

Properties

Properties class

- [Документація](#)
- Клас Properties у Java є спеціалізованим підкласом Hashtable.
- Дані зберігаються у вигляді пар ключ-значення, де і ключі, і значення є **String**.
- Використовується для обробки даних конфігурації, таких як параметри або файли конфігурації.

Основні методи Properties класу

- **load**(InputStream inStream): завантажує пари ключ-значення з файлу (або потоку) в об'єкт Properties.
- **store**(OutputStream out, String comments): записує пари ключ-значення з об'єкта Properties у файл.
- **getProperty**(String key): повертає значення, пов'язане з указаним ключем.
- **getProperty**(String key, String defaultValue): повертає значення, пов'язане з указаним ключем або **defaultValue** якщо немає значення.
- **setProperty**(String key, String value): встановлює пару ключ-значення в об'єкті Properties

Приклад файлу конфігурації

```
# Database configuration  
database.url=jdbc:mysql://localhost:3306/mydb  
username=myuser  
password=mypassword
```

```
public static Properties readConfiguration(String filename) {  
    Properties properties = new Properties();  
    try {  
        properties.load(new FileInputStream(filename));  
    } catch (IOException e) {  
        System.out.println("Error! " + e.getMessage());  
    }  
    return properties;  
}
```

```
public static void main(String[] args) {  
    String configFilename = "./resources/config.properties";  
    Properties properties = readConfiguration(configFilename);  
    System.out.println(properties);  
}
```

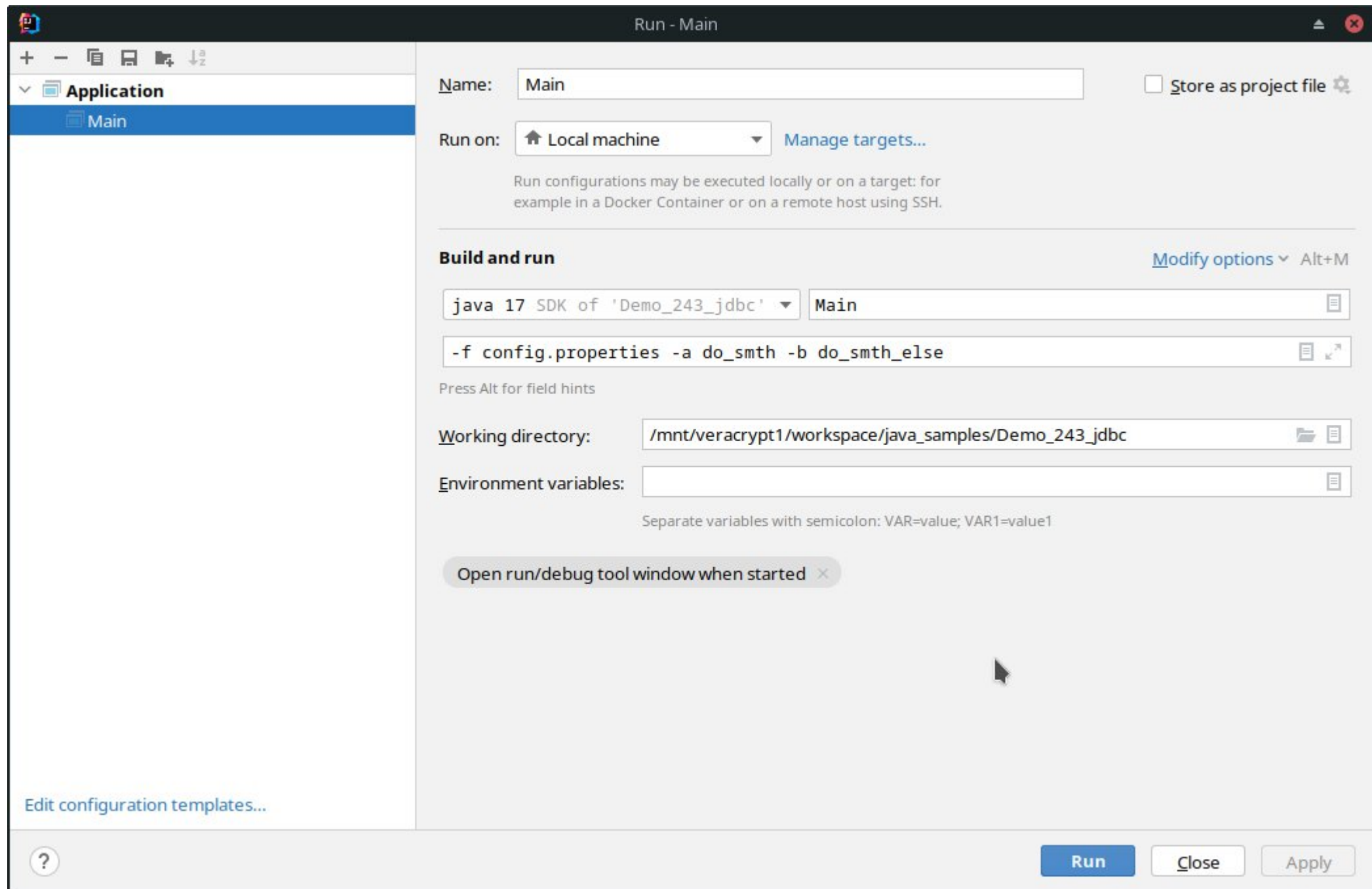
```
{password=mypassword, database.url=jdbc:mysql://localhost:3306/mydb, username=myuser}
```

Варіанти конфігурування програми

- Параметри запуску;
- Змінні середовища;
- Конфігураційні файли

Параметри запуску

- Параметри запуску Java, (аргументи командного рядка або аргументи JVM), — це параметри та прапорці, які можна передати віртуальній машині Java (JVM) під час запуску програми Java.
- Використовуються для керування різними аспектами JVM і запусненою на ній програмою.




```
public static void main(String[] args) {  
    for (String arg: args) {  
        System.out.println(arg);  
    }  
}
```



The screenshot shows the 'Run' console of an IDE. At the top, there is a tab labeled 'Run: Main x'. Below the tab, a vertical toolbar on the left contains icons for running (a green play button), debugging (a wrench), and other actions. The main area of the console displays the command used to run the program: `/usr/lib/jvm/java-17-openjdk/bin/java -javaage`. Below the command, the arguments `-f`, `config.properties`, `-a`, `do_smth`, `-b`, and `do_smth_else` are listed. At the bottom of the console, the message `Process finished with exit code 0` is displayed.

```
Run: Main x  
/usr/lib/jvm/java-17-openjdk/bin/java -javaage  
-f  
config.properties  
-a  
do_smth  
-b  
do_smth_else  
  
Process finished with exit code 0
```

```
public class LaunchParametersParser {
    private final Map<String, String> config = new HashMap<>();

    public LaunchParametersParser(String[] args) {
        mapParametersIntoMap(args);
    }
    private void mapParametersIntoMap(String[] args) {
        for (int i = 0; i < args.length; i += 2) {
            if (args[i].contains("-") && i + 1 < args.length) {
                config.put(extractFlags(args[i]), args[i + 1]);
            }
        }
    }
    private String extractFlags(String flag) {
        return flag.replaceAll("[^A-Za-z]+", "");
    }
    public Map<String, String> getConfig() {
        return this.config;
    }
    public String getValue(String key) {
        return this.config.get(key);
    }
}
```

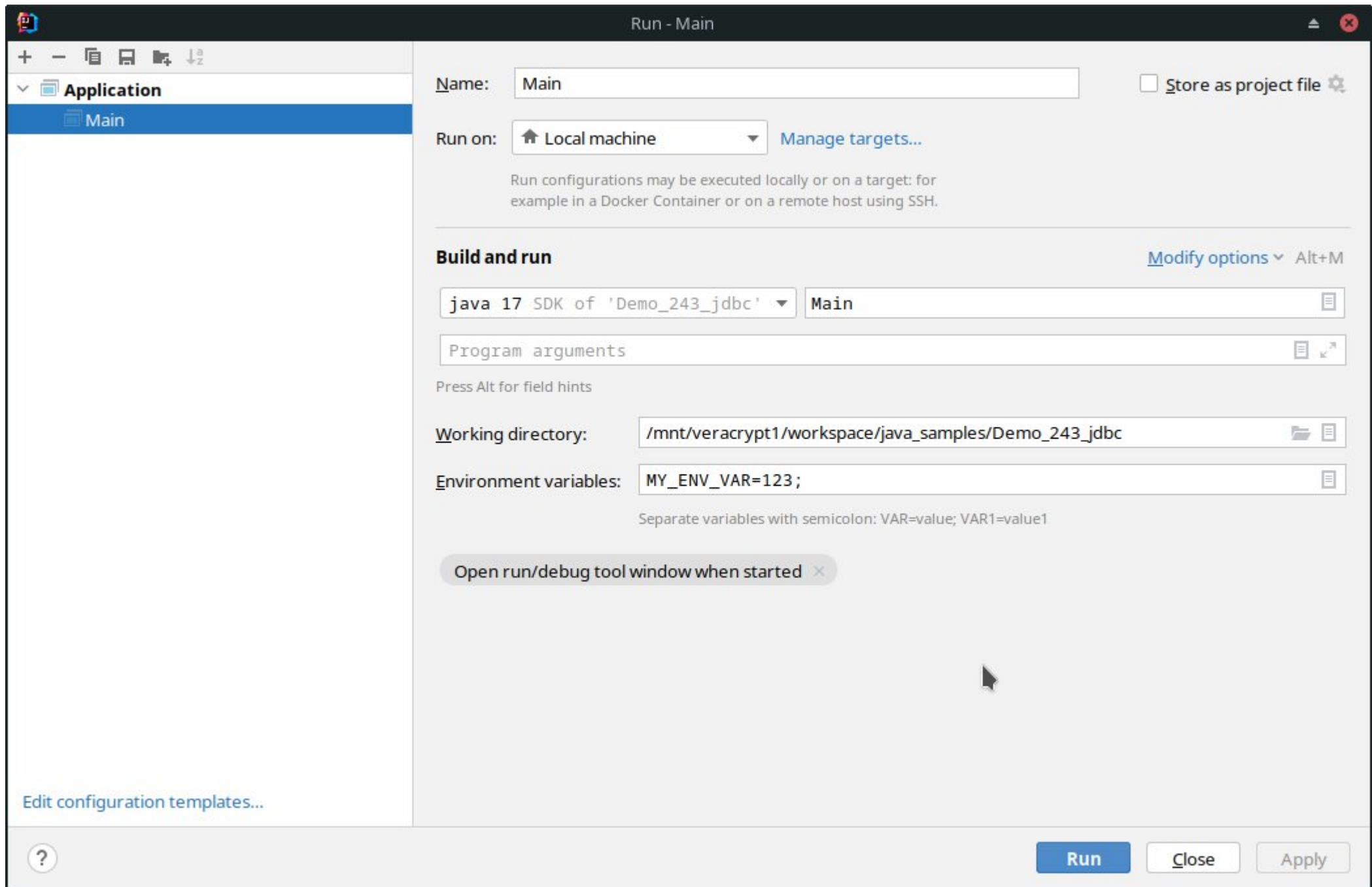
```
public class AppConfig {  
    private static final String CONFIG_FILE = "config.properties";  
    private static Properties properties;  
  
    public AppConfig() {  
        loadConfig("");  
    }  
  
    public AppConfig(String configFile) {  
        loadConfig(configFile);  
    }  
  
    private void loadConfig(String configFile) {  
        properties = new Properties();  
        try {  
            if (configFile == null || configFile.equals("")) {  
                properties.load(new FileInputStream(CONFIG_FILE));  
            } else {  
                properties.load(new FileInputStream(configFile));  
            }  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

Environment parameters

- Значення, яке може впливати на поведінку запускених процесів на комп'ютері.
- Є частиною середовища, в якому виконується процес.

Environment variables приклади

- PATH
- **/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games**
- HOME (Unix-like systems) or USERPROFILE (Windows)
- **/home/oleksandrval**
- JAVA_HOME
- DATABASE_URL
- API_KEY



Використання змінних середовища

```
public class Main {  
  
    public static void main(String[] args) {  
        System.out.println("My env var is: " + System.getenv("MY_ENV_VAR"));  
    }  
}
```



```
Run: Main x  
/usr/lib/jvm/java-17-openjdk/bin/java -javaagent:/opt/jetbrains/  
My env var is: 123  
  
Process finished with exit code 0
```

Використання параметрів запуску

```
public class Main {  
    public static void printUsage() {  
        System.out.println("Usage: ");  
        System.out.println("Parameters required: ");  
        System.out.println("1) Path to .properties file");  
    }  
    public static String getPropertiesFileName(String[] args) {  
        return args[0];  
    }  
    public static void main(String[] args) throws SQLException {  
        if (args.length < 1) {  
            printUsage();  
            return;  
        }  
        String propertiesFileName = getPropertiesFileName(args);  
        Properties properties = PropertiesWrapper.loadProperties(propertiesFileName);  
        System.out.println(properties);  
    }  
}
```


Використання параметрів запуску

```
Properties properties = readConfiguration(configFilename);  
String password = properties.getProperty("password");  
String defaultPassword = properties.getProperty("cassandra_password", "123456");  
System.out.println("password: " + password);  
System.out.println("cassandra_password: " + defaultPassword);
```

```
password: mypassword  
cassandra_password: 123456
```

JDBC

Java Database Connectivity

JDBC

- Встановлювати зв'язок із базою даних;
- Виконувати SQL запити;
- Отримувати результати виконання SQL запитів;
- Керувати транзакціями;
- Обробляти помилки та виняткові ситуації.

Встановлення з'єднання

```
public class MysqlConnector {  
    public static Connection getConnection(Properties properties) throws SQLException {  
        return DriverManager.getConnection(  
            properties.getProperty("db.url"), properties);  
    }  
}
```

DriverManager class

- Діє як інтерфейс між користувачами та драйверами;
- Встановлює зв'язок між базою даних і відповідним драйвером;
- Містить усі відповідні методи для створення зв'язку між програмою Java і базою даних.

Connection interface

- Потрібен для встановлення зв'язку між базою даних та програмою;
- Являється породжуючим для:
 - Statement();
 - PreparedStatement();
 - DatabaseMetadata();
- Надає методи керування транзакціями:
 - commit();
 - rollback();
 - setAutocommit();

Statement Interface

- Надає методи для виконання запитів у базу даних;
- Являється породжуючим для ResultSet;

```
public ResultSet executeQuery(String sql):
```

Використовується для виконання запиту SELECT. Повертає об'єкт ResultSet.

```
public int executeUpdate(String sql):
```

Використовується для виконання запитів типу: CREATE, DELETE, INSERT...

```
public boolean execute(String sql):
```

використовується для виконання запитів різних типів.

```
public int[] executeBatch():
```

Використовується для виконання сету команд за один раз.

```
Connection connection = MySqlConnection.getConnection(properties);
Statement statement = connection.createStatement();

int result = statement.executeUpdate("INSERT INTO `user` (`name`) VALUES ('john');");
System.out.println("Records affected: " + result);

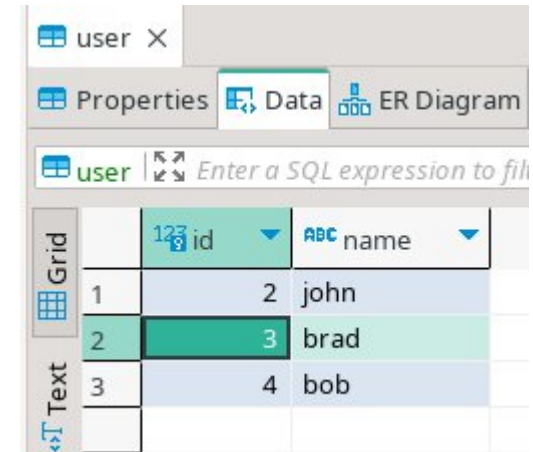
result = statement.executeUpdate(
    "INSERT INTO `user` (`name`) VALUES ('brad'), ('bob')");
System.out.println("Records affected: " + result);

statement.close();
connection.close();
```



```
Main x
↑ /usr/lib/jvm/java-17-openjdk/bin/java -javaagent:/opt/jetbrains/idea-IU-223.861
↓ {password=appuser, user=appuser, db.url=jdbc:mysql://localhost:3306/demo_243}
Records affected: 1
Records affected: 2

Process finished with exit code 0
```



	id	name
1	2	john
2	3	brad
3	4	bob

ResultSet Interface

- Об'єкт ResultSet підтримує курсор, що вказує на рядок таблиці. Спочатку курсор вказує на перед першим рядком.

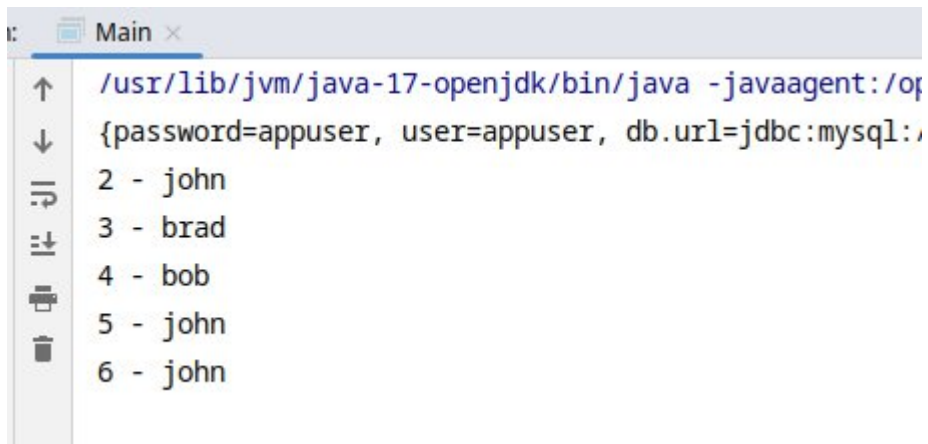
```
Statement statement = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,  
        ResultSet.CONCUR_UPDATABLE);
```

```

Connection connection = MySqlConnection.getConnection(properties);
Statement statement = connection.createStatement();
int result = statement.executeUpdate("INSERT INTO `user` (`name`) VALUES ('john');");

ResultSet resultSet = statement.executeQuery("SELECT * FROM user");
while (resultSet.next()) {
    System.out.println(resultSet.getInt(1));
    System.out.println(resultSet.getString(2));
}
resultSet.close();
statement.close();
connection.close();

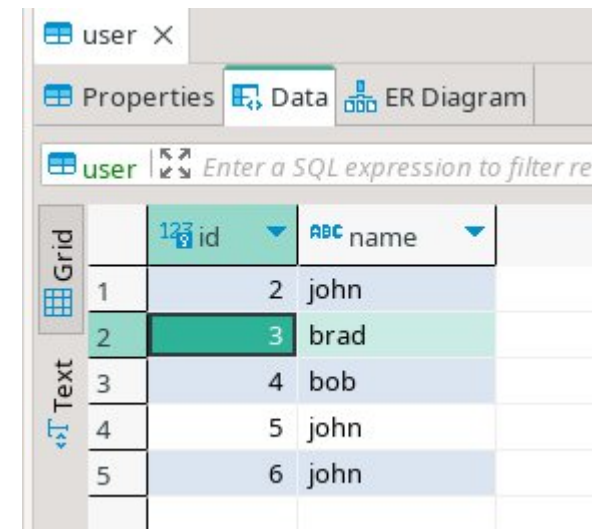
```



```

Main x
↑ /usr/lib/jvm/java-17-openjdk/bin/java -javaagent:/o
↓ {password=appuser, user=appuser, db.url=jdbc:mysql:
2 - john
3 - brad
4 - bob
5 - john
6 - john

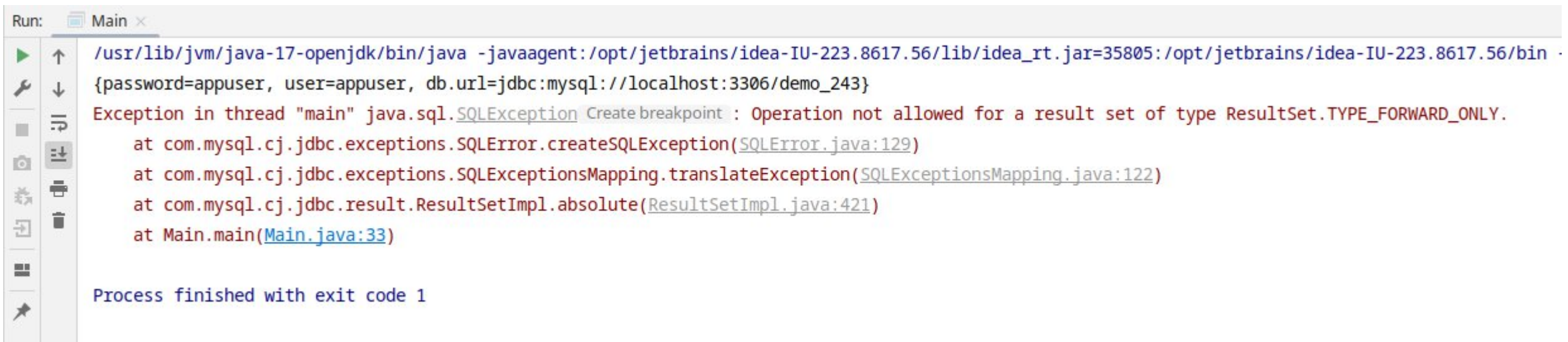
```



	id	name
1	2	john
2	3	brad
3	4	bob
4	5	john
5	6	john

```
Connection connection = MysqlConnector.getConnection(properties);  
Statement statement = connection.createStatement();
```

```
ResultSet resultSet = statement.executeQuery("SELECT * FROM user");  
if (resultSet.absolute(1)) {  
    System.out.println(resultSet.getInt(1) + " - " + resultSet.getString(2));  
}  
  
if (resultSet.absolute(5)) {  
    System.out.println(resultSet.getInt(1) + " - " + resultSet.getString(2));  
}
```



The screenshot shows the 'Run' window of an IDE. At the top, it says 'Run: Main x'. Below this, the command line for the Java process is displayed: `/usr/lib/jvm/java-17-openjdk/bin/java -javaagent:/opt/jetbrains/idea-IU-223.8617.56/lib/idea_rt.jar=35805:/opt/jetbrains/idea-IU-223.8617.56/bin {password=appuser, user=appuser, db.url=jdbc:mysql://localhost:3306/demo_243}`. The main part of the window shows a stack trace for a `SQLException` with the message 'Operation not allowed for a result set of type ResultSet.TYPE_FORWARD_ONLY'. The stack trace includes the following frames: `com.mysql.cj.jdbc.exceptions.SQLException.createSQLException` (line 129), `com.mysql.cj.jdbc.exceptions.SQLExceptionsMapping.translateException` (line 122), `com.mysql.cj.jdbc.result.ResultSetImpl.absolute` (line 421), and `Main.main` (line 33). At the bottom, it states 'Process finished with exit code 1'. On the left side of the window, there is a vertical toolbar with icons for running, debugging, and other IDE functions.

```
Run: Main x  
/usr/lib/jvm/java-17-openjdk/bin/java -javaagent:/opt/jetbrains/idea-IU-223.8617.56/lib/idea_rt.jar=35805:/opt/jetbrains/idea-IU-223.8617.56/bin  
{password=appuser, user=appuser, db.url=jdbc:mysql://localhost:3306/demo_243}  
Exception in thread "main" java.sql.SQLException Create breakpoint : Operation not allowed for a result set of type ResultSet.TYPE_FORWARD_ONLY.  
    at com.mysql.cj.jdbc.exceptions.SQLException.createSQLException(SQLException.java:129)  
    at com.mysql.cj.jdbc.exceptions.SQLExceptionsMapping.translateException(SQLExceptionsMapping.java:122)  
    at com.mysql.cj.jdbc.result.ResultSetImpl.absolute(ResultSetImpl.java:421)  
    at Main.main(Main.java:33)  
  
Process finished with exit code 1
```

```

Connection connection = MySqlConnection.getConnection(properties);
Statement statement = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);

ResultSet resultSet = statement.executeQuery("SELECT * FROM user");
if (resultSet.absolute(1)) {
    System.out.println(resultSet.getInt(1) + " - " + resultSet.getString(2));
}
if (resultSet.absolute(5)) {
    System.out.println(resultSet.getInt(1) + " - " + resultSet.getString(2));
}

```

Run: Main x

```

/usr/lib/jvm/java-17-openjdk/bin/java -javaagent:/opt/jetbrains/idea-IU-223.861
{password=appuser, user=appuser, db.url=jdbc:mysql://localhost:3306/demo_243}
2 - john
6 - john

Process finished with exit code 0

```

user X

Properties Data ER Diagram

user Enter a SQL expression to filter re

	id	name
1	2	john
2	3	brad
3	4	bob
4	5	john
5	6	john

PreparedStatement Interface

- Є підінтерфейсом Statement.
- Використовується для виконання параметризованого запиту.

PreparedStatement Interface

```
statement.executeUpdate("INSERT INTO `user` (`name`) VALUES ('john');");
```

```
String sqlQuery = "SELECT * from user where name = ?";
```

```
Connection connection = MySqlConnection.getConnection(properties);
```

```
PreparedStatement preparedStatement = connection.prepareStatement(sqlQuery);
```

```
preparedStatement.setObject(1, "john", Types.VARCHAR);
```

```
ResultSet resultSet = preparedStatement.executeQuery();
```

```
while (resultSet.next()) {
```

```
    System.out.println(resultSet.getInt(1) + " - " + resultSet.getString("name"));
```

```
}
```

```
resultSet.close();
```

```
preparedStatement.close();
```

```
connection.close();
```

PreparedStatement Interface

```
String sqlQuery = "INSERT INTO `user` (`name`) VALUES (?)";

Connection connection = MysqlConnector.getConnection(properties);
PreparedStatement preparedStatement = connection.prepareStatement(sqlQuery,
    Statement.RETURN_GENERATED_KEYS);


preparedStatement.setObject(1, "vlad", Types.VARCHAR);
if (preparedStatement.executeUpdate() > 0) {
    ResultSet resultSet = preparedStatement.getGeneratedKeys();
    while (resultSet.next()) {
        System.out.println("Inserted id: " + resultSet.getInt(1));
    }
}
```

ResultSetMetadata Interface

- getColumnCount();
- getColumnName(int column);
- getColumnType(int column);
- getColumnName(int column);
- getColumnDisplaySize(int column);
- isNullable(int column);
- isAutoIncrement(int column);
- isReadOnly(int column);
- isSearchable(int column);
- isSigned(int column);


```
String sqlQuery = "SELECT * from user where name = ?";

try (Connection connection = MySqlConnection.getConnection(properties);
    PreparedStatement preparedStatement = connection.prepareStatement(sqlQuery)) {
    preparedStatement.setObject(1, "john", Types.VARCHAR);
    try (ResultSet resultSet = preparedStatement.executeQuery()) {
        ResultSetMetaData metaData = resultSet.getMetaData();
        while (resultSet.next()) {
            System.out.println(resultSet.getInt(1) + " - " +
                               resultSet.getString("name"));
        }
        System.out.println("Amount of columns: " + metaData.getColumnCount());
        System.out.println("First columnName: " + metaData.getColumnName(1));
        System.out.println("Second columnName: " + metaData.getColumnName(2));
    }
} catch (SQLException e) {
    e.printStackTrace();
}
```



```
/ - john
Amount of columns: 2
First columnName: id
Second columnName: name
```

Connection pool

Встановлення з'єднання із БД

- Створити об'єкт Connection;
- Відкриття TCP-сокета для читання/запису даних;
- Читання / запис даних через сокет;
- Закриття Connection об'єкту;
- Закриття сокета.

```
public class SimpleConnectionPool {  
  
    public static SimpleConnectionPool instance;  
  
    private String jdbcUrl;  
    private String username;  
    private String password;  
    private List<Connection> connectionPool;  
    private int maxConnections;  
}
```

```
public static SimpleConnectionPool getInstance(String jdbcUrl, String username, String password, int maxConnections) {  
    if (instance == null) {  
        return new SimpleConnectionPool(jdbcUrl, username, password, maxConnections);  
    }  
    return instance;  
}
```

```
private SimpleConnectionPool(String jdbcUrl, String username, String password, int maxConnections) {  
    this.jdbcUrl = jdbcUrl;  
    this.username = username;  
    this.password = password;  
    this.maxConnections = maxConnections;  
    this.connectionPool = new ArrayList<>(maxConnections);  
  
    initializeConnectionPool();  
}
```

```
private void initializeConnectionPool() {  
    try {  
        for (int i = 0; i < maxConnections; i++) {  
            Connection connection = DriverManager.getConnection(jdbcUrl, username, password);  
            connectionPool.add(connection);  
        }  
    } catch (SQLException e) {  
        throw new RuntimeException("Error initializing the connection pool: " + e.getMessage(), e);  
    }  
}
```

```
public synchronized Connection getConnection() {  
    if (connectionPool.isEmpty()) {  
        throw new RuntimeException("No available connections in the pool");  
    }  
  
    Connection connection = connectionPool.remove(connectionPool.size() - 1);  
  
    try {  
        if (connection.isClosed() || !connection.isValid(5)) {  
            connection = DriverManager.getConnection(jdbcUrl, username, password);  
        }  
    } catch (SQLException e) {  
        throw new RuntimeException("Error getting a connection from the pool: " + e.getMessage(), e);  
    }  
  
    return connection;  
}
```

```
public synchronized void releaseConnection(Connection connection) {  
    if (connection != null) {  
        connectionPool.add(connection);  
    }  
}
```

```
public void shutdown() {  
    for (Connection connection : connectionPool) {  
        try {  
            connection.close();  
        } catch (SQLException e) {  
            }  
    }  
    connectionPool.clear();  
}
```


Transactions

```
try {  
    connection.setAutoCommit(false);  
    //...  
  
    connection.commit();  
  
} catch (SQLException e) {  
  
    connection.rollback();  
  
} finally {  
    connection.setAutoCommit(true);  
}
```

DTO vs DAO

Data Transfer Object

```
public class UserDTO {  
    private int id;  
    private String name;  
    private String email;  
  
}
```

Data Access Object

```
public class UserDao {
    private Connection conn;
    private static String INSERT_USER =
        "INSERT INTO users (name, email) VALUES (?, ?)";
    private static String GET_USER_BY_ID = "SELECT * FROM users WHERE id = ?";

    public UserDao(Connection conn) {
        this.conn = conn;
    }

    public void addUser(UserDTO user) throws SQLException {
        PreparedStatement stmt = conn.prepareStatement(INSERT_USER);
        stmt.setString(1, user.getName());
        stmt.setString(2, user.getEmail());
        stmt.executeUpdate();
    }
}
```