```java
package server.managers;

import java.io.IOException;
import java.io.InputStream;
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Objects;
import java.util.Properties;

/**
 * Класс {@code DatabaseManager} обеспечивает доступ к базе данных PostgreSQL.studs.
 * @author Артемий Кульбако
 * @version 1.1
 * @since 01.06.19
 */
public final class DatabaseManager {

    private String DB_URL;
    private String USER;
    private String PASSWORD;
    private static DatabaseManager instance;

    {
        try (InputStream inputStream = this.getClass().getClassLoader().getResourceAsStream("helios_db.properties")) {
            Properties prop = System.getProperties();
            prop.load(inputStream);
            DB_URL = prop.getProperty("url_address");
            USER = prop.getProperty("user");
            PASSWORD = prop.getProperty("password");
            System.out.println("База данных сконфигурирована с helios_db.properties.");
        } catch (IOException | NullPointerException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }

    /**
     * Предоставляет доступ к БД. БД должна быть одна на приложение, поэтому реализован singleton шаблон.
     */
    public static DatabaseManager getInstance() {
        if (instance == null) {
            instance = new DatabaseManager();
        }
        return instance;
    }

    /**
     * Инициализирует БД и осуществляет пробное подключение к ней.
     */
    private DatabaseManager() {
        try (Connection testConnection = DriverManager.getConnection(DB_URL, USER, PASSWORD);
             ResultSet testRequest = testConnection.createStatement().executeQuery("SELECT version()")) {
            System.out.println("Идёт установка связи с БД.");
            while (testRequest.next())
                System.out.println("Связь с БД установлена." + " Версия: " + testRequest.getString(1));
        } catch (SQLException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }

    /**
     * Возвращает новое соединение с БД.
     * @return объект {@link Connection} для связи.
     * @throws SQLException
     */
```

```
69      public Connection getConnection() throws SQLException {
70          return DriverManager.getConnection(DB_URL, USER, PASSWORD);
71      }
72
73      public String getDB_URL() {
74          return DB_URL;
75      }
76
77      public String getUSER() {
78          return USER;
79      }
80
81      public String getPASSWORD() {
82          return PASSWORD;
83      }
84
85      @Override
86      public boolean equals(Object o) {
87          if (this == o) return true;
88          if (!(o instanceof DatabaseManager)) return false;
89          DatabaseManager manager = (DatabaseManager) o;
90          return Objects.equals(DB_URL, manager.DB_URL) &&
91                  Objects.equals(USER, manager.USER) &&
92                  Objects.equals(PASSWORD, manager.PASSWORD);
93      }
94
95      @Override
96      public int hashCode() {
97          return Objects.hash(DB_URL, USER, PASSWORD);
98      }
99
100     @Override
101     public String toString() {
102         return "DatabaseManager{" +
103                 "DB_URL='" + DB_URL + '\'' +
104                 ", USER='" + USER + '\'' +
105                 ", PASSWORD='" + PASSWORD + '\'' +
106                 '}';
107     }
108 }
```

```java
package server.managers;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;

/**
 * Класс {@code PasswordManager} содержит статические методы для работы с паролями.
 * @author Артемий Кульбако
 * @version 1.2
 * @since 02.06.19
 */
public class PasswordManager {

    /**
     * Генерирует новый 8-ми значный пароль, содержащий прописные и строчные символы латинского алфавита и цифры.
     * За генерацию символов отвечает {@link java.security.SecureRandom}.
     * @return новый пароль.
     */
    public static String generateNewPassword() {
        String chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
        SecureRandom random = new SecureRandom();
        StringBuilder password = new StringBuilder();
        for (int i = 0; i < 8; i++) password.append(chars.charAt(random.nextInt(chars.length())));
        System.out.println("Новый пароль сгенерирован.");
        return password.toString();
    }


    /**
     * Получает хэш-последовательность для переданной строки, согласно указанному алгоритму (соль не добавляется).
     * @param nudeString строка, для которой необходимо получить хэш.
     * @param algorithm алгоритм хэширования.
     * @return хэш-последовательность.
     * @throws NoSuchAlgorithmException если алгоритм не найден.
     */
    public static String getHash(String nudeString, String algorithm) throws NoSuchAlgorithmException {
            MessageDigest mDigest = MessageDigest.getInstance(algorithm);
            byte[] hash = mDigest.digest(nudeString.getBytes());
            StringBuilder strongPassword = new StringBuilder();
            for (byte b : hash) strongPassword.append(Integer.toString((b & 0xff) + 0x100, 16).substring(1));
            System.out.println("Получена hash-последовательность.");
            return strongPassword.toString();
    }
}

```

```java
package server.managers;

import com.google.gson.*;
import tale.Coordinates;
import tale.Shorty;
import com.google.gson.reflect.TypeToken;
import tale.WalletBalance;
import java.lang.reflect.Type;
import java.sql.*;
import java.time.OffsetDateTime;
import java.util.*;
import java.util.Date;

/**
 * Класс {@code CollectionManager} обеспечивает доступ к коллекции.
 * @author Артемий Кульбако
 * @version 2.6
 * @since 21.05.19
 */
public final class CollectionManager {

    private List<Shorty> citizens;
    private Gson serializer;
    private Type collectionType;
    private Date initDate;
    private boolean collInit;
    private static CollectionManager instance;

    {
        citizens = Collections.synchronizedList(new LinkedList<>());
        serializer = new Gson();
        collectionType = new TypeToken<LinkedList<Shorty>>() {}.getType();
        collInit = false; //метка, сигнализирущая о статусе коллекции
    }

    /**
     * Предоставляет доступ к коллекции и связанному с ней файлу. Коллекция должна быть одна на приложение, поэтому
     * реализован singleton шаблон.
     */
    public static CollectionManager getInstance() {
        if (instance == null) {
            instance = new CollectionManager();
        }
        return instance;
    }

    /**
     * Загружает коллекцию из базы данных.
     */
    private CollectionManager() {
        System.out.println("Инициализация коллекции " + DatabaseManager.getInstance().getDB_URL());
        System.out.println(load());
        collInit = true;
        initDate = new Date();
    }

    /**
     * Записывает элементы коллекции в базу данных. Так как необходим нескольким командам, реализован в этом классе.
     * @return результат операции
     */
    public String save() {
        DatabaseManager manager = DatabaseManager.getInstance();
        try (Statement clearTable = manager.getConnection().createStatement()) {
            clearTable.executeLargeUpdate("DELETE FROM shortys");
            for (Shorty s: citizens) {
                try (PreparedStatement fillTable = manager.getConnection().
                        prepareStatement("INSERT INTO shortys VALUES (?, ?, ?, ?, ?, ?, ?)")) {
                    fillTable.setString(1, s.getName());
                    fillTable.setDouble(2, s.getWallet().getMoney());
```

```
70                      fillTable.setInt(3, s.getWallet().getStocks());
71                      fillTable.setDouble(4, s.getCoordinates().getX());
72                      fillTable.setDouble(5, s.getCoordinates().getY());
73                      fillTable.setString(6, serializer.toJson(s.getBirthDate()));
74                      fillTable.setInt(7, s.getMasterID());
75                      fillTable.executeUpdate();
76                  }
77              }
78              return "Изменения успешно сохранены.";
79          } catch (SQLException e) {
80              e.printStackTrace();
81              return "Не удалось сохранить изменения из-за ошибки на сервере. Попробуйте ещё раз позже.";
82          }
83      }
84
85      /**
86       * Заполняет коллекцию данными из БД. Если коллекция загружается впервые, и возникает исключение, то метод выводит
87       * стек трассировки и завершает работу программы, при возникновении исключений во время последующих загрузок только
88       * выводится стек трассировки.
89       * @return результат загружки коллекции.
90       */
91      public String load() {
92          try (ResultSet answer = DatabaseManager.getInstance().getConnection().createStatement().
93                  executeQuery("SELECT * FROM shortys")) {
94              citizens.clear();
95              while (answer.next()) {
96                  String name = answer.getString("name");
97                  WalletBalance cash = new WalletBalance(answer.getDouble("cash_sum"), answer.getInt("cash_amount"));
98                  Coordinates position = new Coordinates(answer.getDouble("position_x"), answer.getDouble("position_y"));
99                  OffsetDateTime birthDate = serializer.fromJson(answer.getString("birthDate"), OffsetDateTime.class);
100                 int masterID = answer.getInt("masterID");
101                 citizens.add(new Shorty(name, cash, position, birthDate, masterID));
102             }
103             return "Коллекция загружена.";
104         } catch (SQLException | JsonSyntaxException e) {
105             e.printStackTrace();
106             if (!collInit) System.exit(1);
107             return "Возникла непредвиденная ошибка. Коллекция не может быть загружена сейчас. Попробуйте позже.";
108         }
109     }
110
111     public List<Shorty> getCitizens() {
112         return citizens;
113     }
114
115     public Gson getSerializer() {
116         return serializer;
117     }
118
119     public Type getCollectionType() {
120         return collectionType;
121     }
122
123     @Override
124     public String toString() {
125         return "Тип коллекции: " + citizens.getClass() +
126                 "\nТип элементов: " + Shorty.class +
127                 "\nДата инициализации: " + initDate +
128                 "\nКоличество элементов: " + citizens.size();
129     }
130
131     @Override
132     public boolean equals(Object o) {
133         if (this == o) return true;
134         if (!(o instanceof CollectionManager)) return false;
```

```
135         CollectionManager that = (CollectionManager) o;
136         return Objects.equals(citizens, that.citizens) &&
137                 Objects.equals(serializer, that.serializer) &&
138                 Objects.equals(collectionType, that.collectionType);
139     }
140 
141     @Override
142     public int hashCode() {
143         return Objects.hash(citizens, serializer, collectionType);
144     }
145 }
```