det pågældende postnummer.

b. (14 %)

`public String makeLabel(String name, String adress, String zipcode) throws NotFoundZipcodeException`

Metoden modtager en række strenge, som skal sættes sammen til én returstreng, der repræsenterer en pakkemærkat. Der skal mellemrum imellem hver information i returstrengen.

Dog mangler en information, som skal indgå i returstrengen, nemlig postdistrikt.

Metoden skal skaffe denne information ved brug af `getCity` metoden fra opgave 2a.

Der skal kastes en `NotFoundZipcodeException` videre, hvis der ikke findes en by med det pågældende postnummer.

Opgave 3 (20 %)

I denne opgave indgår der tre klasser/typer:

`Student`	entitetsklasse, der indeholder information om et `Student` objekt.
`Education`	`enum`, til at repræsentere studieretning for en studerende
`Students`	indeholder metoder til at søge i en liste af `Student` objekter

a. (4 %)

I klassen **`Student`** skal der defineres en række attributter og tilhørende konstruktørmetode til at håndtere følgende informationer:

- Studie-id (type `String`)
- Navn (type `String`)
- Uddannelsesretning (type `Education`)

b. (8 %)

I klassen **`Students`** implementeres metoden:

```
public Student retrieveStudent(ArrayList<Student> all, String studyId)
```

`M`etoden modtager en arrayliste af `Student` objekter, og et studie-id.

Metoden returnerer et `Student` objekt fra listen med matchende studie-id.

c. (8 %)

I klassen **`Students`** implementeres metoden:

```
public ArrayList<Student> retrieveStudents(ArrayList<Student> all, Education education)
```

`M`etoden modtager en arrayliste af `Student` objekter, og en uddannelsesretning.

Metoden returnerer en liste af studerende med matchende uddannelsesretning.

Opgave 4 (20 %)

I klassen **`WeatherInfoAssembler`** implementeres metoden:

```
public WeatherInfo gatherWeatherInfo(int[] temps)
```

`M`etoden modtager en array af heltal, som repræsenterer en række temperaturmålinger.

Metoden skal returnere et `WeatherInfo` objekt, som indeholde den højeste, den laveste og gennemsnitstemperaturen (sidstnævnte afrundet til en decimal). Du skal selv tilføje relevante attributter og metoder til klassen `WeatherInfo`.

Opgave 5 (20 %)

I denne opgave indgår der to klasser:

`Car`	entitetsklasse, der indeholder information om et `Car` objekt.
`CarHandler`	indeholder metoder der anvender em liste af `Car` objekter

I klassen **`CarHandler`** skal der implementeres en række metoder. Du kan i den forbindelse have behov for at lave tilføjelser i **`Car`** klassen.

a. (10 %)

```
public void sortByMakeModelYear(ArrayList<Car> cars)
```

Metoden modtager en arrayliste af `Car` objekter, som skal sorteres efter mærke. Hvis to biler er af samme mærke, skal de dernæst sorteres efter model. Hvis de både er samme mærke og model, skal de sorteres efter år. Rækkefølgen skal bevares.

b. (10 %)

```
public void sortByYearMakeModel(ArrayList<Car> cars)
```

Metoden modtager en arrayliste af `Car` objekter, som skal sorteres efter år. Hvis to biler er af samme år, skal de dernæst sorteres efter mærke. Hvis de både er samme år og mærke, skal de sorteres efter model. Rækkefølgen skal bevares.