JDBC

- Follow the instructions to install the database H2;
- in the project **Prograd**, create a package to take care of database connections:

  br.edu.ufabc.prograd.jdbc
- JDBC (Java DataBase Connectivity): set of well-defined interfaces inside the package java.sql;
- driver JDBC: concrete classes making a link between API JDBC and the database;
- in the package **br.edu.ufabc.prograd.jdbc**, create a class named **ConnectionFactory** with the code:

```
package br.edu.ufabc.prograd.jdbc;

public class ConnectionFactory {
	public Connection getConnection(){
		System.out.println("Connecting to the database");
		try {
		// the line below is not necessary anymore from JDBC 4 (Java 6)
			//Class.forName("org.h2.Driver");
		return
DriverManager.getConnection("jdbc:h2:tcp://localhost/~progweb","admin","admin");
		} catch (Exception e) {
			throw new RuntimeException(e);
		}
	}
}
```

- add the imports needed (`java.sql.Connection and java.sql.DriverManager`) using the shortcut **ctrl+1**;
- add the driver of H2 (a JAR file) containing the implementation JDBC of H2 to the *classpath* of the project. The driver of H2 is called h2-1.3.170.jar and is inside the folder h2 -> pasta bin. Follow the steps: right click over the project, option ***Build Path -> Configure Build Path -> Libraries -> Add External JARs.***
- Test the connection with the following code (add exception handling with *try-catch*) using **ctrl+1**):

```
// testing connection
Connection connection = new ConnectionFactory().getConnection();
System.out.println("Connection done");
connection.close();
```

- make and test the connection;
- always handle exceptions and always remember to close connections to the database;
- the drivers can be downloaded from manufacturer's website;
- why a connection factory: only one place to modify when/if needed; possibility to create a pool

of connections; it follows design patterns about encapsulating the construction of complex objects (see Go4 book);

- in the project **Prograd**, create a package responsible for Data Access Objects (DAO):

  br.edu.ufabc.prograd.dao

- in the package **br.edu.ufabc.prograd.dao**, create a Javabean named **StudentDAO**;
- remember to import the class `java.sql` (careful: it's not the class `com.sql`);
- add the connection with the database in the constructor of StudentDAO:

```
public class StudentDAO {
	private Connection connection;

	public StudentDAO() {
		this.connection = new ConnectionFactory().getConnection();
	}
```

- implement the insertion:

```
	public void insert(Student student) {
	String sql = "insert into students (name,email,address) values
(?,?,?)";

	try { // prepared statement for insertion
		PreparedStatement stmt = connection.prepareStatement(sql);

		// it sets the values
		stmt.setString(1, student.getName());
		stmt.setString(2, student.getEmail());
		stmt.setString(3, student.getAddress());

		// it runs
		stmt.execute();

		// it closes statement
		stmt.close();
	} catch (SQLException e) {
		throw new RuntimeException(e);
	}
}
```

- modify the class **CreateStudent**, adding a code to test the insertion:

```
	// recording a student
	StudentDAO dao = new StudentDAO();
	dao.insert(student);
	System.out.println("Recorded!");
```

  - problems with SQL:

```
String sql = "insert into students (name,email,address) values ('"+name+"','"+email+"'+'"+address+"')";
```

- difficult to read;
- likely to have errors;
- prejudice against Joana D'Arc;
- SQL Injection: technique used to attack data-driven applications. **Example**: http://xkcd.com/327/
- sanitize inputs for database: input handling (special characters, etc);
- more generic and more secure:

```
String sql = "insert into students (name,email,address) values (?,?,?)";
```

- PreparedStatement: interface to execute clauses, it sanitizes inputs for database;