Politechnika Krakowska

Wydział Fizyki, Matematyki i Informatyki

ZADANIE 7

Zaawansowane techniki programowania

Prowadzący: dr inż. Jerzy Jaworowski

II stopień - rok 1, semestr 1

wykonanie:

Piotr Adam Tomaszewski

Nr albumu: 104896

Wstęp

Celem zadania było zaimplementowanie aplikacji klienckiej Client.java odpowiadającej na pytanie jaki procent zaewidencjonowanych w bazie klientów nie posiada ważnego ubezpieczenia na wskazany model samochodu oraz dzień (np. Jaki procent klientów nie posiadało wykupionego ubezpieczenia samochodu marki 'Range Rover' w dniu '05/26/2014'). Poprawnie zaokrąglony wynik należy wyznaczyć w procentach z dokładnością do jednego miejsca dziesiętnego. W rozwiązaniu należało wykorzystać entity beans. Dla potrzeb zadania można było zaprojektować zgodną z potrzebami rozwiązania pewną ilość komponentów oraz niezbędnych interfejsów.

IGameRemote.java

Interfejs ***IGameRemote*** posiadający deklarację metod, która pozwalają na zarejestrowanie użytkownika w systemie.

Deklaracja metody ***register*** rejestrująca użytkownika w systemie - zwraca true jeżeli proces rejestracji zakończył się poprawnie. Jeżeli rejestracja zakończyła się niepowodzeniem, metoda register zwraca wartość false.

@param hwork - numer zadania

@param album – numer albumu studenta

@return wartość logiczna czy połączenie przebiegło pomyślnie

```
public boolean register(int hwork, String album);
```

Rysunek 1 Deklaracja metody register

Client.java

Klasa ***Client*** odpowiada na pytanie jaki procent zaewidencjonowanych w bazie klientów nie posiada ważnego ubezpieczenia na wskazany model samochodu oraz dzień. Wynik wyznaczono w procentach z dokładnością do jednego miejsca dziesiętnego. Pola marka i data są polami przechowującymi kolejno nazwę marki samochodu oraz dzień sprawdzenia.

```
public class Client {
    private String marka;
    private Date data;
```

Rysunek 2 Implementacja klasy Client

Metoda ***sprawdzenieKryterium*** pobiera wartości z bazy danych niezbędne do wyznaczenia wyniku określonego z warunków zadania. Pobierana jest ilość osób, które mają ważne ubezpieczenie. Ilość osób o nieważnym ubezpieczeniu określana jest przez różnicę wszystkich ludzi od osób z ważnym ubezpieczeniem.

@param klient - obiekt klient klasy Client

@param em - interfejs stosowany do współpracy z persistence context

@param ileKlientow - wartość liczbowa przechowująca ilość osób w bazie

@return Procent osób które nie posiadają ważnego ubezpieczenia.

```
private float sprawdzenieKryterium(Client klient, EntityManager em,
        long ileKlientow){
    long aktualne = 0;
    float k=0;

        aktualne = klient.pobierzIloscWCzasie(em);
        k=((float)(ileKlientow-aktualne)/(float)ileKlientow)*100;

    return k;
}
```

Rysunek 3 Implementacja metody sprawdzenieKryterium

Metoda ***polaczenieDb*** pozwalająca na połączenie się z bazą danych. Wywoływana w niej jest również metoda pobierająca ilość wszystkich użytkowników bazy.

@param klient - obiekt klient klasy Client

@return Procent osób które nie posiadają ważnego ubezpieczenia.

```
private float polaczenieDb(Client klient){
    EntityManagerFactory ef;
    EntityManager em;

    ef = Persistence.createEntityManagerFactory("persistence104896");
    em = ef.createEntityManager();
    long iloscOsob = klient.iloscKlientow(em);

    return sprawdzenieKryterium(klient,em,iloscOsob);
}
```

Rysunek 4 Implementacja metody polaczenieDb

Metoda ***iloscKlientow*** która wykonuje zapytanie na bazie danych i zwraca wartość liczbową wszystkich klientów

@param em - interfejs stosowany do współpracy z persistence context

@return Wartość liczbowa wszystkich klientów w bazie

```
private long iloscKlientow(EntityManager em) {
    Query allClients = em.createQuery(
        "SELECT COUNT(tbc) FROM TbCustomer tbc"
    );
    long wynik = (long)allClients.getSingleResult();

    return wynik;
}
```

Rysunek 5 Implementacja metody iloscKlientow

Metoda ***iloscKlientow*** która wykonuje zapytanie na bazie danych i wartość liczbową wszystkich klientów

@param em - interfejs stosowany do współpracy z persistence context

@return Wartość liczbowa wszystkich klientów w bazie

```
private long iloscKlientow(EntityManager em) {
    Query allClients = em.createQuery(
        "SELECT COUNT(tbc) FROM TbCustomer tbc"
    );
    long wynik = (long)allClients.getSingleResult();

    return wynik;
}
```

Rysunek 6 Implementacja metody iloscKlientow

Metoda ***pobierzIloscWCzasie*** wykonuje zapytanie na bazie danych, której wynikiem jest ilość osób, które mają aktualne ubezpieczenie.

@param em - interfejs stosowany do współpracy z persistence context

@return Wartość liczbowa wszystkich klientów w bazie o aktualnym ubezpieczeniu

```
private long pobierzIloscWCzasie(EntityManager em) {
    Query zapytanie = em.createQuery(
            "SELECT COUNT(tbi.customerId) " +
            "FROM TbInsurance tbi " +
            "JOIN TbModel tbm ON tbi.modelId=tbm.id " +
            "WHERE tbm.model = :markasam " +
            "AND :datasam BETWEEN tbi.dateFrom AND tbi.dateTo");

    zapytanie.setParameter("markasam", marka);
    zapytanie.setParameter("datasam",data);
    return (long) zapytanie.getSingleResult();
}
```

Rysunek 7 Implementacja metody pobierzIloscWCzasie

Metoda ***formatujDate*** pozwalająca na poprawne sformatowanie daty pobranej z bazy danych.

@param format1 - druga możliwość przedstawienia daty

@param format2 - trzecia możliwość przedstawienia daty

@param parser - pierwsza możliwość przedstawienia daty

@param dataString - wartość daty pobrana z pliku tekstowego

```
private void formatujDate(DateFormat format1, DateFormat format2,
        DateFormat parser, String dataString){

    try {
        data = parser.parse(dataString);
        if (data==null) {
            data = format1.parse(dataString);
        }else if (data==null) {
            data = format2.parse(dataString);
        }
    } catch (ParseException ex) {
        data= null;
    }


}
```

Rysunek 8 Implementacja metody formatujDate

Metoda ***pobierzZpliku*** pobierająca wartości z pliku i zapisująca je do zmiennych globalnych.

@param fileName - nazwa pliku potrzebna do jego otwarcia

@param dataString - zmienna pomocnicza pobierająca datę w postaci String

@param parser - parser określający przedstawienie daty

@throws FileNotFoundException Wyjątek na wypadek problemów przy otwarciu pliku

```
private void pobierzZpliku(String fileName, String dataString,
        DateFormat parser) throws FileNotFoundException{

    Scanner plik = new Scanner(new File(fileName));
    marka = plik.nextLine();
    dataString = plik.nextLine();

    formatujDate(
            new SimpleDateFormat("dd/MM/YYYY"),
            new SimpleDateFormat("MM/dd/YYYY"),
            parser, dataString
    );
}
```

Rysunek 9 Implementacja metody pobierzZpliku

Główna metota ***main*** w któej tworzony jest obiekt klasy Client i inicjowane jest połączenie w celu zarejestrowania użytkownika. Jeśli rejestracja przebiegnie pomyślnie to podejmowane są kroki do obliczenia procentowej ilości osób, które nie posiadają ważnego ubezpieczenia.

@param args - parametr wejściowy uruchamiany przy kompilacji

```
public static void main(String[] args) {
    Client klient = new Client();
    String connect = "java:global/ejb-project/GameManager!"
            +"pl.jrj.game.IGameRemote";

    try {
        InitialContext pol = new InitialContext();

        pl.jrj.game.IGameRemote game = (pl.jrj.game.IGameRemote)
                pol.lookup(connect);

        if (game.register(7, "104896")){
            klient.pobierzZpliku(
                    args[0],"",
                    DateFormat.getDateInstance(
                            DateFormat.DEFAULT,
                            Locale.getDefault()
                    )
            );
            System.out.format(Locale.US,
                    "%.1f%%\n", klient.polaczenieDb(klient));
        }
    } catch (NamingException | FileNotFoundException ex) {
        System.out.format(Locale.US, "0.0");
    } catch (Exception ex) {
        System.out.format(Locale.US, "1.0");

    }

}
```

Rysunek 10 Implementacja metody main

TbCustomer.java

Klasa ***TbCustomer*** mapująca tabelę tb_customer

```
@Entity
@Table(name = "tb_customer")
public class TbCustomer implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @NotNull
    @Column(name = "Id")
    private Integer id;
    @Size(max = 255)
    @Column(name = "firstName")
    private String firstName;
    @Size(max = 255)
    @Column(name = "lastName")
    private String lastName;
```

Rysunek 11Implementacja klasy TbCustomer

Konstruktor bezparametrowy ***TbCustomer***

```
public TbCustomer() {
}
```

Rysunek 12 Konstruktor bezargumentowy

Konstruktor jednoparametrowy przyjmujący wartość id

@param id - wartość id z tabeli tb_customer

```
public TbCustomer(Integer id) {
    this.id = id;
}
```

Rysunek 13 Konstruktor jednoargumentowy

Metoda ***getId*** będąca getterem dla id klienta

@return wartość id klienta

```
public Integer getId() {
    return id;
}
```

Rysunek 14 Implementacja metody getId

Metoda ***setId*** będąca setterem id klienta

@param id wartość id klienta

```
public void setId(Integer id) {
    this.id = id;
}
```

Rysunek 15 Implementacja metody setId

Metoda ***getFirstName*** będąca getterem dla imienia klienta

@return imię klienta

```
public String getFirstName() {
    return firstName;
}
```

Rysunek 16 Implementacja metody getFirstName

Metoda ***setFirstName*** będąca setterem dla imienia klienta

@param firstName imię klienta

```
public void setFirstName(String firstName) {
    this.firstName = firstName;
}
```

Rysunek 17 Implementacja metody setFirstName

Metoda ***getLastName*** będąca getterem dla nazwiska klienta

@return nazwisko klienta

```
public String getLastName() {
    return lastName;
}
```

Rysunek 18 Implementacja metody getLastName

Metoda ***setLastName*** będąca setterem dla nazwiska klienta

@param lastName nazwisko klienta

```
public void setLastName(String lastName) {
    this.lastName = lastName;
}
```

Rysunek 19 Implementacja metody setLastName

Metoda ***hashCode*** sprawdzająca zależność jeżeli wartość hashCode dla 2 obiektów jest taka sama, to obiekty te mogą być równoznaczne.

@return wartość (1/0 - true/false) jeśli obiekty są równe

```
@Override
public int hashCode() {
    int hash = 0;
    hash += (id != null ? id.hashCode() : 0);
    return hash;
}
```

Rysunek 20 Implementacja metody hashCode

Metoda ***equals*** jest mechanizmem pozwalającym porównać czy obiekty znaczą to samo niż czy są tym samym obiektem.

@param object - wartość do sprawdzenia

@return wartość true/false jeśli wartości zmiennych są równe.

```
@Override
public boolean equals(Object object) {
    if (!(object instanceof TbCustomer)) {
        return false;
    }
    TbCustomer other = (TbCustomer) object;
    if ((this.id == null && other.id != null) ||
            (this.id != null && !this.id.equals(other.id))) {
        return false;
    }
    return true;
}
```

Rysunek 21 Implementacja metody equals

Metoda ***toString*** wywoływana automatycznie, kiedy obiekt ma zostać przedstawiony jako wartość tekstowa.

@return wartość tekstowa z obiektu.

```
@Override
public String toString() {
    return "p.TbCustomer[ id=" + id + " ]";
}
```

Rysunek 22 Implementacja metody toString

TbInsurance.java

Klasa ***TbInsurance*** mapująca tabelę tb_insurance

```
@Entity
@Table(name = "tb_insurance")
public class TbInsurance implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @NotNull
    @Column(name = "Id")
    private Integer id;
    @Column(name = "customerId")
    private Integer customerId;
    @Column(name = "modelId")
    private Integer modelId;
    @Column(name = "dateFrom")
    @Temporal(TemporalType.TIMESTAMP)
    private Date dateFrom;
    @Column(name = "dateTo")
    @Temporal(TemporalType.TIMESTAMP)
    private Date dateTo;
```

Rysunek 23 Implementacja klasy TbInsurane

Konstruktor bezargumentowy ***TbInsurance***

```
public TbInsurance() {
}
```

Rysunek 24 Konstruktor bezargumentowy

Konstruktor jednoargumentowy ***TbInsurance***

@param id - wartość id z tabeli tb_insurance

```
public TbInsurance(Integer id) {
    this.id = id;
}
```

Rysunek 25 Konstruktor jednoargumentowy

Metoda ***getId*** będąca getterem dla kolumny Id

@return wartość Id

```
public Integer getId() {
    return id;
}
```

Rysunek 26 Implementacja metody getId

Metoda ***setId*** będąca setterem dla kolumny Id

@param id wartość Id

```
public void setId(Integer id) {
    this.id = id;
}
```

Rysunek 27 Implementacja metody setId

Metoda ***getCustomerId*** będąca getterem dla kolumny customerId

@return wartość customerId

```
public Integer getCustomerId() {
    return customerId;
}
```

Rysunek 28 Implementacja metody getCustomerId

Metoda ***setCustomerId*** będąca setterem dla kolumny customerId

@param customerId wartość customerId

```
public void setCustomerId(Integer customerId) {
    this.customerId = customerId;
}
```

Rysunek 29 Implementacja metody setCustomerId

Metoda ***getModelId*** będąca getterem dla kolumny modelId

@return wartość modelId

```
public Integer getModelId() {
    return modelId;
}
```

Rysunek 30 Implementacja metody getModelId

Metoda ***setModelId*** będąca setterem dla kolumny modelId

@param modelId wartość modelId

```
public void setModelId(Integer modelId) {
    this.modelId = modelId;
}
```

Rysunek 31 Implementacja metody setModelId

Metoda ***getDateFrom*** będąca getterem dla kolumny dateFrom

@return Wartość dateFrom

```
public Date getDateFrom() {
    return dateFrom;
}
```

Rysunek 32 Implementacja metody getDateFrom

Metoda ***setDateFrom*** będąca setterem dla kolumny dateFrom

@param dateFrom Wartość dateFrom

```
public void setDateFrom(Date dateFrom) {
    this.dateFrom = dateFrom;
}
```

Rysunek 33 Implementacja metody setDateFrom

Metoda ***getDateTo*** będąca getterem dla kolumny dateTo

@return Wartość dateTo

```
public Date getDateTo() {
    return dateTo;
}
```

Rysunek 34 Implementacja metody getDateTo

Metoda ***setDateTo*** będąca setterem dla kolumny dateTo

@param dateTo Wartość dateTo

```
public void setDateTo(Date dateTo) {
    this.dateTo = dateTo;
}
```

Rysunek 35 Implementacja metody setDateTo

Metoda ***hashCode*** sprawdzająca zależność jeżeli wartość hashCode dla 2 obiektów jest taka sama, to obiekty te mogą być równoznaczne.

@return wartość (1/0 - true/false) jeśli obiekty są równe

```
@Override
public int hashCode() {
    int hash = 0;
    hash += (id != null ? id.hashCode() : 0);
    return hash;
}
```

Rysunek 36 Implementacja metody hashCode

Metoda ***equals*** jest mechanizmem pozwalającym porównać czy obiekty znaczą to samo niż czy są tym samym obiektem.

@param object - wartość do sprawdzenia

@return wartość true/false jeśli wartości zmiennych są równe.

```
@Override
public boolean equals(Object object) {
    if (!(object instanceof TbInsurance)) {
        return false;
    }
    TbInsurance other = (TbInsurance) object;
    if ((this.id == null && other.id != null) ||
            (this.id != null && !this.id.equals(other.id))) {
        return false;
    }
    return true;
}
```

Rysunek 37 Implementacja metody equals

Metoda ***toString*** wywoływana automatycznie, kiedy obiekt ma zostać przedstawiony jako wartość tekstowa.

@return wartość tekstowa z obiektu.

```
@Override
public String toString() {
    return "p.TbInsurance[ id=" + id + " ]";
}
```

Rysunek 38 Implementacja metody toString

TbModel.java

Klasa ***TbModel*** mapująca tabelę tb_model

```
@Entity
@Table(name = "tb_model")
public class TbModel implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @NotNull
    @Column(name = "Id")
    private Integer id;
    @Size(max = 255)
    @Column(name = "model")
    private String model;
```

Rysunek 39 Implementacja klasy TbModel

Konstruktor bezargumentowy ***TbModel***

```
public TbModel() {
}
```

Rysunek 40 Konstruktor bezargumentowy

Konstruktor jednoargumentowy ***TbModel***

@param id - wartość id z tabeli tb_model

```
public TbModel(Integer id) {
    this.id = id;
}
```

Rysunek 41 Konstruktor jednoargumentowy

Metoda ***getId*** będąca getterem dla kolumny Id

@return wartość Id

```
public Integer getId() {
    return id;
}
```

Rysunek 42 Implementacja metody getId

Metoda ***setId*** będąca setterem dla kolumny Id

@param id wartość Id

```
public void setId(Integer id) {
    this.id = id;
}
```

Rysunek 43 Implementacja metody setId

Metoda ***getModel*** będąca getterem dla kolumny model

@return wartość model

```
public String getModel() {
    return model;
}
```

Rysunek 44 Implementacja metody getModel

Metoda ***setModel*** będąca setterem dla kolumny model

@param model wartość model

```
public void setModel(String model) {
    this.model = model;
}
```

Rysunek 45 Implementacja metody setModel

Metoda ***hashCode*** sprawdzająca zależność jeżeli wartość hashCode dla 2 obiektów jest taka sama, to obiekty te mogą być równoznaczne.

@return wartość (1/0 - true/false) jeśli obiekty są równe

```
@Override
public int hashCode() {
    int hash = 0;
    hash += (id != null ? id.hashCode() : 0);
    return hash;
}
```

Rysunek 46 Implementacja metody hashCode

Metoda ***equals*** jest mechanizmem pozwalającym porównać czy obiekty znaczą to samo niż czy są tym samym obiektem.

@param object - wartość do sprawdzenia

@return wartość true/false jeśli wartości zmiennych są równe.

```
@Override
public boolean equals(Object object) {
    if (!(object instanceof TbModel)) {
        return false;
    }
    TbModel other = (TbModel) object;
    if ((this.id == null && other.id != null) ||
            (this.id != null && !this.id.equals(other.id))) {
        return false;
    }
    return true;
}
```

Rysunek 47 Implementacja metody equals

Metoda ***toString*** wywoływana automatycznie, kiedy obiekt ma zostać przedstawiony jako wartość tekstowa.

@return wartość tekstowa z obiektu.

```
@Override
public String toString() {
    return "p.TbModel[ id=" + id + " ]";
}
```

Rysunek 48 Implementacja metody toString

Mechanizm operowania danymi oraz algorytm wyznaczania wyniku

Program na samym początku rejestruje użytkownika w systemie przy użyciu InitialContext. Po przejściu poprawnej weryfikacji program pobiera parametr będący nazwą pliku z którego odczytywana jest marka samochodu oraz data sprawdzenia ważności ubezpieczenia. Te dwa parametry zapisane są jako zmienne globalne (pola klasy). Kolejnym krokiem jest połączenie się z bazą danych w metodzie *polaczenieDb.* Połączenie odbywa się za pomocą nazwy JDBC resources zapisanego w pliku persistence.xml w znaczniku *jta-data-source*. Wcześniej jednak zostały stworzone klasy mapujące tabele baz danych (TbCustomer.java, TbInsurance.java oraz TbModel.java) – są one powiązane z entity beans. Kolejnym krokiem po stworzeniu EntityManager'a wykonuje zapytanie zliczające ilość wszystkich klientów w tabeli tb_customer. W zapytaniu nie została napisana nazwa tabeli z której mają zostać zliczone dane, a nazwa klasy mapującej. Wykonanie zapytania odbyło się przy użyciu metody createQuery która tworzy instancję zapytania do wykonywania instrukcji zapytania Java Persistence. Kolejnym krokiem było wyznaczenie ilości osób które mają aktualne ubezpieczenie w dniu przekazanym przez parametr. Program, po wyznaczeniu takiej liczby, wyznacza ilość osób o nie aktualnym ubezpieczeniu poprzez różnicę wszystkich klientów i klientów którzy mają aktualne ubezpieczenie. Tak otrzymane dane dzielone są przez ilość wszystkich klientów i mnożone przez 100. Zwrócony wynik wypisywany jest jako wartość procentowa.