Кафедра вычислительной техники



**ЛАБОРАТОРНАЯ РАБОТА №7**

**по дисциплине:** *технология программирования*

**на тему:** *Организация в Java хранения данных в базе данных.*

|  |  |
| --- | --- |
| Выполнил: | Проверил: |
| Студент гр. *АВТ-808*, *АВТФ* | *ассистент каф. ВТ* |
| *Горбанев В.В.* | *Михайленко Дмитрий Анатольевич* |
| «\_\_\_» \_\_\_\_\_\_ 20\_\_г. | «\_\_\_» \_\_\_\_\_\_ 20\_\_г. |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| (подпись) | (подпись) |

# **ЦЕЛЬ И ЗАДАНИЕ**

1. Изучить особенности реализации хранения данных в базе данных в приложениях Java.
2. Доработать программу, созданную в лабораторной работе № 5:

**Вариант 11**

* Добавить в меню пункты сохранение и загрузку объектов в базу данных. Можно использовать любую СУБД.
* Загрузку и сохранение объектов только заданного типа.

# **Ход работы**

В ходе лабораторной работе была реализована база данных MySQL.

Был создан класс:

public class DataBase

который реализовывает весь функционал связанный с базой данных.

В этом классе содержаться поля:

private static Connection *connection*;  
private static Statement *statement*;  
private static ResultSet *resultSet*;

connection – служит для подключения к базе данных;

statement – служит для выполнения запросов на сервер и отправку нужных данных;

resultSet – служит для считывания данных после того, как statement отправил запрос на их получения;

Он имеет такие методы как:

public static void connect()

public static void create()

public static void saveALL()

public static void saveCats()

public static void saveDogs()

public static void downloadALL(int time)

public static void downloadCats(int time)

public static void downloadDogs(int time)

Метод connect() подсоединяет клинет к базе данных:

public static void connect() throws ClassNotFoundException, SQLException {  
 final String URL = "jdbc:mysql://localhost:3306/database?autoReconnect=true&useSSL=false";  
 Class.*forName*("com.mysql.jdbc.Driver");  
 *connection* = DriverManager.*getConnection*(URL, "root", "root");  
}

Метод Class.forName("com.mysql.jdbc.Driver"); проверяет есть ли нужный драйвер для работы с базами данных.

Метод DriverManager.getConnection(URL, "root", "root"); осуществляет подключения к работающей базе данных;

Метод create() создает таблицу, где будут хранится питомцы пользователя:

public static void create() throws SQLException {  
 *statement* = *connection*.createStatement();  
 *statement*.executeUpdate("drop table pets");  
 *statement*.executeUpdate("create table pets(" +  
 "ID int not null auto\_increment" +  
 ",type int not null" +  
 ",x int not null" +  
 ",y int not null" +  
 ",idPet int not null" +  
 ",primary key(id)" +  
 ");");  
}

Методы saveAll() записывает из массива хранящегося в singltone в таблице MySQL:

public static void saveALL() throws SQLException {  
 int how = 0;  
 for (int i = 0;i<Singleton.*getInstance*().sizeArray();i++){  
 if(Singleton.*getInstance*().Get(i) instanceof Cat){  
 how = 1;  
 }else {  
 how = 2;  
 }  
 int petX = Singleton.*getInstance*().Get(i).getX();  
 int petY = Singleton.*getInstance*().Get(i).getY();  
 int id = Singleton.*getInstance*().Get(i).getID();  
 String petsSettings = "insert into pets (type,x,y,idPet) values ("+how+","+petX+","+petY+","+id+")";  
 *statement*.executeUpdate(petsSettings);  
 }  
}

Методы downloadAll() считывает из таблицы MySQL в массив клиента, предварительно отчистив его:

public static void downloadALL(int time) throws SQLException {  
 Cat myCat;  
 Dog myDog;  
 *resultSet* = *statement*.executeQuery("select \* from pets;");  
 Singleton.*getInstance*().Clear();  
 Singleton.*getInstance*().getWindow().quantityPet=0;  
 while(*resultSet*.next()){  
 if(*resultSet*.getInt(2) == 1) {  
 myCat = new Cat(*resultSet*.getInt(4),time);  
 myCat.setX(*resultSet*.getInt(3));  
 myCat.setY(*resultSet*.getInt(4));  
 Singleton.*getInstance*().getArray().add(myCat);  
 Singleton.*getInstance*().getWindow().quantityPet++;  
 }else {  
 myDog = new Dog(*resultSet*.getInt(4),time);  
 myDog.setX(*resultSet*.getInt(3));  
 myDog.setY(*resultSet*.getInt(4));  
 Singleton.*getInstance*().getArray().add(myDog);  
 Singleton.*getInstance*().getWindow().quantityPet++;  
 }  
 }  
}

Методы saveCats() и saveDogs() сохраняют в таблицу соответствующий тип питомца:

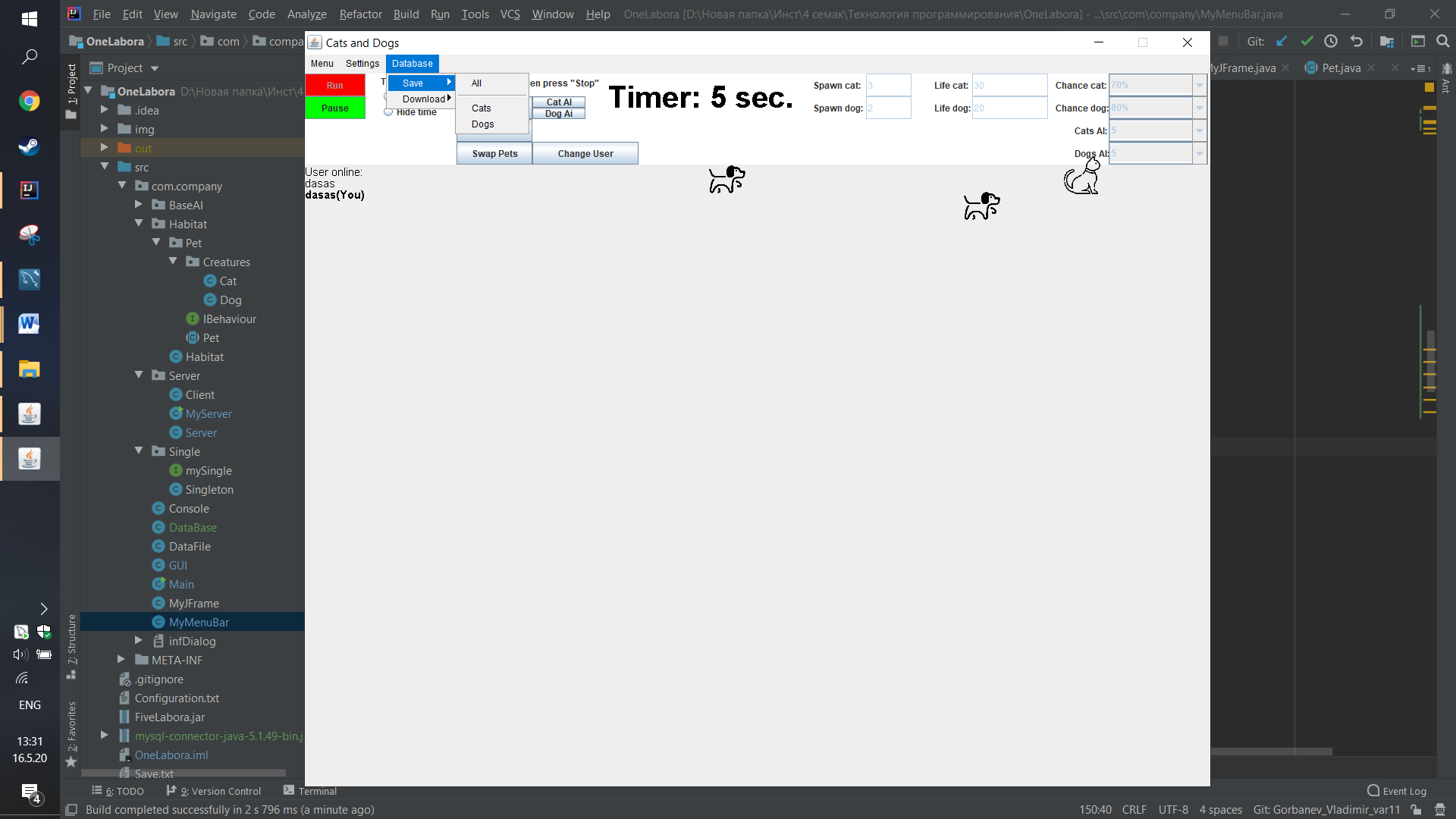
public static void saveCats() throws SQLException {  
 int how = 0;  
 for (int i = 0;i<Singleton.*getInstance*().sizeArray();i++){  
 if(Singleton.*getInstance*().Get(i) instanceof Cat){  
 how = 1;  
 int petX = Singleton.*getInstance*().Get(i).getX();  
 int petY = Singleton.*getInstance*().Get(i).getY();  
 int id = Singleton.*getInstance*().Get(i).getID();  
 String petsSettings = "insert into pets (type,x,y,idPet) values ("+how+","+petX+","+petY+","+id+")";  
 *statement*.executeUpdate(petsSettings);  
 }  
 }  
}  
public static void saveDogs() throws SQLException {  
 int how = 0;  
 for (int i = 0;i<Singleton.*getInstance*().sizeArray();i++){  
 if(Singleton.*getInstance*().Get(i) instanceof Cat){  
 how = 2;  
 int petX = Singleton.*getInstance*().Get(i).getX();  
 int petY = Singleton.*getInstance*().Get(i).getY();  
 int id = Singleton.*getInstance*().Get(i).getID();  
 String petsSettings = "insert into pets (type,x,y,idPet) values ("+how+","+petX+","+petY+","+id+")";  
 *statement*.executeUpdate(petsSettings);  
 }  
 }  
}

Методы downloadCats() и downloadCats ()загружают из таблицу соответствующий тип питомца, при этом отчищая нынешний массив питомцев у пользователя:

public static void downloadCats(int time) throws SQLException {  
 Cat myCat;  
 *resultSet* = *statement*.executeQuery("select \* from pets where type=1;");  
 Singleton.*getInstance*().Clear();  
 Singleton.*getInstance*().getWindow().quantityPet=0;  
 while(*resultSet*.next()){  
 myCat = new Cat(*resultSet*.getInt(4),time);  
 myCat.setX(*resultSet*.getInt(3));  
 myCat.setY(*resultSet*.getInt(4));  
 Singleton.*getInstance*().getArray().add(myCat);  
 Singleton.*getInstance*().getWindow().quantityPet++;  
 }  
}  
public static void downloadDogs(int time) throws SQLException {  
 Dog myDog;  
 *resultSet* = *statement*.executeQuery("select \* from pets where type=2;");  
 Singleton.*getInstance*().Clear();  
 Singleton.*getInstance*().getWindow().quantityPet=0;  
 while(*resultSet*.next()){  
 myDog = new Dog(*resultSet*.getInt(4),time);  
 myDog.setX(*resultSet*.getInt(3));  
 myDog.setY(*resultSet*.getInt(4));  
 Singleton.*getInstance*().getArray().add(myDog);  
 Singleton.*getInstance*().getWindow().quantityPet++;  
 }  
}

Для использования этих методов в менюбаре программы был добавлен пункт Database:

Menu downloadInData = new Menu("Download");  
MenuItem downloadAll = new MenuItem("All");  
MenuItem downloadCats = new MenuItem("Cats");  
MenuItem downloadDogs = new MenuItem("Dogs");  
downloadInData.add(downloadAll);  
downloadInData.add(new MenuItem("-"));  
downloadInData.add(downloadCats);  
downloadInData.add(downloadDogs);  
Database.add(saveInData);  
Database.add(downloadInData);  
  
saveAll.addActionListener(e -> {  
 try {  
 base.*saveALL*();  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
});  
saveCats.addActionListener(e -> {  
 try {  
 base.*saveCats*();  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
});  
saveDogs.addActionListener(e -> {  
 try {  
 base.*saveDogs*();  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
});  
downloadAll.addActionListener(e -> {  
 try {  
 base.*downloadALL*(myGUI.myTimer);  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
});  
downloadCats.addActionListener(e -> {  
 try {  
 base.*downloadCats*(myGUI.myTimer);  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
});  
downloadDogs.addActionListener(e -> {  
 try {  
 base.*downloadDogs*(myGUI.myTimer);  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
});



*Рис. 1. Результат выполнения метода кнопки «start».*

# Вывод:

В ходе лабораторной работы были изучить особенности реализации хранения данных в базе данных в приложениях Java.