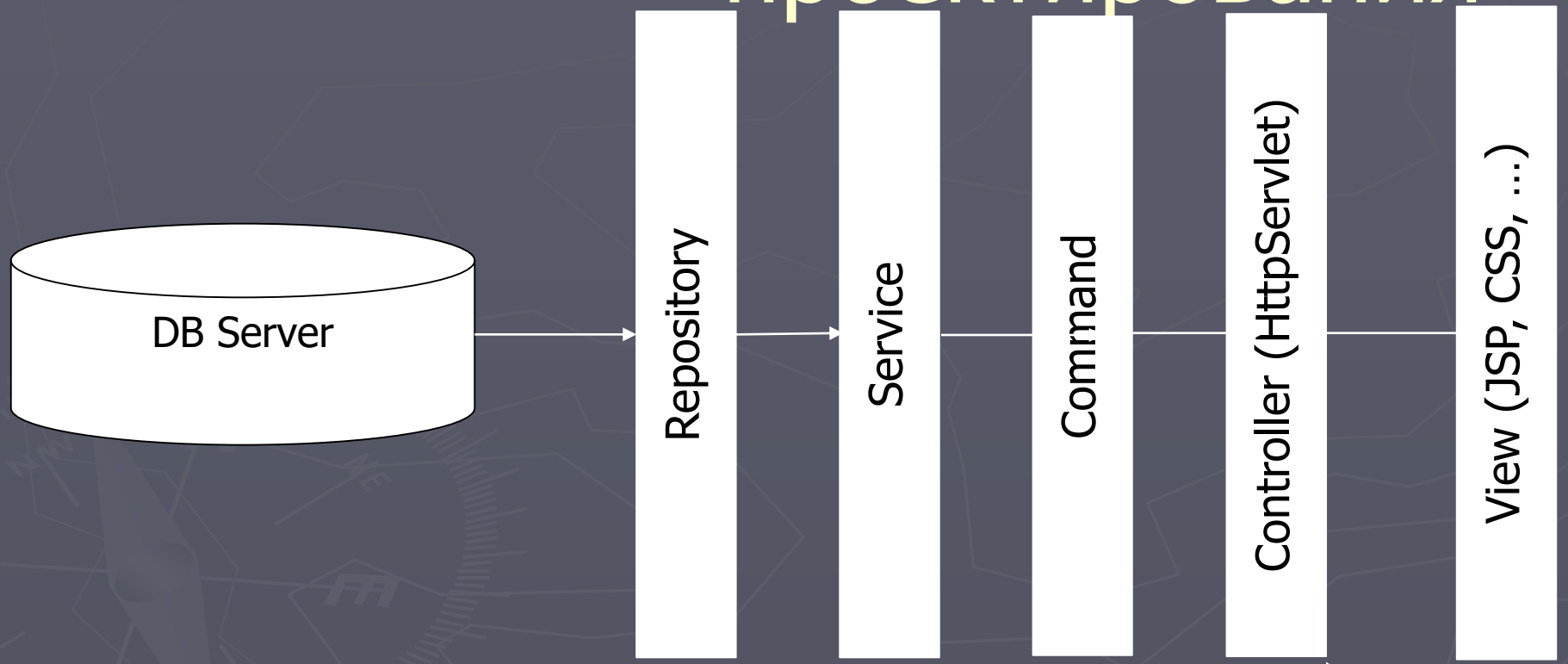
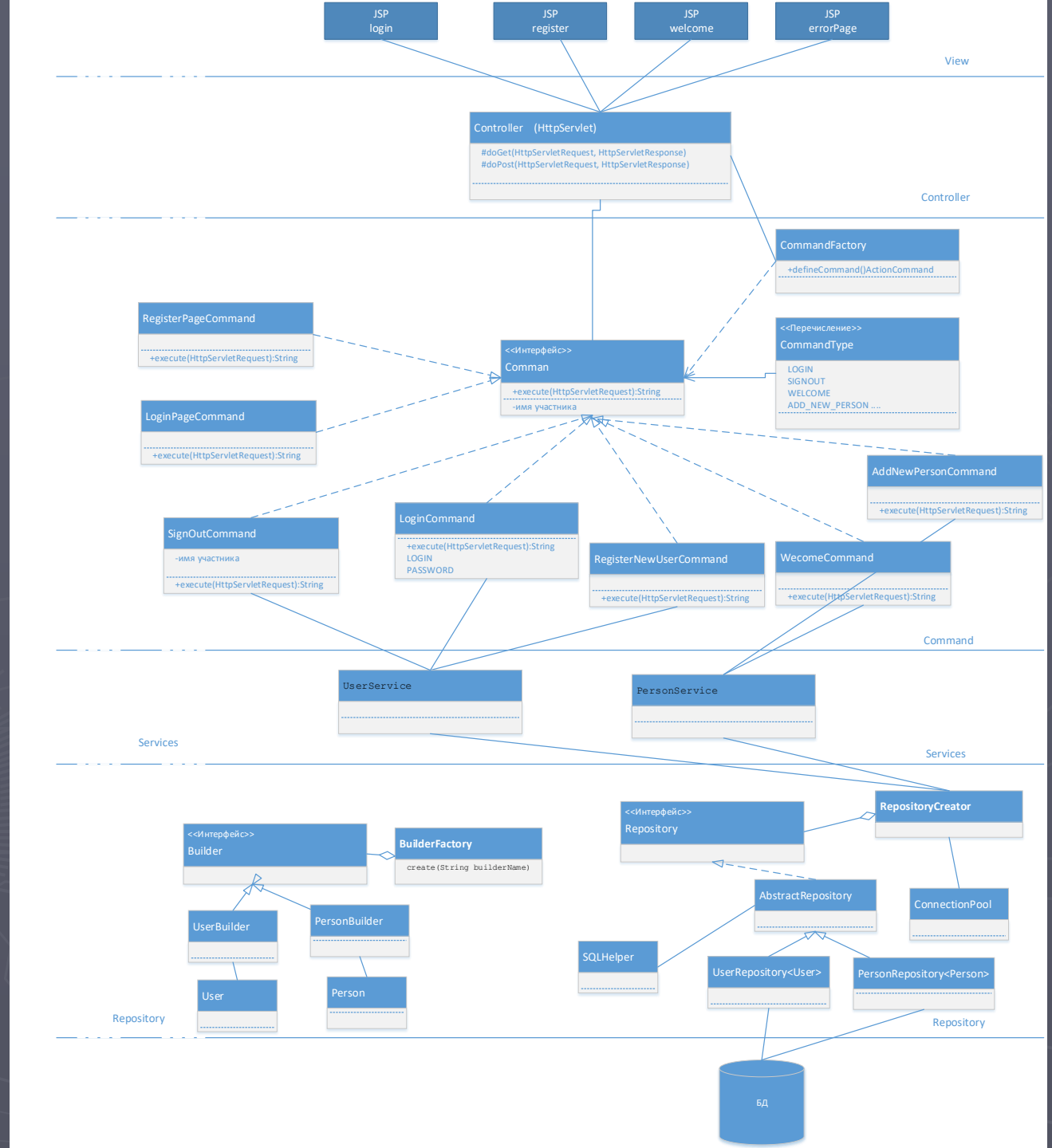
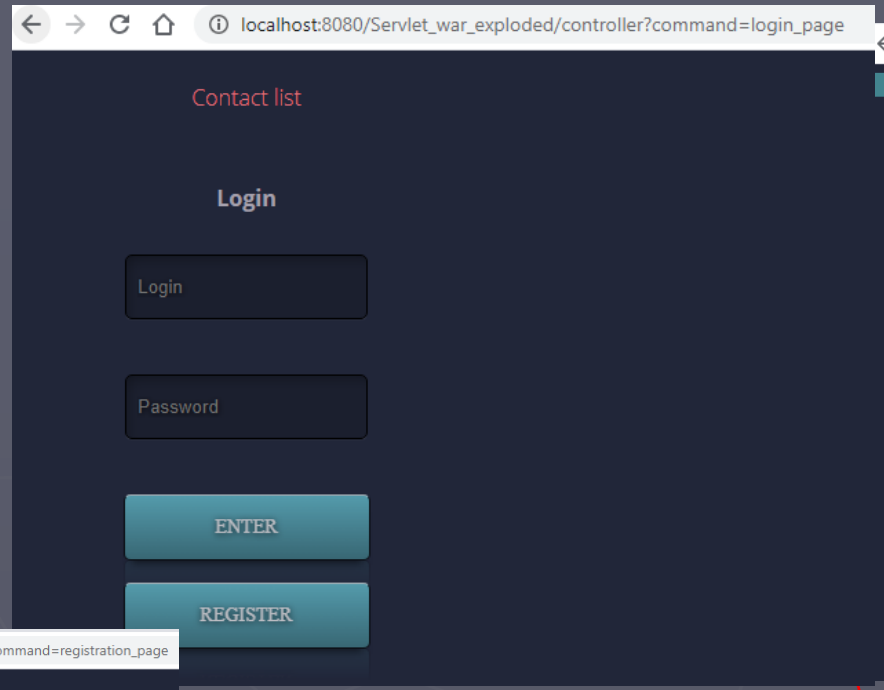
Многоуровневая архитектура приложения. Применение шаблонов проектирования

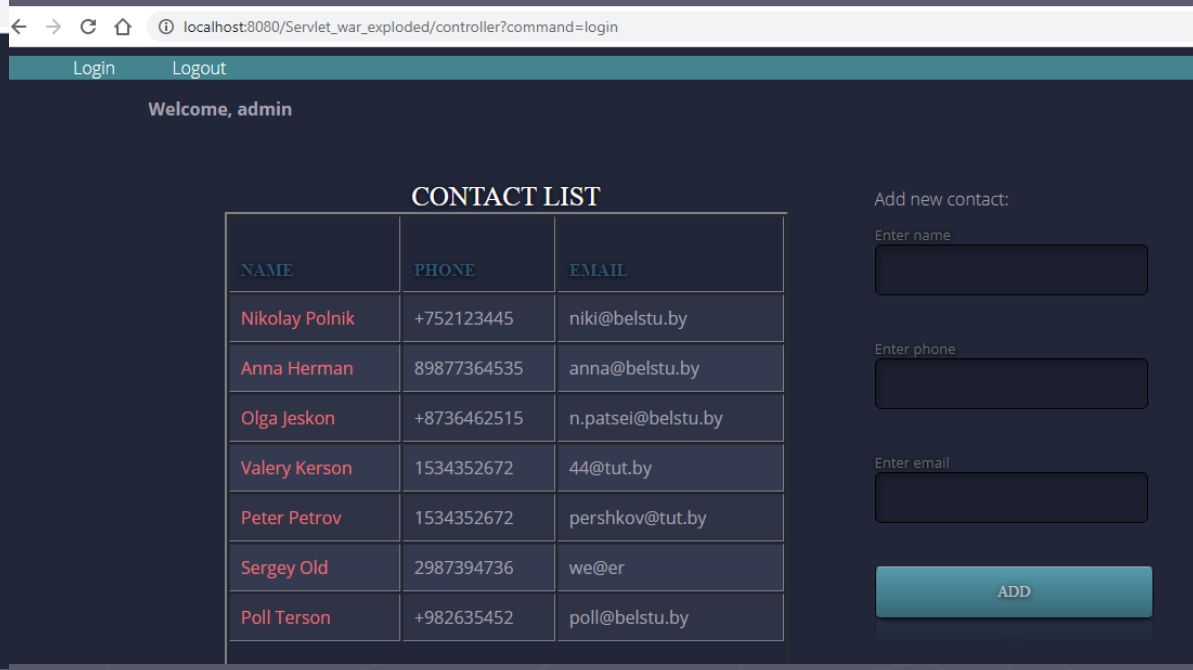


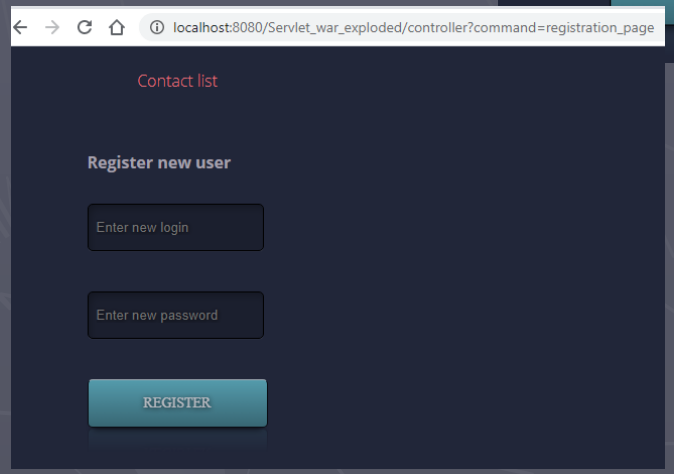
MVC (Model/View/Controller).

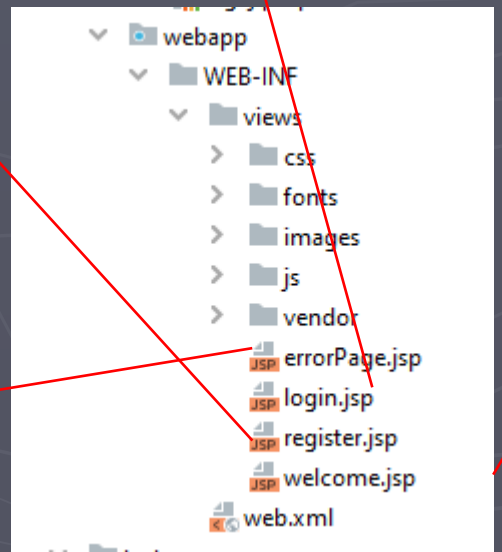

JSP
login
JSP
register
JSP
welcome
JSP
errorPage
View
Controller (HttpServlet)
#doGet(HttpServletRequest, HttpServletResponse)
#doPost(HttpServletRequest, HttpServletResponse)
Controller
CommandFactory
+defineCommand()ActionCommand
RegisterPageCommand
+execute(HttpServletRequest):String
<<Интерфейс>>
Comman
+execute(HttpServletRequest):String
-имя участника
<<Перечисление>>
CommandType
LOGIN
SIGNOUT
WELCOME
ADD_NEW_PERSON ....
LoginPageCommand
+execute(HttpServletRequest):String
AddNewPersonCommand
+execute(HttpServletRequest):String
SignOutCommand
-имя участника
+execute(HttpServletRequest):String
LoginCommand
+execute(HttpServletRequest):String
LOGIN
PASSWORD
RegisterNewUserCommand
+execute(HttpServletRequest):String
WecomeCommand
+execute(HttpServletRequest):String
Command
UserService
PersonService
Services
Services
RepositoryCreator
<<Интерфейс>>
Builder
BuilderFactory
create(String builderName)
<<Интерфейс>>
Repository
AbstractRepository
ConnectionPool
UserBuilder
PersonBuilder
SQLHelper
UserRepository<User>
PersonRepository<Person>
User
Person
Repository
Repository
БД

1. JSP


localhost:8080/Servlet_war_exploded/controller?command=login_page
Contact list
Login
Login
Password
ENTER
REGISTER


localhost:8080/Servlet_war_exploded/controller?command=login
Login
Logout
Welcome, admin
CONTACT LIST
NAME
PHONE
EMAIL
Nikolay Polnik
+752123445
niki@belstu.by
Anna Herman
89877364535
anna@belstu.by
Olga Jeskon
+8736462515
n.patsei@belstu.by
Valery Kerson
1534352672
44@tut.by
Peter Petrov
1534352672
pershkov@tut.by
Sergey Old
2987394736
we@er
Poll Terson
+982635452
poll@belstu.by
Add new contact:
Enter name
Enter phone
Enter email
ADD


localhost:8080/Servlet_war_exploded/controller?command=registration_page
Contact list
Register new user
Enter new login
Enter new password
REGISTER


webapp
WEB-INF
views
css
fonts
images
js
vendor
errorPage.jsp
login.jsp
register.jsp
welcome.jsp
web.xml


localhost:8080/Servlet_war_exploded/controller?command=register_new_user
ERROR...
Status code:
Column 'id' not found.

web.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">

    <servlet>
        <servlet-name>controller</servlet-name>
        <servlet-class>by.patsei.controller.Controller</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>controller</servlet-name>
        <url-pattern>/controller</url-pattern>
    </servlet-mapping>


    <welcome-file-list>
        <welcome-file>/WEB-INF/views/login.jsp</welcome-file>
    </welcome-file-list>

</web-app>
```

login.jsp

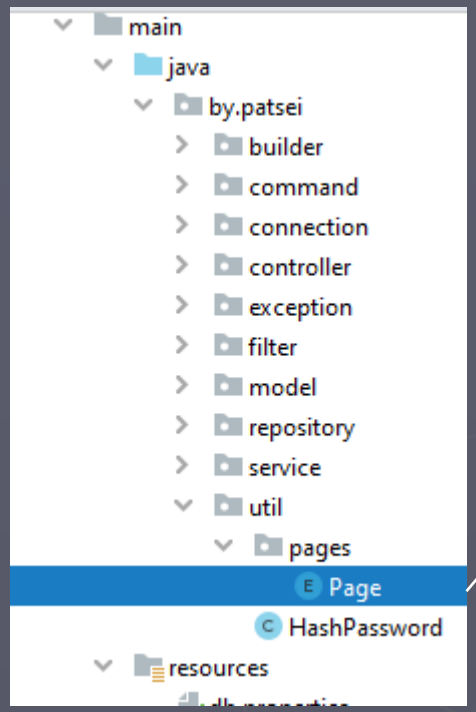
```
...
</body>
    <div class = "login-page">
    <div class="form">
<p><font color="red">${errorMessage}</font></p>
    <form class="login-form"  action="${pageContext.servletContext.contextPath}/controller?command=login" method="POST">
        <p> Вход  в систему </p>
        <p>  Имя     : <input  name="loginName" type="text" />
        </p>
       <p>   Пароль : <input name="password" type="password" />
        </p>
           <input class ="button-main-page" type="submit"  value="Войти"/>

    </form>
  <div>
      <form action="${pageContext.servletContext.contextPath}/controller?command=registration_page" method="post">
    <input class ="button-main-page"   type="submit"  value="Регистрация"/>
       </form>
   </div>
</div>
</div>

</body>
</html>
```

register.jsp

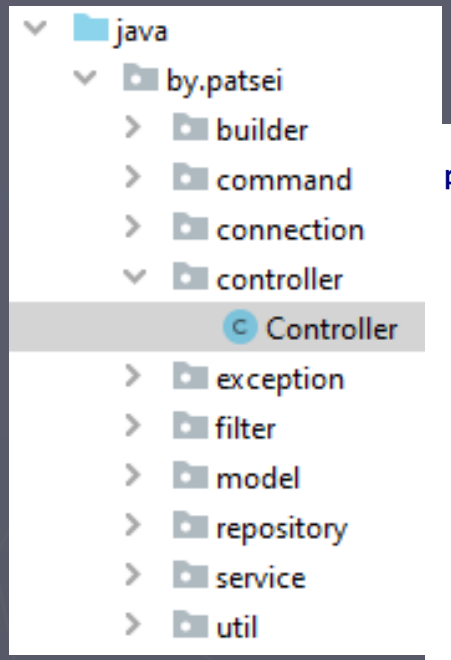
```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>

<p><font color="red">${errorRegister}</font></p>

<form action="${pageContext.servletContext.contextPath}/controller?command=register_new_user" method="POST">
    <p> Регистрация нового пользователя </p>
    <p>  Введите имя     : <input  name="newLoginName" type="text" />
    </p>
    <p>   Введите пароль : <input name="newPassword" type="password" />
    </p>
    <input class ="button-main-page" type="submit"  value="Зарегистрировать"/>

</form>
</body>
</html>
```

```
    <div class="navbar-collapse">
        <ul class="nav navbar-nav">
            <li class="active"><a href="#"></a></li>
        </ul>
        <ul class="nav navbar-nav navbar-right">
            <li><a href="${pageContext.request.contextPath}/controller?command=login_page">Вход</a></li>
        </ul>
        <ul class="nav navbar-nav navbar-right">
            <li><a href="${pageContext.servletContext.contextPath}/controller?command=sign_out">Выход</a></li>
        </ul>
    </div>

</nav>
<div class="container">
<H3>Добрый день,  ${username}</H3>
    <caption>Список вашей группы</caption>
        <table border="1">

            <tr>
                <th>Имя</th>
                <th>Телефон</th>
                <th>email</th>
            </tr>

            <c:forEach items="${group}" var="person">
            <tr><td>${person.name}</td>
                <td>${person.phone}</td>
                <td>${person.email}</td>
            </tr>
            </c:forEach>
        </table>

        <p><font color="red">${errorMessage}</font></p>
        <form method="POST" action="${pageContext.servletContext.contextPath}/controller?command=add_new_person">
            Новый :
            <p>  Введите имя <input name="nname" type="text" /> </p>
            <p>   Введите телефон <input name="nphone" type="text" /> </p>
            <p>  Введите email <input name="nemail" type="text" /> </p>
            <input class ="button-main-page" value="Добавить" type="submit" />
        </form>
    </div>
<p> ${lastdate}</p>
```

welcome.jsp

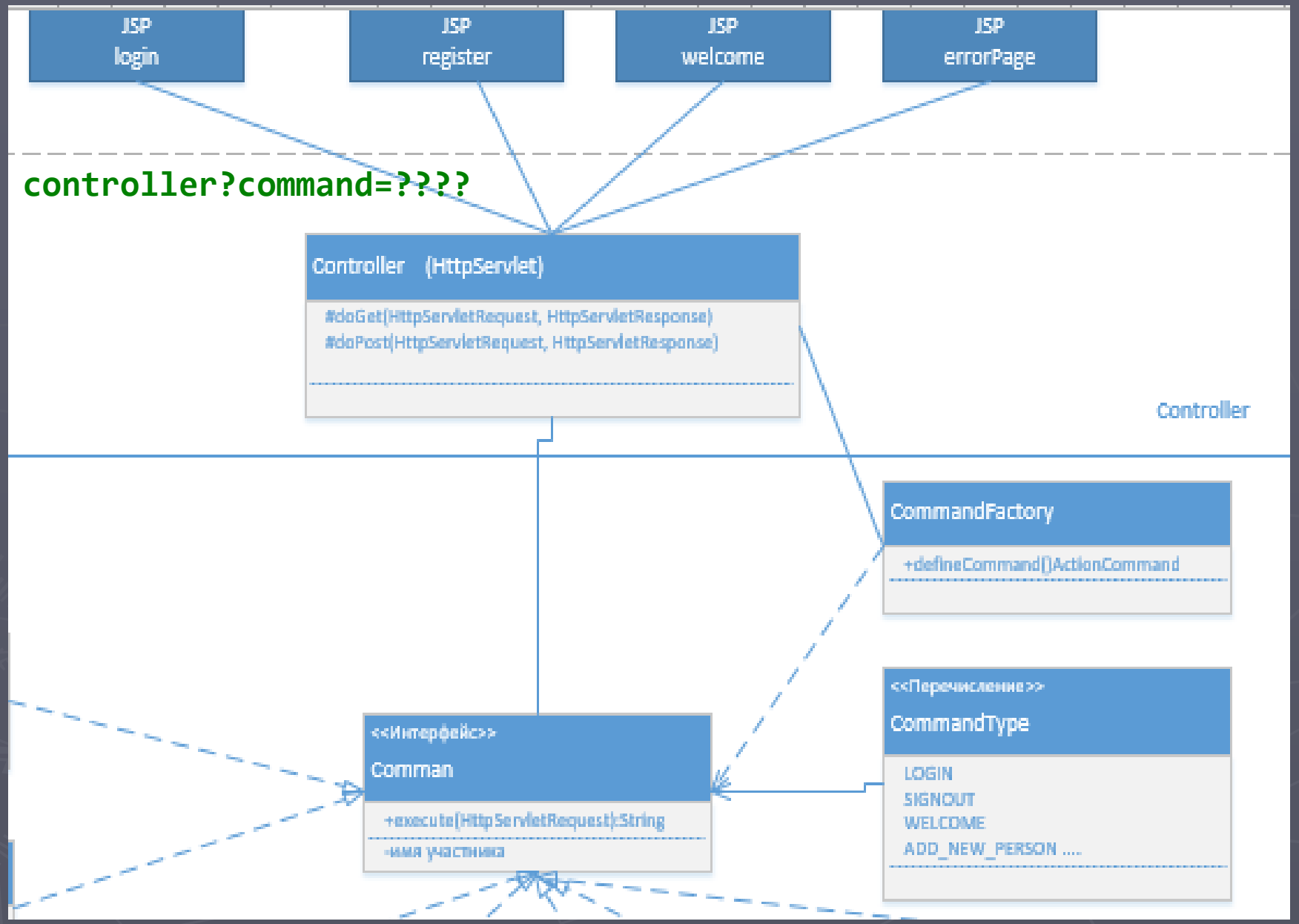



```java
// All pages of applications

public enum Page {

    LOGIN_PAGE("/WEB-INF/views/login.jsp"),
    REGISTER_PAGE("/WEB-INF/views/register.jsp"),
    WELCOME_PAGE ("/WEB-INF/views/welcome.jsp"),
    ERROR_PAGE ("/WEB-INF/views/errorPage.jsp");

    private final String value;

    Page(String value) {
        this.value = value;
    }

      public String getPage() {
        return value;
    }
}
```

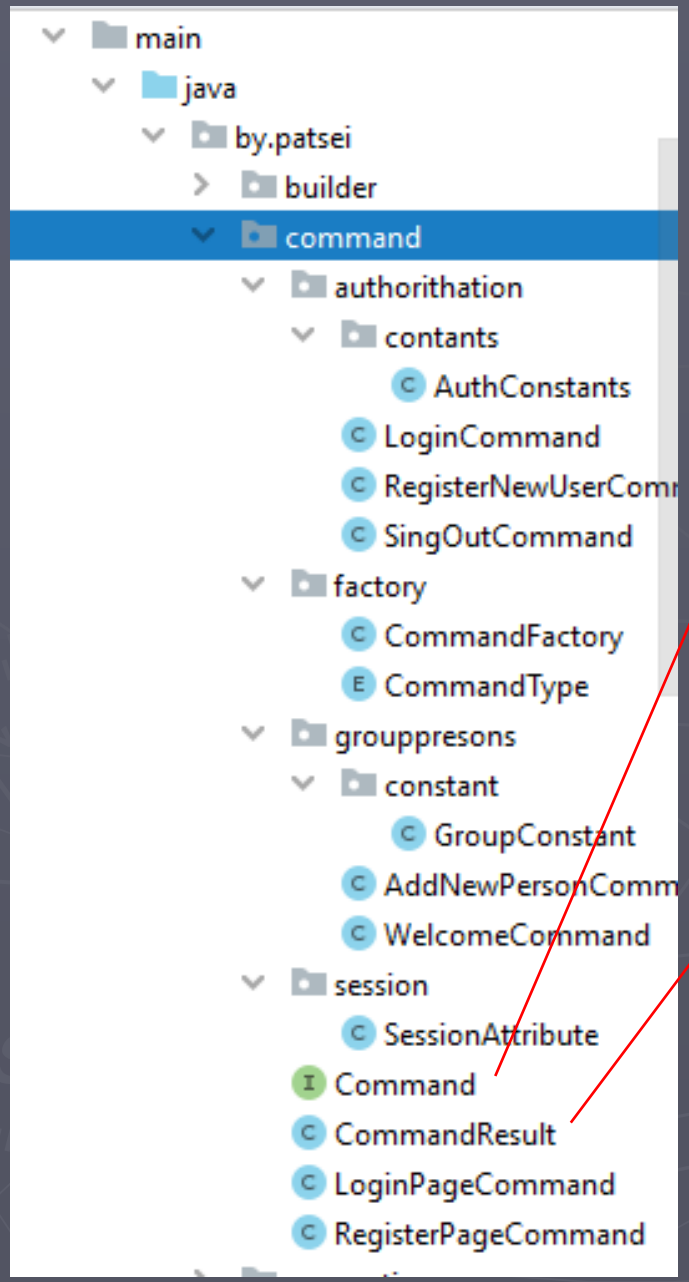


2. Controller

```
public class Controller extends HttpServlet {

    private static final String COMMAND = "command";
    private static final String ERROR_MESSAGE = "error_message";
    private static final Logger LOGGER = Logger.getLogger(Controller.class.getName());

    @Override
    public void destroy() {         ConnectionPool.getInstance().destroy();     }
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        processRequest(request, response);      }
    @Override
    public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        processRequest(request, response);      }

    private void processRequest(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String command = request.getParameter(COMMAND);
        LOGGER.info(COMMAND + "= " + command);
        Command action = CommandFactory.create(command);
        CommandResult commandResult;
        try {
            commandResult = action.execute(request, response);
        } catch (ServiceException | IncorrectDataException e) {
            LOGGER.error(e.getMessage(), e);
            request.setAttribute(ERROR_MESSAGE, e.getMessage());
            commandResult = new CommandResult(Page.ERROR_PAGE.getPage(), false);
        }
        String page = commandResult.getPage();
        if (commandResult.isRedirect()) {                   sendRedirect(response, page);
        } else {              dispatch(request, response, page);        }
    }
    private void dispatch(HttpServletRequest request, HttpServletResponse response, String page) throws ServletException, IOException {
        ServletContext servletContext = getServletContext();
        RequestDispatcher requestDispatcher = servletContext.getRequestDispatcher(page);
        requestDispatcher.forward(request, response);
    }
    private void sendRedirect(HttpServletResponse response, String page) throws IOException {
```

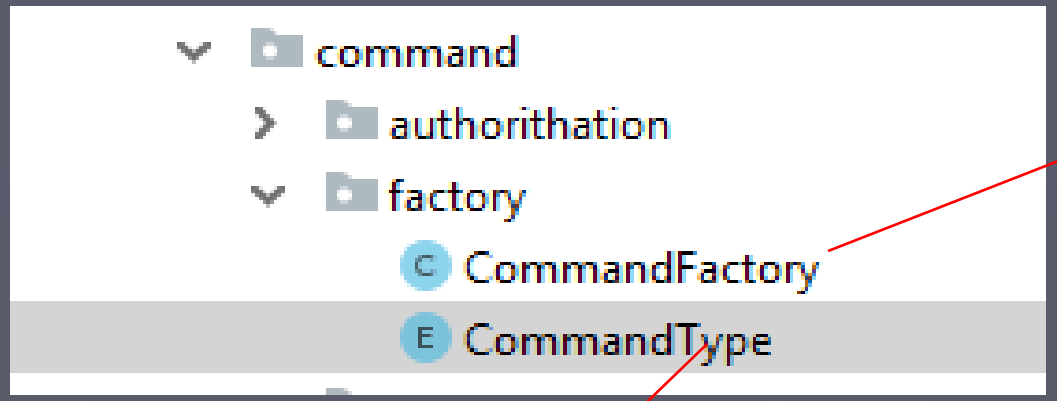

JSP
login
JSP
register
JSP
welcome
JSP
errorPage
controller?command=????
Controller (HttpServlet)
#doGet(HttpServletRequest, HttpServletResponse)
#doPost(HttpServletRequest, HttpServletResponse)
Controller
CommandFactory
+defineCommand():ActionCommand
<<Перечисление>>
CommandType
LOGIN
SIGNOUT
WELCOME
ADD_NEW_PERSON ....
<<Интерфейс>>
Comman
+execute(HttpServletRequest):String
-имя участника

3. УРОВЕНЬ БИЗНЕС ЛОГИКИ (Command)




```
public interface Command {
    CommandResult execute(HttpServletRequest request, HttpServlet
 throws ServiceException, IncorrectDataException, ServletExceptio
}
```

```
public class CommandResult {
    private String page;
    private boolean isRedirect;

    public CommandResult() {    }

    public CommandResult(String page) {   this.page = page;    }

    public CommandResult(String page, boolean isRedirect) {
        this.page = page;
        this.isRedirect = isRedirect;
    }

    public String getPage() {        return page;    }

    public void setPage(String page) {     this.page = page;      }

    public boolean isRedirect() {         return isRedirect;    }
```

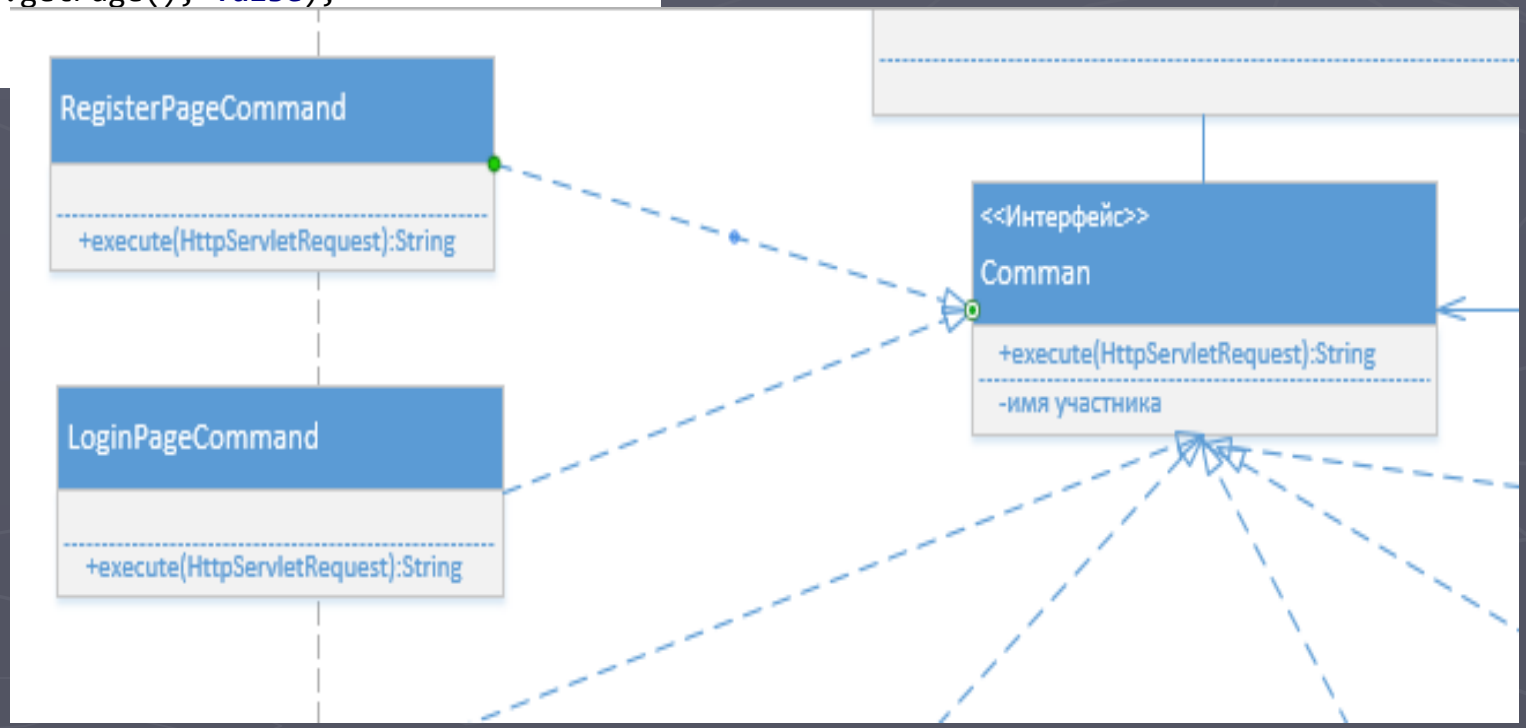


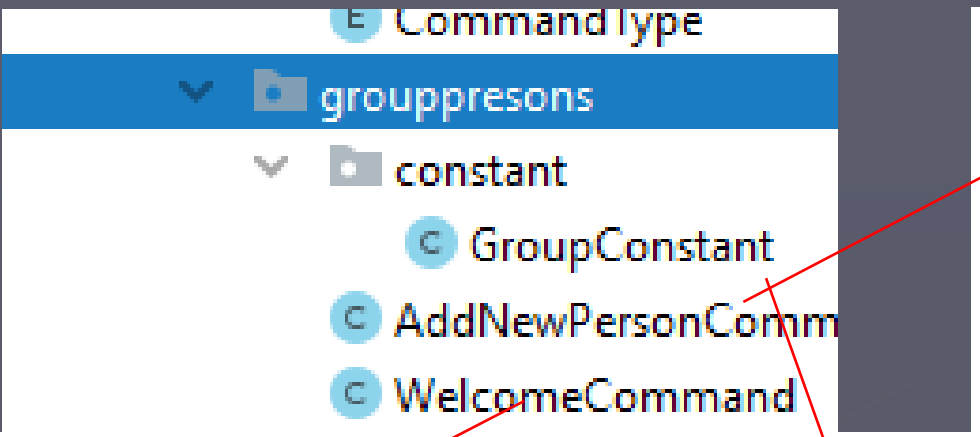

```java
/*
Хранилище команд
 */
public enum CommandType {

    LOGIN("login"),
    SIGN_OUT("sign_out"),
    WELCOME("welcome"),
    REGISTER_NEW_USER("register_new_user"),
    ADD_NEW_PERSON ("add_new_person"),
    LOGIN_PAGE("login_page"),
    REGISTRATION_PAGE("registration_page");

    private String command;
    private CommandType(String command) {
        this.command = command;
    }
}
```

```java
public class CommandFactory {
    public static Command create(String command) {
        command = command.toUpperCase();
        System.out.println(command);
        CommandType commandEnum = CommandType.valueOf(command);
        Command resultCommand;
        switch (commandEnum) {
            case LOGIN: {
                resultCommand = new LoginCommand();              break;
            }
            case REGISTER_NEW_USER: {
                resultCommand = new RegisterNewUserCommand();          break;
            }
            case SIGN_OUT: {
                resultCommand = new SingOutCommand();            break;
            }
            case ADD_NEW_PERSON:{
                resultCommand = new AddNewPersonCommand();         break;
            }
            case LOGIN_PAGE:{
                resultCommand = new LoginPageCommand();         break;
            }
            case WELCOME:{
                resultCommand = new WelcomeCommand();          break;
            }
            case REGISTRATION_PAGE:{
                resultCommand = new RegisterPageCommand();     break;
            }

            default: {
                throw new IllegalArgumentException("Invalid command" + commandEr
            }
        }
        return resultCommand;
    }
}
```

```java
public class LoginPageCommand implements Command {
    @Override
    public CommandResult execute(HttpServletRequest request,
HttpServletResponse response) throws ServiceException,
IncorrectDataException {
        System.out.println("LOGIN PAGE");
        return new CommandResult(Page.LOGIN_PAGE.getPage(),false);
    }
```

```java
public class RegisterPageCommand implements Command {
    @Override
    public CommandResult execute(HttpServletRequest request, HttpServletResponse
response) throws ServiceException, IncorrectDataException, ServiceException,
IOException {
        System.out.println("REGISTER_PAGE");
        return new CommandResult(Page.REGISTER_PAGE.getPage(), false);
    }
}
```

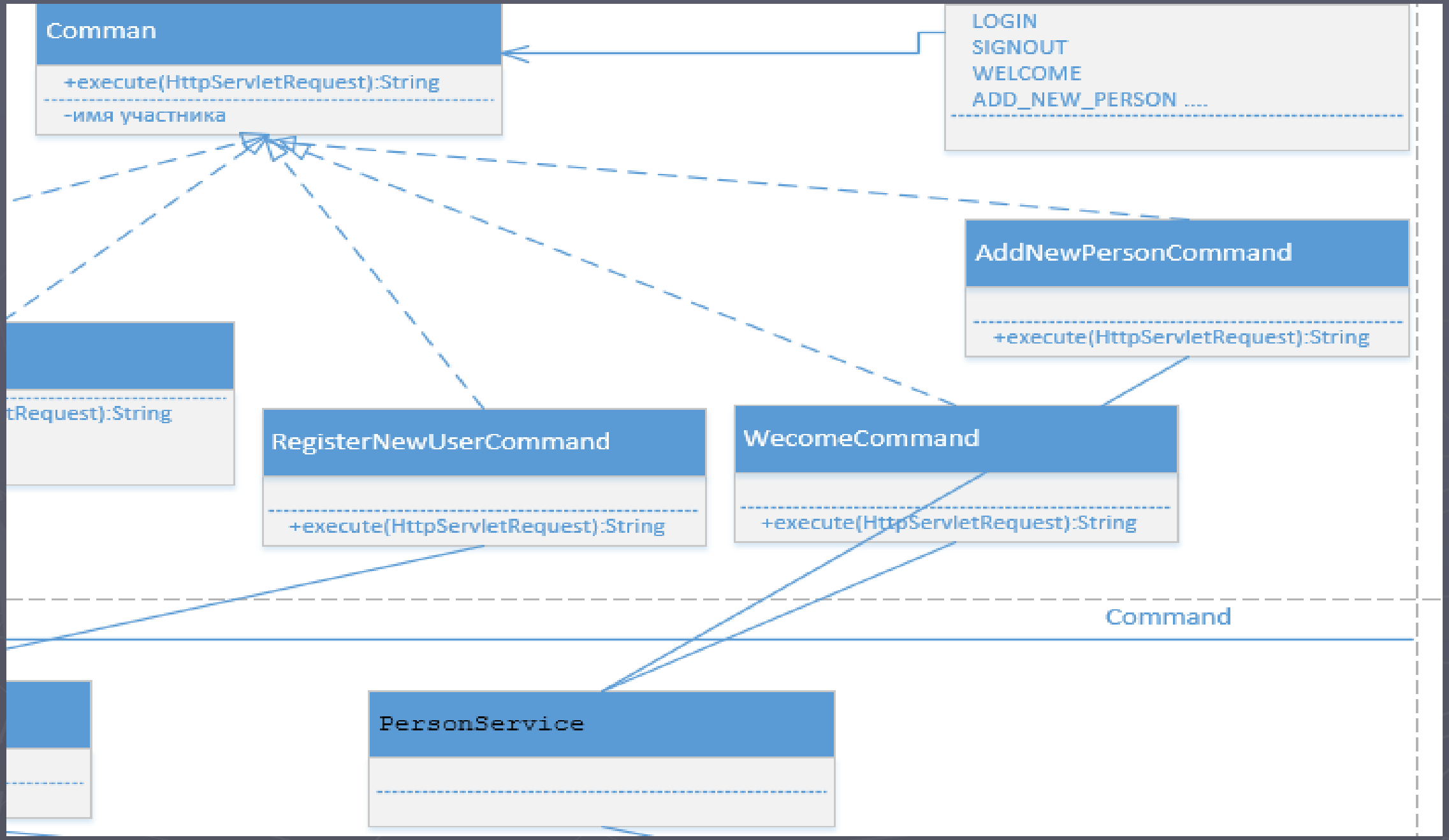


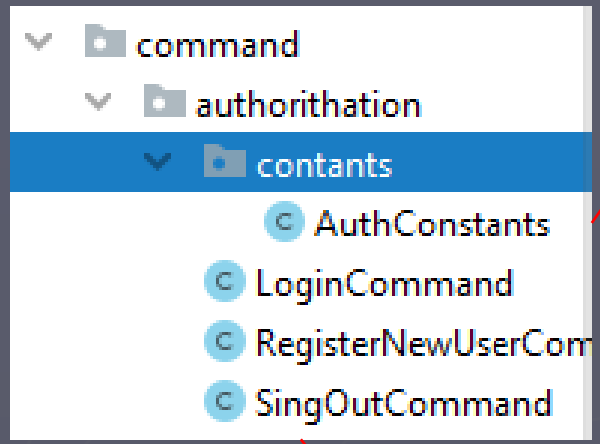



```java
public class WelcomeCommand implements Command {
    @Override
    public CommandResult execute(HttpServletRequest request,
            throws ServiceException, IncorrectDataException

        PersonService personService = new PersonService();
        List<Person> clients = personService.findAll();
        if (!clients.isEmpty()) {
            request.setAttribute(LISTGROUP, clients);
        }
        return new CommandResult(Page.WELCOME_PAGE.getPage()
    }
}
```

```java
public class GroupConstant {

//welcome jsp
    public static final String NEWNAME = "nname",
    public static final String NEWPHONE = "nphone";
    public static final String NEWEMAIL = "nemail";

//ERROR MESSAGE
    public final static String ERROR_MESSAGE =   "errorMessage";
    public final static String ERROR_MESSAGE_TEXT =   "Заполните все поля";
    public final static String LISTGROUP = "group";
}
```

```java
public class AddNewPersonCommand implements Command {
    private static final Logger LOGGER = Logger.getLogger(AddNewPersonCommand.class.getName());

    @Override
    public CommandResult execute(HttpServletRequest request, HttpServletResponse response)
            throws ServiceException, IncorrectDataException, ServletException {

        PersonService personService = new PersonService();
        Optional<String> newName = of(request)
                .map(httpServletRequest -> httpServletRequest.getParameter(NEWNAME));
        Optional<String> newPhone = of(request)
                .map(httpServletRequest -> httpServletRequest.getParameter(NEWPHONE));
        Optional<String> newEmail = of(request)
                .map(httpServletRequest -> httpServletRequest.getParameter(NEWEMAIL));

        if (isEmpty(newName.get()) || isEmpty(newPhone.get()) || isEmpty(newEmail.get())) {
            LOGGER.info("missing parameter for new person in addition mode");
            request.setAttribute(ERROR_MESSAGE, ERROR_MESSAGE_TEXT);
        } else {
            Person newperson = new Person(newName.get(), newPhone.get(), newEmail.get());
            personService.save(newperson);
        }
        List<Person> clients = personService.findAll();
        if (!clients.isEmpty()) {
            request.setAttribute(LISTGROUP, clients);
        }
        return new CommandResult(Page.WELCOME_PAGE.getPage(), false);
    }
}
```

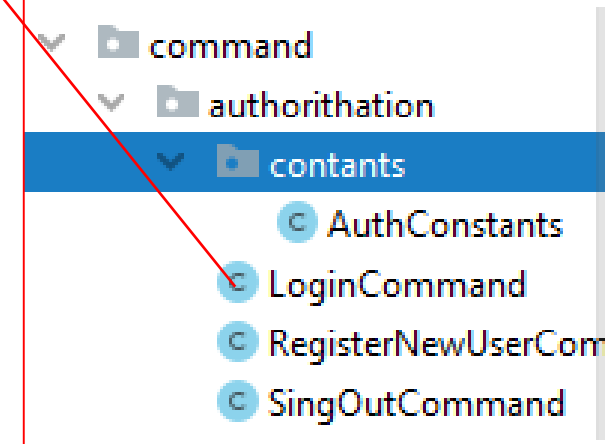

Comman
+execute(HttpServletRequest):String
-имя участника
LOGIN
SIGNOUT
WELCOME
ADD_NEW_PERSON ....
AddNewPersonCommand
+execute(HttpServletRequest):String
tRequest):String
RegisterNewUserCommand
+execute(HttpServletRequest):String
WecomeCommand
+execute(HttpServletRequest):String
Command
PersonService



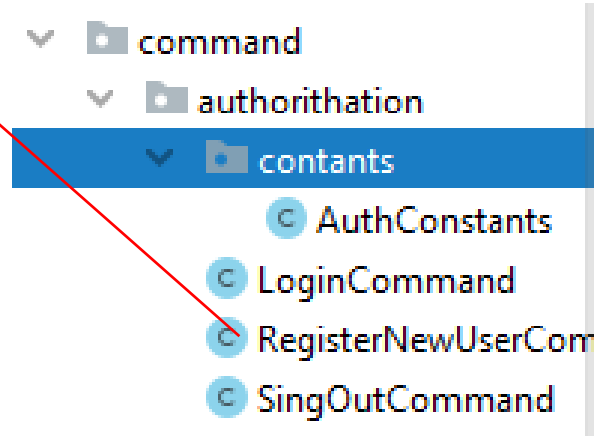

```java
public class AuthConstants {

//Login jsp
    public final static String LOGIN = "loginName";
    public final static String PASSWORD = "password";

//ERROR MESSAGE
    public final static String ERROR_MESSAGE =   "errorMessage";
    public final static String ERROR_MESSAGE_TEXT =   "Неверный логин или пароль, заполните все поля";
    public final static String AUTHENTICATION_ERROR_TEXT = "Неверный логи или пароль!!";
    public final static String REGISTER_ERROR_MESSAGE_IF_EXIST ="Выберите другое  имя, такой пользователь существет";
    public final static String REGISTER_ERROR ="errorRegister";
    public final static String COMMAND_WELCOME = "/controller?command=welcome";

//REGISTER JSP
    public final static  String NAME_FOR_REGISTER =   "newLoginName";
    public final static String PASSWORD_FOR_REGISTER = "newPassword";
}
```

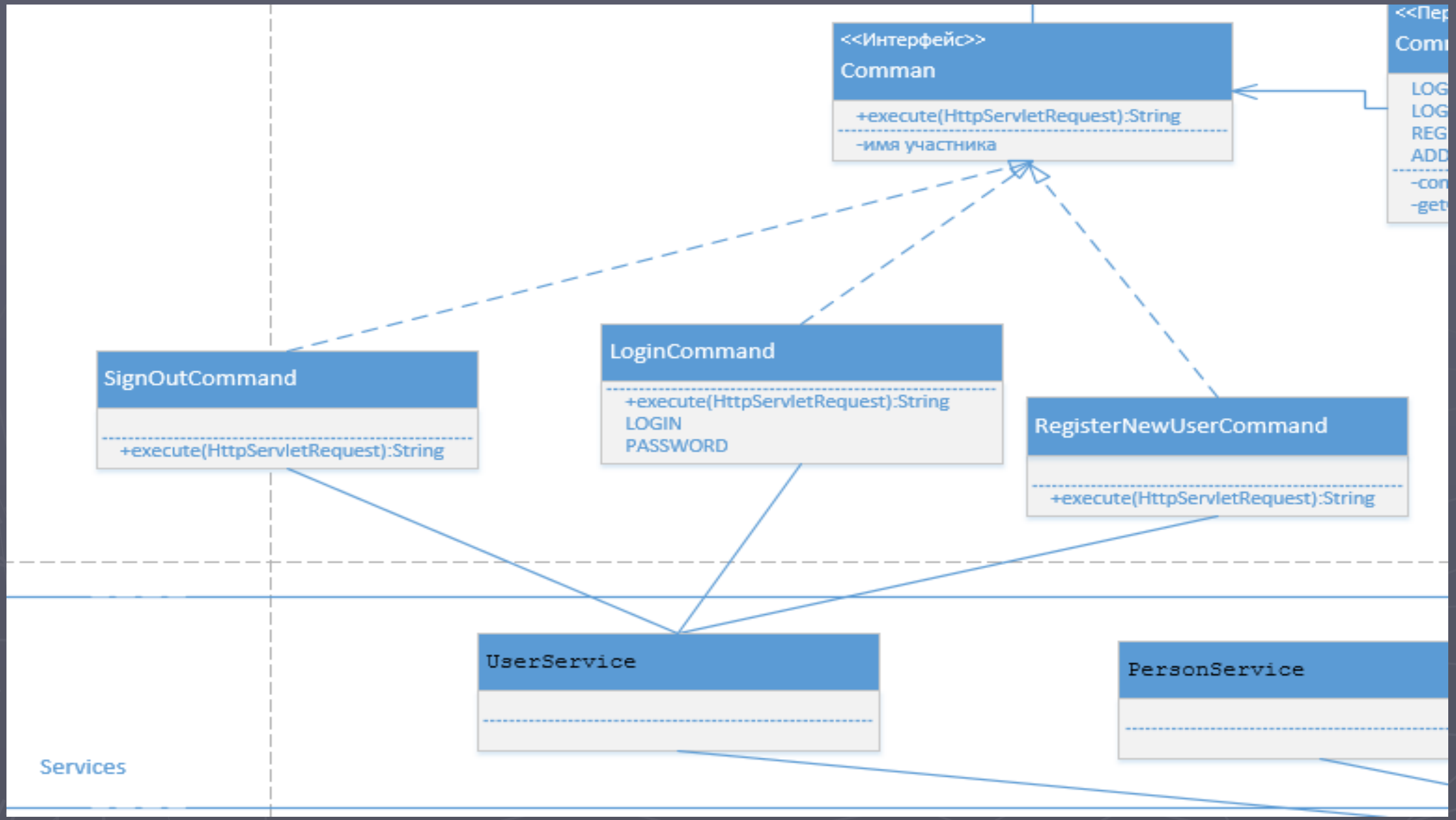
```java
public class SingOutCommand implements Command {

    private static final Logger LOGGER = Logger.getLogger(SingOutCommand.class.getName());

    @Override
    public CommandResult execute(HttpServletRequest request, HttpServletResponse response) throws ServiceException, IncorrectDataException {
        HttpSession session = request.getSession();
        String username  = (String)session.getAttribute(SessionAttribute.NAME);
        LOGGER.info("user was above: name:" + username);
        session.removeAttribute(SessionAttribute.NAME);
        return new CommandResult(Page.LOGIN_PAGE.getPage(), false);
    }
}
```

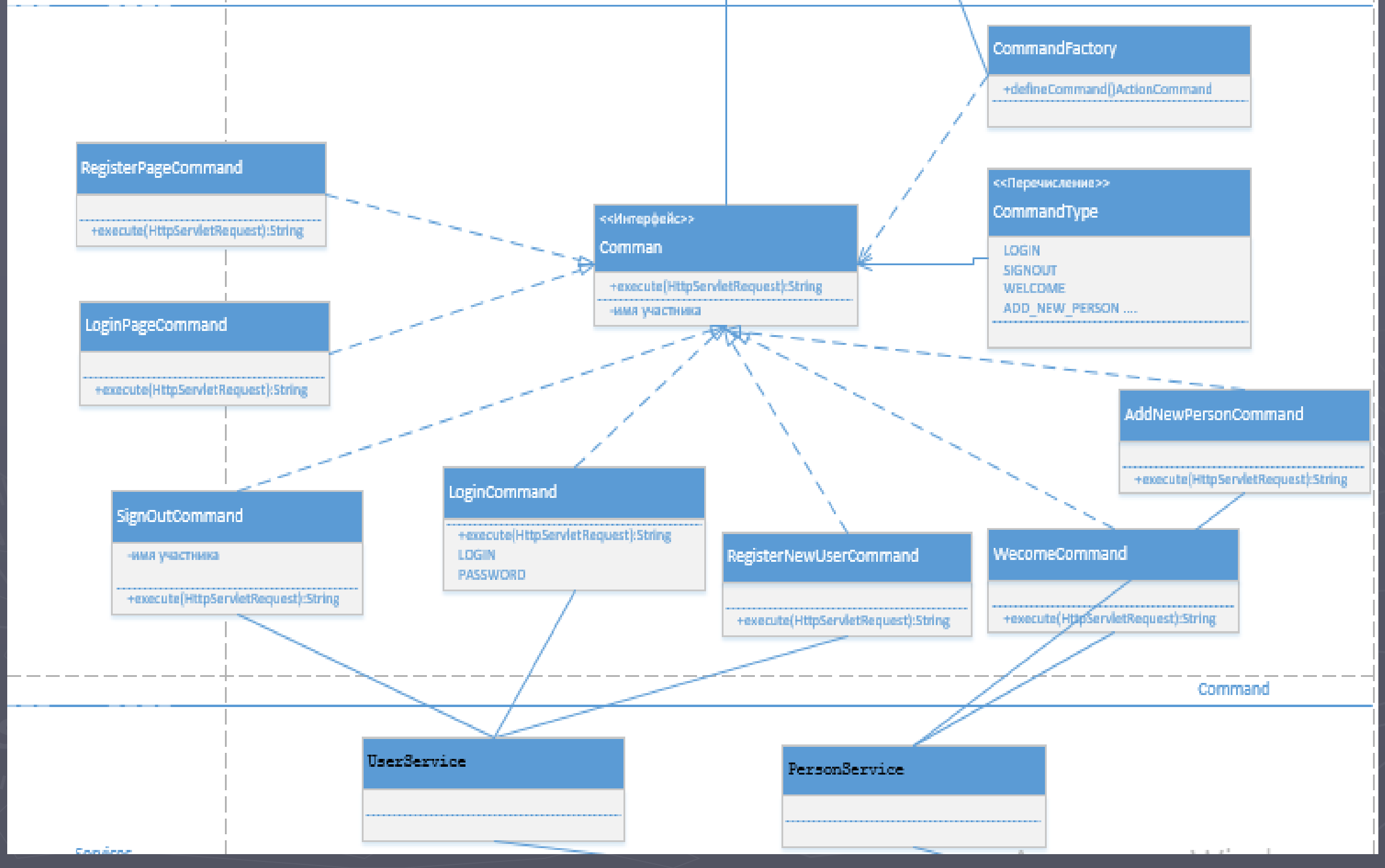
```java
public class LoginCommand implements Command {

    private static final Logger LOGGER = Logger.getLogger(LoginCommand.class.getName());

    private void setAttributesToSession(String name, HttpServletRequest request) {
        HttpSession session = request.getSession();
        session.setAttribute(SessionAttribute.NAME, name);
    }

    @Override
    public CommandResult execute(HttpServletRequest request, HttpServletResponse response) throws ServiceException, IncorrectDataException, ServletException, IOExc
        boolean isUserFind = false;
        Optional<String> login = of(request)
                .map(httpServletRequest -> httpServletRequest.getParameter(LOGIN));
        Optional<String> password = of(request)
                .map(httpServletRequest -> httpServletRequest.getParameter(PASSWORD));
        if (isEmpty(login.get()) || isEmpty(password.get())) {
            return forwardLoginWithError(request, ERROR_MESSAGE, ERROR_MESSAGE_TEXT);
        }
        byte[] pass = HashPassword.getHash(password.get());
        isUserFind = initializeUserIfExist(login.get(), pass, request);
        if (!isUserFind) {
            LOGGER.info("user with such login and password doesn't exist");
            return forwardLoginWithError(request, ERROR_MESSAGE, AUTHENTICATION_ERROR_TEXT);
        } else {
            LOGGER.info("user has been authorized: login:" + login + " password:" + password);
            return new CommandResult(COMMAND_WELCOME, false);
        }
    }

    public boolean initializeUserIfExist(String login, byte[] password, HttpServletRequest request) throws  ServiceException {
        UserService userService = new UserService();
        Optional<User> user = userService.login(login, password);
        boolean userExist = false;
        if (user.isPresent()) {
            setAttributesToSession(user.get().getLogin(), request);
            userExist = true;
        }
        return userExist;
    }

    private CommandResult forwardLoginWithError(HttpServletRequest request, final String ERROR, final String ERROR_MESSAGE) {
        request.setAttribute(ERROR, ERROR_MESSAGE);
```



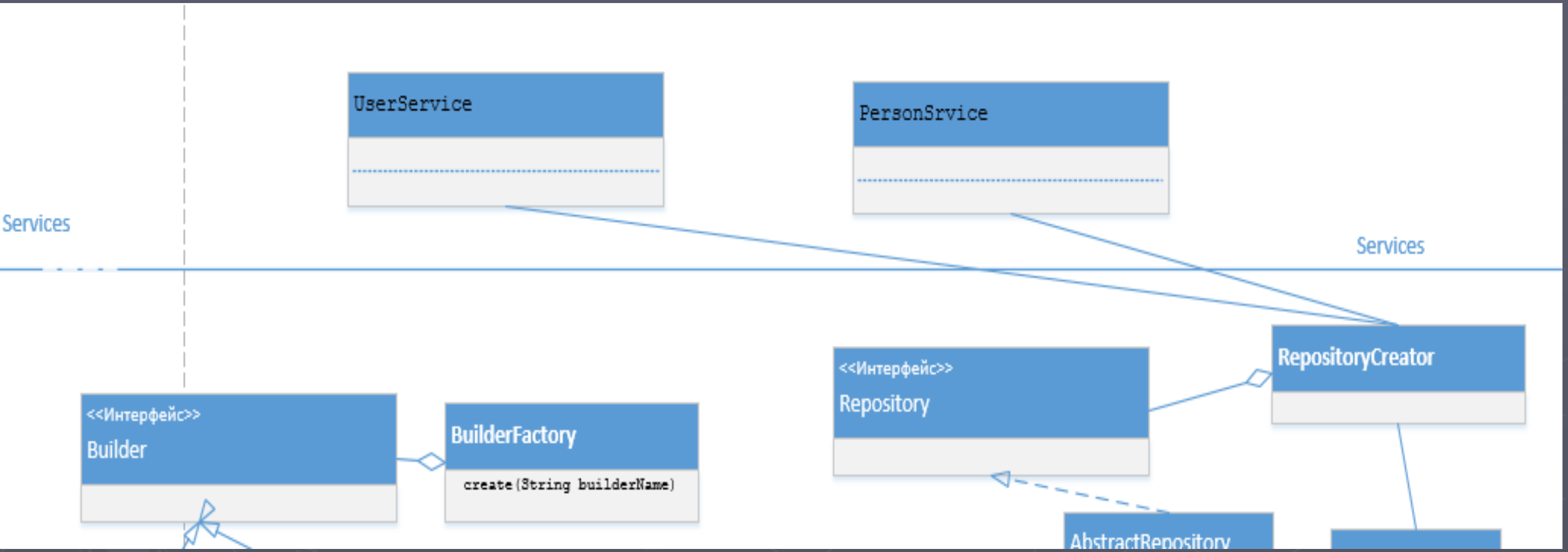
```java
public class RegisterNewUserCommand implements Command {

    private static final Logger LOGGER = Logger.getLogger(RegisterNewUserCommand.class.getName());

    private CommandResult forwardToRegisterWithError(HttpServletRequest request, String ERROR, String ERROR_MESSAGE) {
        request.setAttribute(ERROR, ERROR_MESSAGE);
        return new CommandResult(Page.REGISTER_PAGE.getPage(), false);
    }

    private CommandResult forwardToLogin(HttpServletRequest request) {
        return new CommandResult(Page.LOGIN_PAGE.getPage(), false);
    }

    @Override
    public CommandResult execute(HttpServletRequest request, HttpServletResponse response) throws ServiceException, IncorrectDataException {
        Optional<String> login = of(request)
                .map(httpServletRequest -> httpServletRequest.getParameter(NAME_FOR_REGISTER));
        Optional<String> password = of(request)
                .map(httpServletRequest -> httpServletRequest.getParameter(PASSWORD_FOR_REGISTER));
        if (isEmpty(login.get()) || isEmpty(password.get())) {
            LOGGER.info("invalid login or password format was received:" + login + " " + password);
            return forwardToRegisterWithError(request, REGISTER_ERROR, ERROR_MESSAGE_TEXT);
        }
        byte[] pass = HashPassword.getHash(password.get());
        User user = new User(login.get(), pass);
        UserService userService = new UserService();
        int userCount = userService.save(user);
        if (userCount != 0) {
            LOGGER.info("user was registered: login:" + login + " password:" + password);
            return forwardToLogin(request);
        } else {
            LOGGER.info("invalid login or password format was received:" + login + " " + password);
            return forwardToRegisterWithError(request, REGISTER_ERROR, REGISTER_ERROR_MESSAGE_IF_EXIST);
        }
    }
}
```

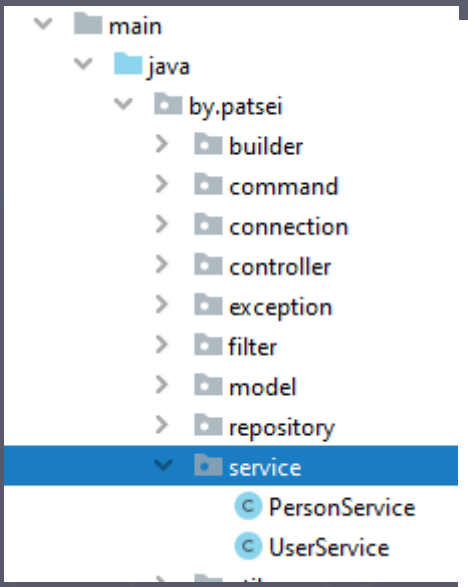



<<Интерфейс>>
Comman
+execute(HttpServletRequest):String
-имя участника
SignOutCommand
+execute(HttpServletRequest):String
LoginCommand
+execute(HttpServletRequest):String
LOGIN
PASSWORD
RegisterNewUserCommand
+execute(HttpServletRequest):String
UserService
PersonService
Services


CommandFactory
+defineCommand():ActionCommand
RegisterPageCommand
+execute(HttpServletRequest):String
<<Интерфейс>>
Comman
+execute(HttpServletRequest):String
-имя участника
<<Перечисление>>
CommandType
LOGIN
SIGNOUT
WELCOME
ADD_NEW_PERSON ....
LoginPageCommand
+execute(HttpServletRequest):String
AddNewPersonCommand
+execute(HttpServletRequest):String
SignOutCommand
-имя участника
+execute(HttpServletRequest):String
LoginCommand
+execute(HttpServletRequest):String
LOGIN
PASSWORD
RegisterNewUserCommand
+execute(HttpServletRequest):String
WecomeCommand
+execute(HttpServletRequest):String
Command
UserService
PersonService

4. УРОВНЬ СЕРВИСОВ

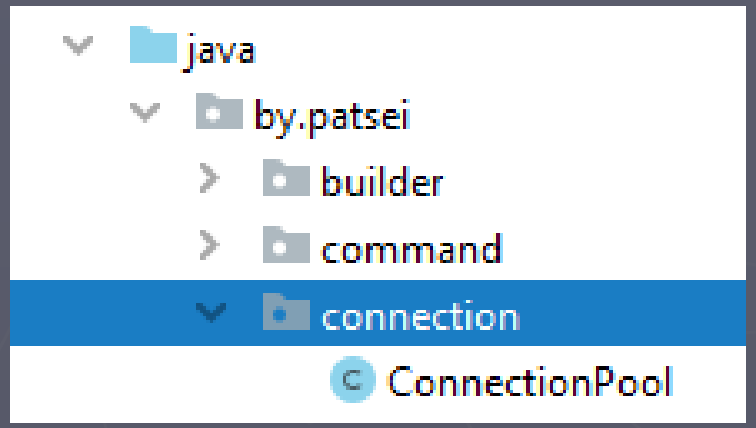


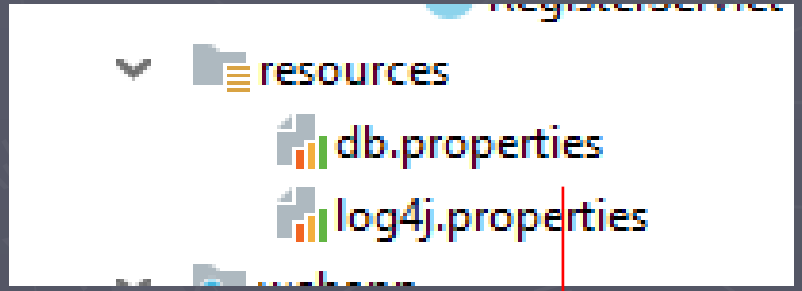


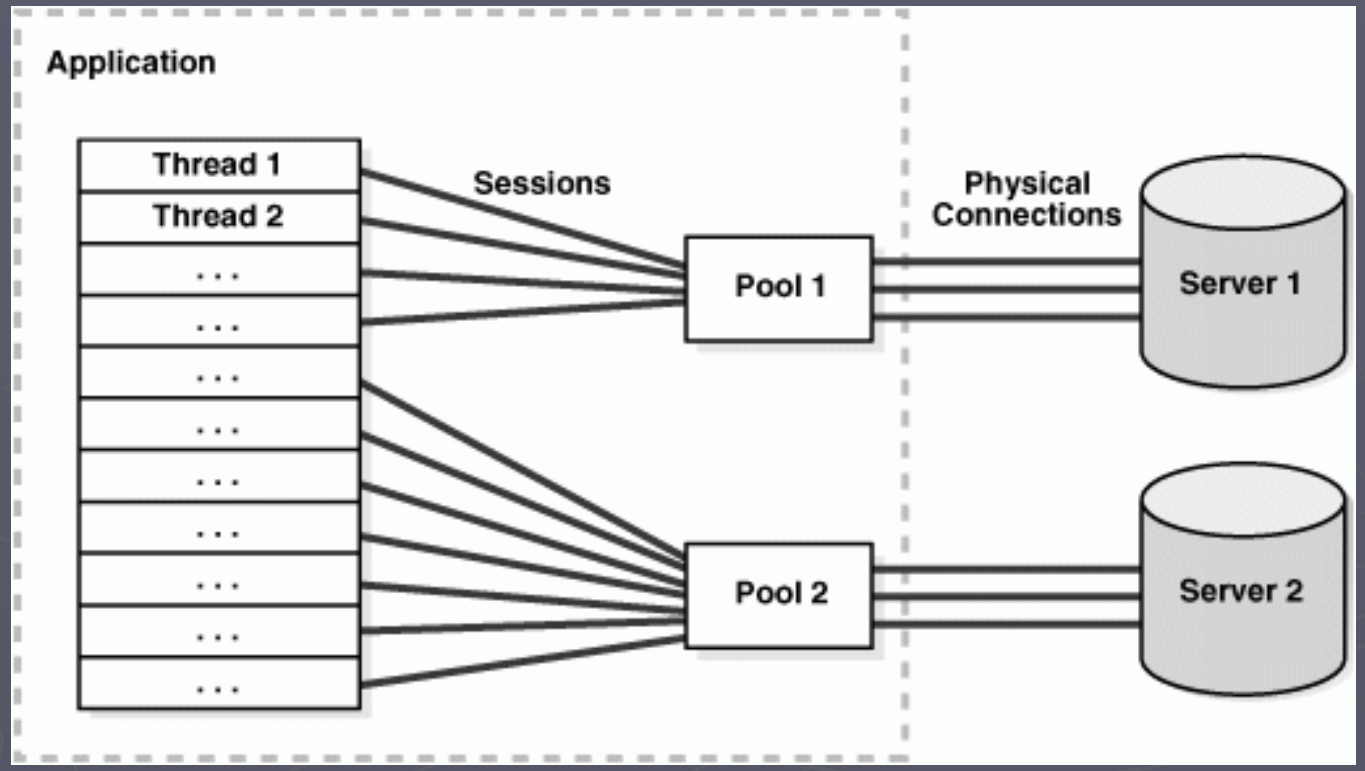

```java
// Service with user - login, save user
import java.util.Optional;

public class UserService {

    public Optional<User> login(String login, byte[] password) throws ServiceException {
        try (RepositoryCreator repositoryCreator = new RepositoryCreator()) {

            UserRepository userRepository = repositoryCreator.getUserRepository();
            UserByLoginPassword params = new UserByLoginPassword(login, password);
            return userRepository.queryForSingleResult(SQLHelper.SQL_GET_USER, params);

        } catch (RepositoryException e) {
            throw new ServiceException(e.getMessage(), e);
        }
    }
    public Integer save(User user) throws ServiceException {
        try (RepositoryCreator repositoryCreator = new RepositoryCreator()) {

            UserRepository userRepository = repositoryCreator.getUserRepository();
            UserByLogin param = new UserByLogin(user.getLogin());
            if (!userRepository.queryForSingleResult(SQLHelper.SQL_CHECK_LOGIN, param).isPresent()) {
                return userRepository.save(user);
            } else {
                return 0;
            }
        } catch (RepositoryException exception) {
            throw new ServiceException(exception.getMessage(), exception);
        }
    }
}
```

```
public class PersonService {

    public List<Person> findAll() throws ServiceException {
        try (RepositoryCreator repositoryCreator = new RepositoryCreator()) {

            PersonRepository personRepository = repositoryCreator.getPersonRepository();
            return personRepository.findAll();
        } catch (RepositoryException e) {
            throw new ServiceException(e.getMessage(), e);
        }
    }

    public void save(Person person) throws ServiceException {
        try (RepositoryCreator repositoryCreator = new RepositoryCreator()) {

            PersonRepository personRepository = repositoryCreator.getPersonRepository();
            personRepository.save(person);
        } catch (RepositoryException exception) {
            throw new ServiceException(exception.getMessage(), exception);
        }
    }
}
```

5. Пул соединений (Connection pool)







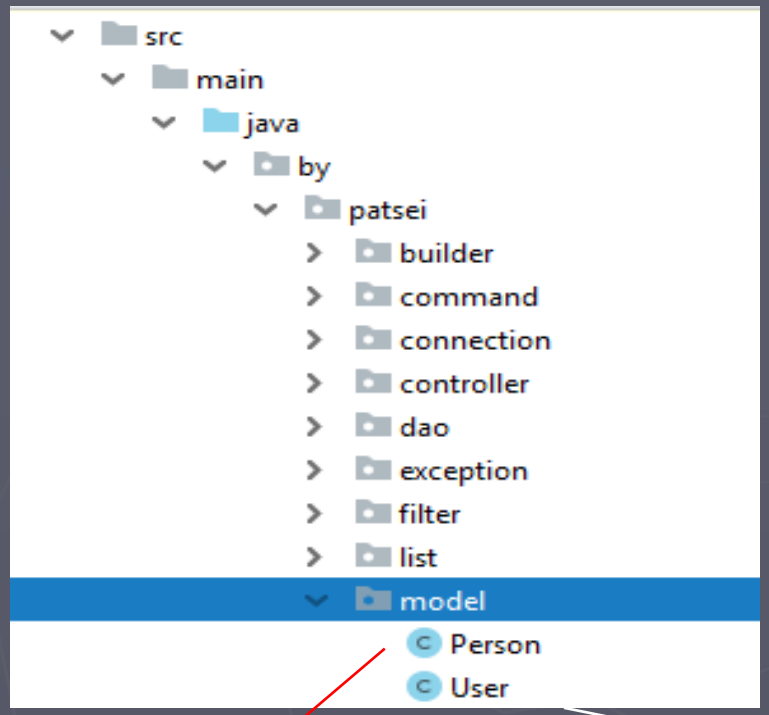

```
db.driver   = com.mysql.jdbc.Driver
db.user = root
db.password = root
db.poolsize = 5
db.url  = jdbc:mysql://localhost:3306/infodb?useSSL=false
db.useUnicode = true
db.encoding = UTF-8
```

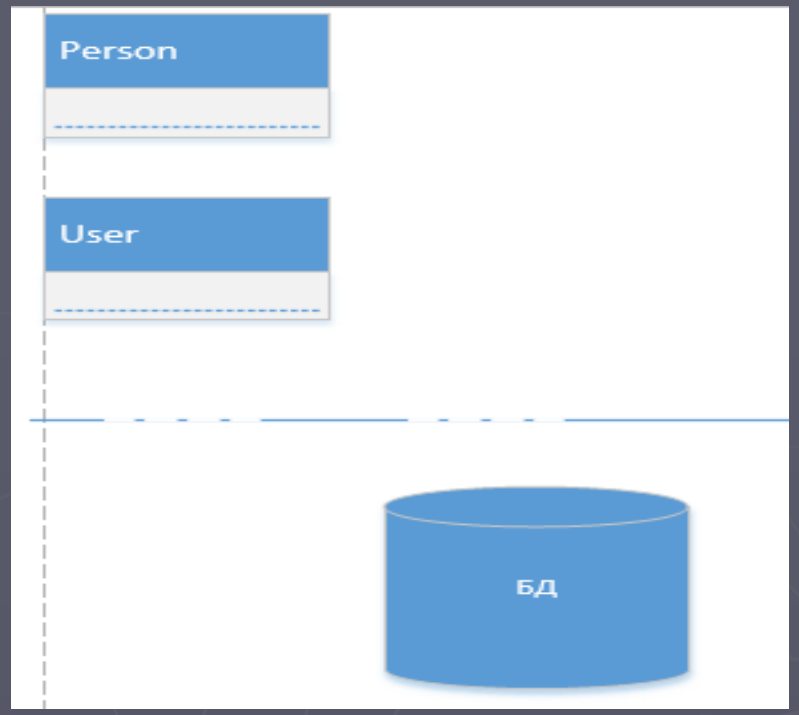
```java
public class ConnectionPool {

    private static final Logger LOGGER = Logger.getLogger(ConnectionPool.class);
    private static final String PROPERTY_PATH = "db";
    private static final int INITIAL_CAPACITY = 10;
    private ArrayBlockingQueue<Connection> freeConnections = new ArrayBlockingQueue<>(INITIAL_CAPACITY);
    private ArrayBlockingQueue<Connection> releaseConnections = new ArrayBlockingQueue<>(INITIAL_CAPACITY);
    private static ReentrantLock lock = new ReentrantLock();
    private volatile static ConnectionPool connectionPool;


    public static ConnectionPool getInstance() {
        try {
            lock.lock();
            if (connectionPool == null) {
                connectionPool = new ConnectionPool();
            }
        } catch (Exception e) {
            LOGGER.error("Can not get Instance", e);
            throw new RuntimeException("Can not get Instance", e);
        } finally {
            lock.unlock();
        }
        return connectionPool;
    }

    private ConnectionPool() throws SQLException {
        try {
            lock.lock();
            if (connectionPool != null) {
                throw new UnsupportedOperationException();
            } else {
                DriverManager.registerDriver(new Driver());
                init();
            }
        } finally {
            lock.unlock();
        }
    }
```

```
private void init() {
    Properties properties = new Properties();
    ResourceBundle resource = ResourceBundle.getBundle(PROPERTY_PATH, Locale.getDefault());
    if (resource == null) {
        LOGGER.error("Error while reading properties");
    } else {
        String connectionURL = resource.getString("db.url");
        String initialCapacityString = resource.getString("db.poolsize");
        String user = resource.getString("db.user");
        String pass = resource.getString("db.password");
        Integer initialCapacity = Integer.valueOf(initialCapacityString);

        for (int i = 0; i < initialCapacity; i++) {
            try {
                Connection connection = DriverManager.getConnection(connectionURL, user, pass);
                freeConnections.add(connection);
            } catch (SQLException e) {
                LOGGER.error("Pool can not initialize", e);
                throw new RuntimeException("Pool can not initialize", e);
            }
        }
    }
}
public Connection getConnection() {
    try {
        Connection connection = freeConnections.take();
        releaseConnections.offer(connection);
        LOGGER.info("Connection was taken, the are free connection " + freeConnections.size());
        return connection;
    } catch (InterruptedException e) {
        throw new RuntimeException("Can not get database", e);
    }
}
```

```
    public void releaseConnection(Connection connection) {
        releaseConnections.remove(connection);
        freeConnections.offer(connection);
        LOGGER.info("Connection was released, the are free connection " + freeConnections.size());
    }

    public void destroy() {
        for (int i = 0; i < freeConnections.size(); i++) {
            try {
                Connection connection = (Connection) freeConnections.take();
                connection.close();
            } catch (InterruptedException e) {
                LOGGER.error("Connection close exception", e);
            } catch (SQLException e) {
                LOGGER.error("database is not closed", e);
                throw new RuntimeException("database is not closed", e);
            }
        }
        try {
            Enumeration<java.sql.Driver> drivers = DriverManager.getDrivers();
            while (drivers.hasMoreElements()) {
                java.sql.Driver driver = drivers.nextElement();
                DriverManager.deregisterDriver(driver);
            }
        } catch (SQLException e) {
            LOGGER.error("Drivers were not deregistrated", e);
        }
    }
}
```

6. Model

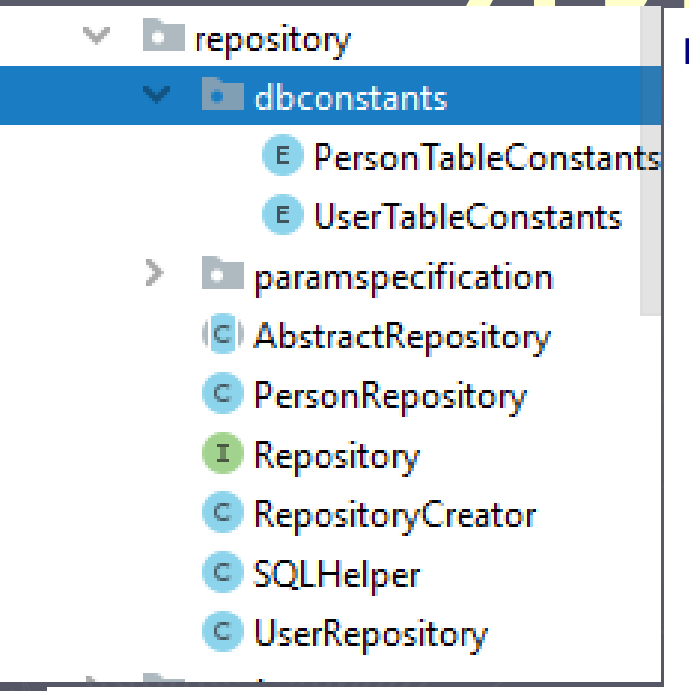




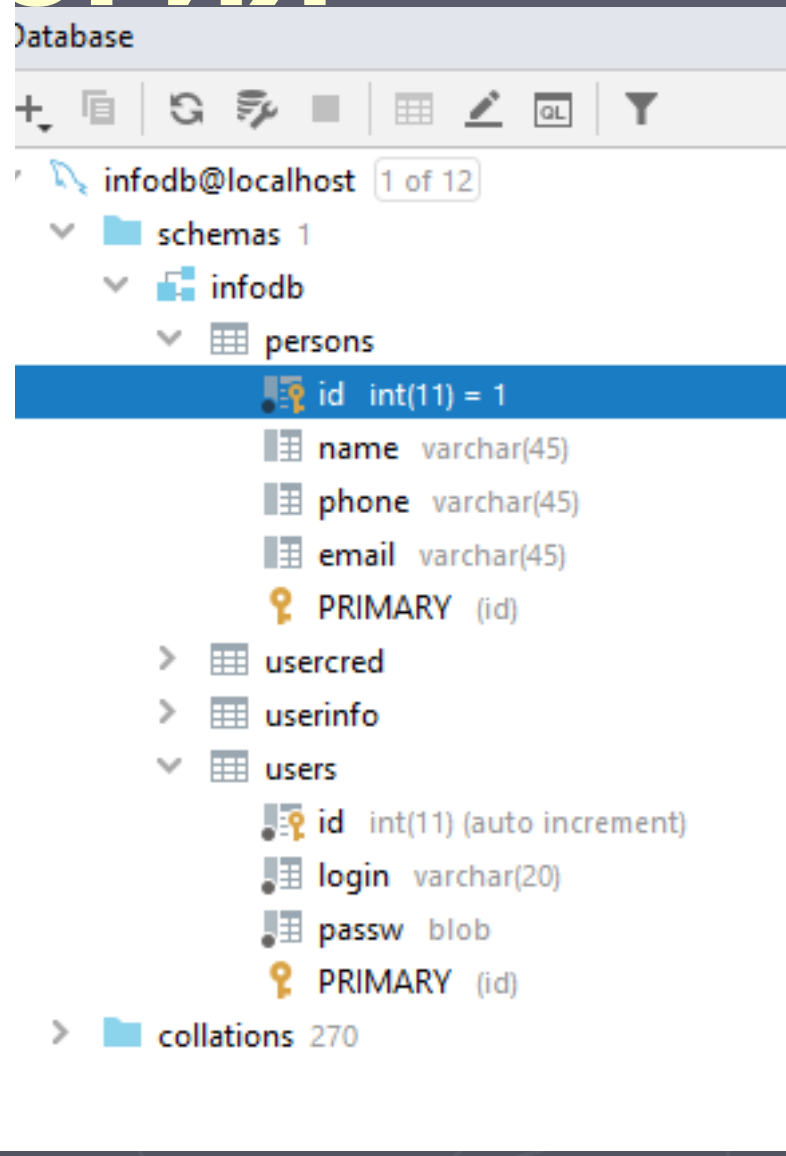

```
public class Person implements Serializable {
        private int id;
        private String name;
        private String phone;
        private String email;
...
```

```
public class User implements Serializable {

    private int id;
    private String login;
    private byte[] passw;
...
```

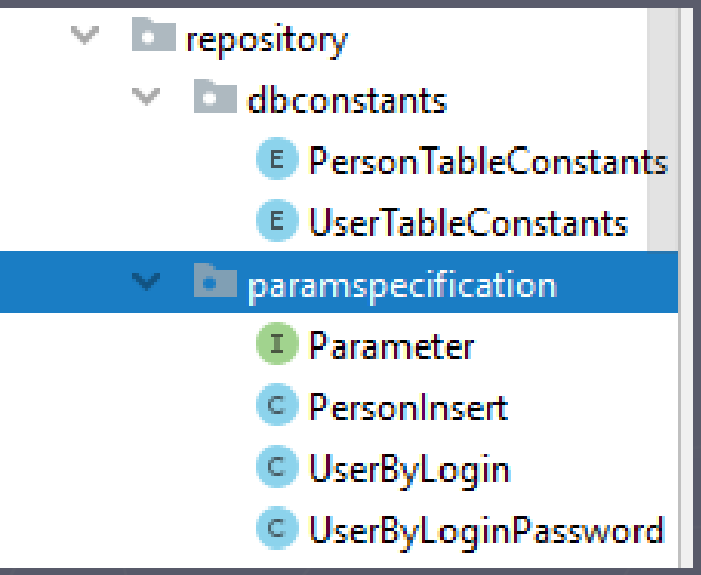
7. УРОВЕНЬ РЕПОЗИТОРИЯ




```java
public enum PersonTableConstants {
    ID("id"),
    NAME("name"),
    PHONE("phone"),
    EMAIL("email");

    private String fieldName;

    private PersonTableConstants(String fieldName) {
        this.fieldName = fieldName;
    }

    public String getFieldName() {
        return fieldName;
    }
```




```java
public enum UserTableConstants {

    ID("id"),
    LOGIN("login"),
    PASSWORD("passw");

    private String fieldName;

    private UserTableConstants(String fieldName) {
        this.fieldName = fieldName;
    }

    public String getFieldName() {
        return fieldName;
    }
}
```

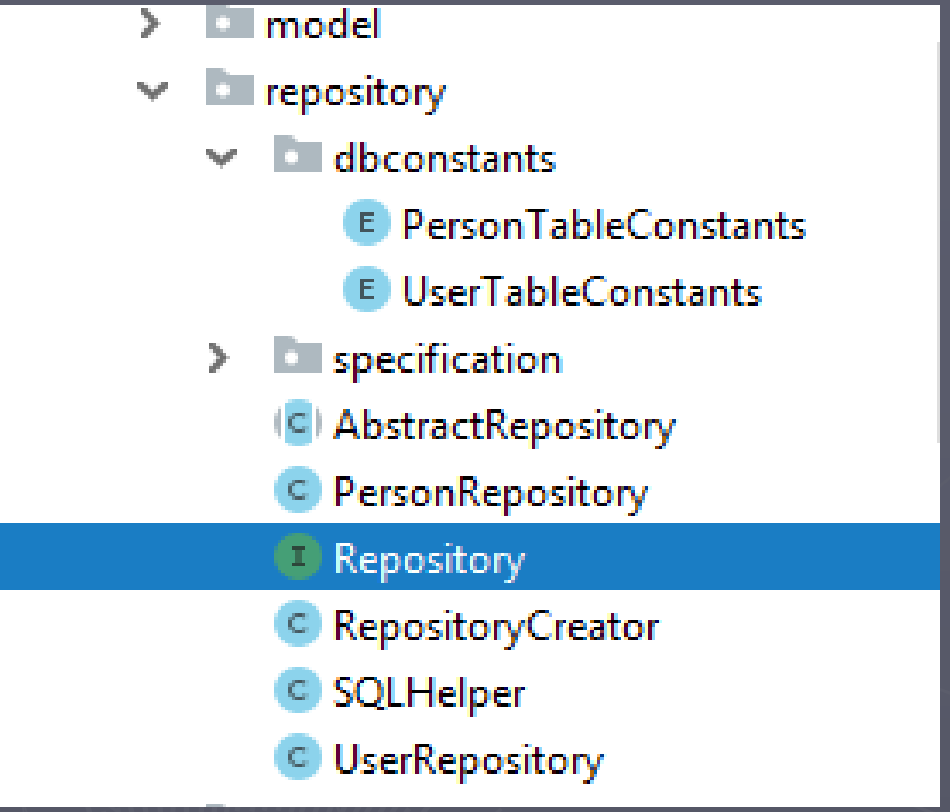



```java
public interface Parameter {
    List<Object> getParameters();
}
```

```java
public class UserByLogin implements Paramet

    private String login;

    public UserByLogin(String login) {
        this.login = login;
    }

    @Override
    public List<Object> getParameters() {
        return Arrays.asList(login);
    }
}
```

```java
public class UserByLoginPassword implements Parameter {

    private String login;
    private byte [] password;

    public UserByLoginPassword(String login, byte [] password) {
        this.login = login;
        this.password = password;
    }

    @Override
    public List<Object> getParameters() {
        return  Arrays.asList(login, password);
    }
}
```

```
public class SQLHelper {

    public static final String ID = "id";
    private static final String INSERT_QUERY = "INSERT INTO ";
    private static final String VALUES = "VALUES";
    private static final String WHERE = "WHERE ";
    private static final String SELECT = "SELECT";
    public static final String USER_TABLE = "users";
    public static final String PERSON_TABLE = "persons";

    public final static String SQL_GET_PERSONS = "select * from " + USER_TABLE;
    public final static String SQL_INSERT_PERSON = "INSERT INTO " + PERSON_TABLE + "(" + PersonTableConstants.NAME +
            "," + PersonTableConstants.PHONE + "," + PersonTableConstants.EMAIL + ") VALUES (? , ?, ?)";
    public final static String SQL_GET_USER = "SELECT " + UserTableConstants.ID.getFieldName() + ", " +
            UserTableConstants.LOGIN.getFieldName() + ", " +
            UserTableConstants.PASSWORD.getFieldName() + " from " + USER_TABLE + " WHERE " +
            UserTableConstants.LOGIN.getFieldName() + " =? and " +
            UserTableConstants.PASSWORD.getFieldName() + " =?";
    public final static String SQL_CHECK_LOGIN = "SELECT  " + UserTableConstants.LOGIN.getFieldName() + " FROM " +
            USER_TABLE + " WHERE " + UserTableConstants.LOGIN.getFieldName() + " = ?";
    public final static String SQL_INSERT_USER = "INSERT INTO " + USER_TABLE + "(" +
            UserTableConstants.LOGIN.getFieldName() + " ," +
            UserTableConstants.PASSWORD.getFieldName() + ") VALUES (? , ?)";


    public static String makeInsertQuery(Map<String, Object> fields, String table) {
        StringBuilder columns = new StringBuilder("(");
        StringBuilder values = new StringBuilder("(");

        for (Map.Entry<String, Object> entry : fields.entrySet()) {
            String column = entry.getKey();
            if (column.equals(ID)) {
                continue;
            }
            columns.append(column).append(", ");
            values.append("?, ");
        }

        values.deleteCharAt(values.lastIndexOf(","));
        columns.deleteCharAt(columns.lastIndexOf(","));
        values.append(")");
        columns.append(")");
```




```
public interface Repository <T> {
        List<T> query(String sqlString, Parameter parameter) throws RepositoryException;
        Optional<T> queryForSingleResult(String sqlString, Parameter parameter) throws Repositor
        List<T> findAll() throws RepositoryException;
        Integer save(T object) throws RepositoryException;
}
```

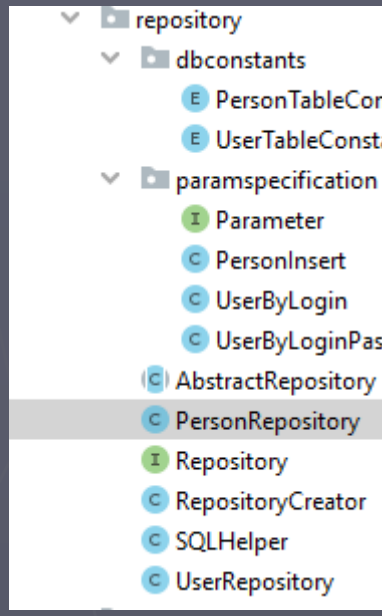
```java
public abstract class AbstractRepository<T> implements Repository<T> {

    private Connection connection;

    private static final Logger LOGGER = Logger.getLogger(AbstractRepository.class);
    private static final String GET_ALL_QUERY = "SELECT * FROM ";
    private  final String WHERE_ID_CONDITION = " WHERE id_" + getTableName() + "=(?)";
    protected  final String DELETE_QUERY = "DELETE from " + getTableName() + " where id_" + getTableName() +

    protected abstract String getTableName();

    AbstractRepository(Connection connection) {
        this.connection = connection;
    }
// Prepare request with params

    public static void prepare(PreparedStatement preparedStatement, List<Object> parameters) throws SQLExcept
        int length = parameters.size();
        for (int i = 0; i < length; i++) {
            if (parameters.get(i) == null) {
                preparedStatement.setNull(i + 1, getType(parameters.get(i)));
            } else {
                preparedStatement.setObject(i + 1, parameters.get(i));
            }
        }
    }
```

```java
public static void prepare(PreparedStatement preparedStatement, Map<String, Object> fields, String tableName) throws SQLException {
    int i = 1;
    for (Map.Entry<String, Object> entry : fields.entrySet()) {
        Object value = entry.getValue();
        String key = entry.getKey();
        if (!key.equals(SQLHelper.ID)) {
            if (value == null) {
                preparedStatement.setNull(i++, getType(value));
            } else {
                preparedStatement.setObject(i++, value);
            }
        }
    }
    Object id = fields.get(SQLHelper.ID);
    if (id != null) {
        preparedStatement.setString(i++, String.valueOf(id));
    }
}

List<T> executeQuery(String sql, Builder<T> builder, List<Object> parameters) throws RepositoryException {
    List<T> objects = new ArrayList<>();
    try {
        PreparedStatement preparedStatement = connection.prepareStatement(sql);
        prepare(preparedStatement, parameters);
        ResultSet resultSet = preparedStatement.executeQuery();
        while (resultSet.next()) {
            T item = builder.build(resultSet);
            objects.add(item);
        }
    } catch (SQLException e) {
        throw new RepositoryException(e.getMessage(), e);
    }
    return objects;
}
```

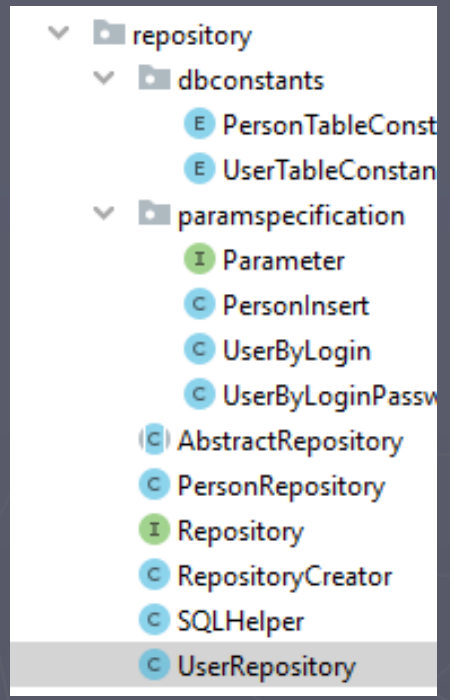
```java
protected Optional<T> executeQueryForSingleResult(String query, Builder<T> builder, List<Object> parameters
    List<T> items = executeQuery(query, builder, parameters);
    return items.size() == 1 ?
            Optional.of(items.get(0)) :
            Optional.empty();
}

protected abstract Map<String, Object> getFields(T obj);

@Override
public Integer    save(T object) throws RepositoryException {
    String sql;
    Map<String, Object> fields = getFields(object);
    sql = SQLHelper.makeInsertQuery(fields, getTableName());
    return executeSave(sql, fields);
}
```

```
 private Integer executeSave(String query, Map<String, Object> fields) throws RepositoryException {
        try {
            PreparedStatement preparedStatement = connection.prepareStatement(query, Statement.RETURN_GENERAT
            prepare(preparedStatement, fields, getTableName());
            LOGGER.info(preparedStatement.toString());
            preparedStatement.executeUpdate();
            ResultSet resultSet = preparedStatement.getGeneratedKeys();
            Integer generatedId = null;
            while (resultSet.next()) {
                generatedId = resultSet.getInt(1);
            }
            return generatedId;
        } catch (SQLException e) {
            throw new RepositoryException(e.getMessage(), e);
        }
    }

    @Override
    public List<T> findAll() throws RepositoryException {
        Builder builder = BuilderFactory.create(getTableName());
        String query = GET_ALL_QUERY + getTableName();
        return executeQuery(query, builder, Collections.emptyList());
    }
}
```

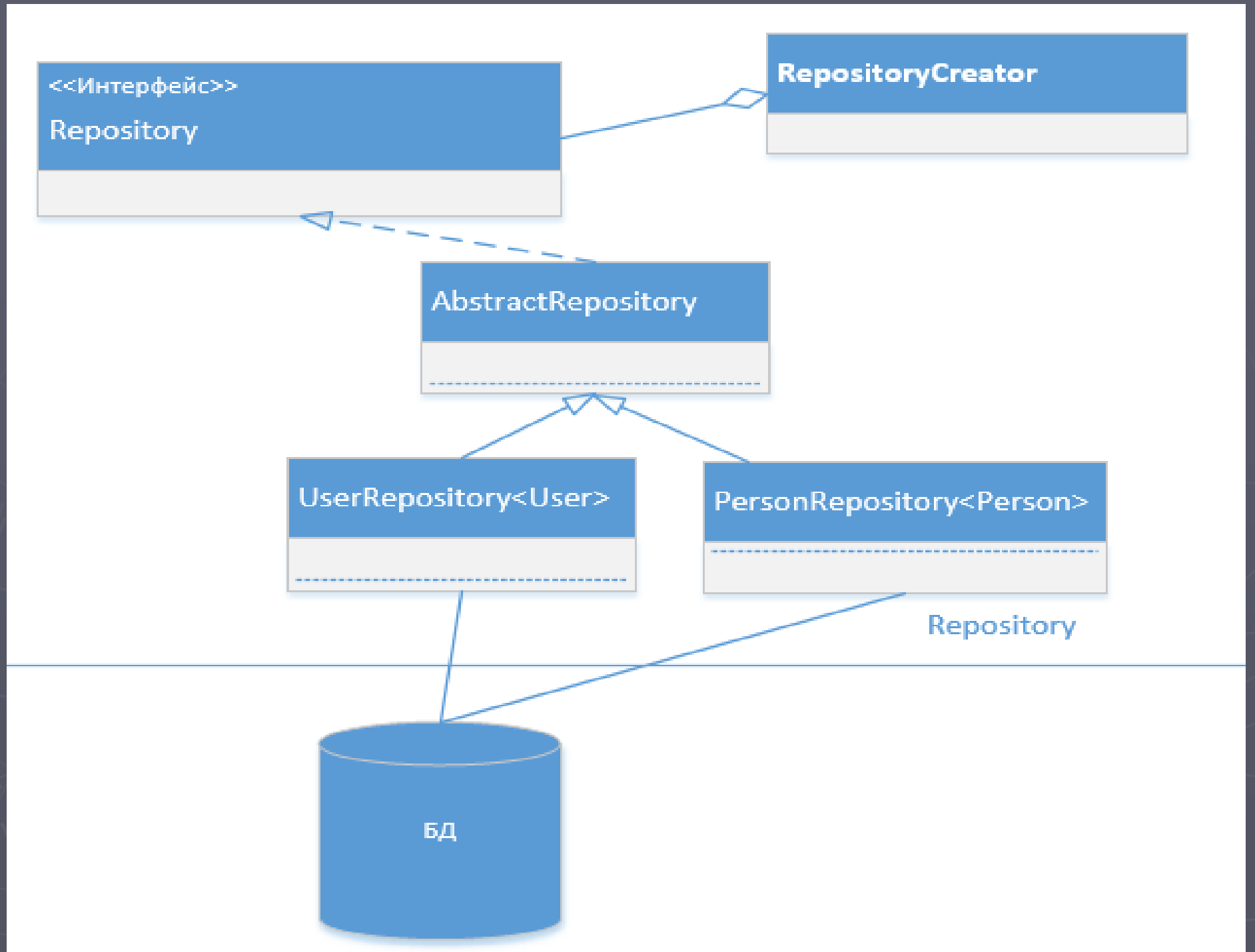
```java
public class RepositoryCreator implements AutoCloseable {
    private ConnectionPool connectionPool;
    private Connection connection;

    public RepositoryCreator() {
        connectionPool = ConnectionPool.getInstance();
        connection = connectionPool.getConnection();
    }

    public UserRepository getUserRepository() {
        return new UserRepository(connection);
    }

    public PersonRepository getPersonRepository() {
        return new PersonRepository(connection);
    }

    @Override
    public void close() {
        connectionPool.releaseConnection(connection);
    }
}
```



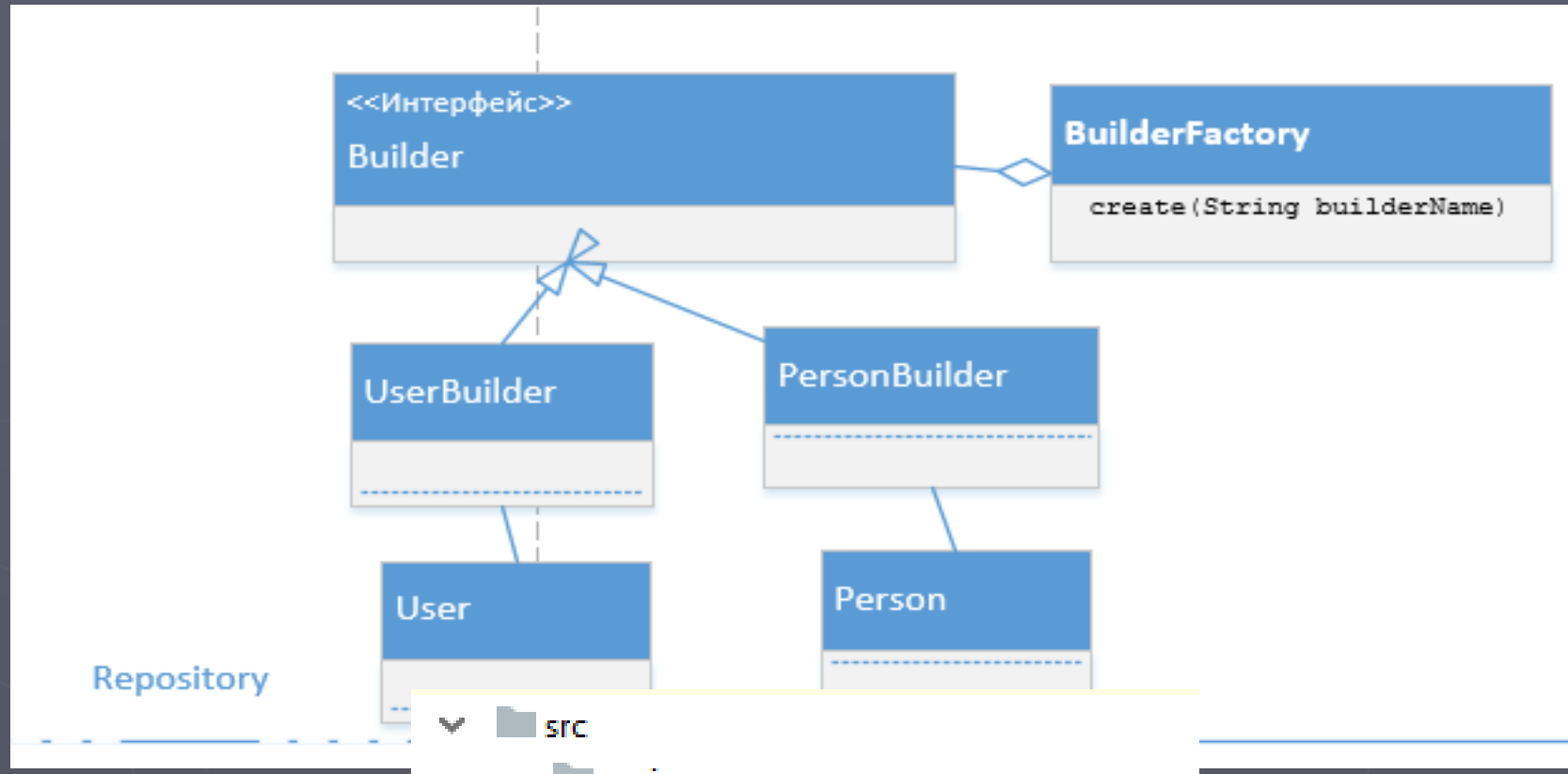

```java
public class PersonRepository extends AbstractRepository<Person> {

    public PersonRepository(Connection connection){

        super(connection);
    }

    @Override
    protected String getTableName() {
        return SQLHelper.PERSON_TABLE ;
    }

    @Override
    public List<Person> query(String sqlString, Parameter paramater) throws RepositoryException {

        List<Person> persons = executeQuery(sqlString,new PersonBuilder(), paramater.getParameters());
        return persons;
    }

    @Override
    public Optional<Person> queryForSingleResult(String sqlString, Parameter parameter) throws RepositoryException {
        List<Person> person = query(sqlString, parameter);
        return person.size() == 1 ?
                Optional.of(person.get(0)) :
                Optional.empty();
    }

    public Map<String,Object> getFields(Person person) {
        Map<String,Object> fields = new HashMap<>();
        fields.put(PersonTableConstants.NAME.getFieldName(), person.getName());
        fields.put(PersonTableConstants.PHONE.getFieldName(), person.getPhone());
        fields.put(PersonTableConstants.EMAIL.getFieldName(), person.getEmail());
        return fields;
    }
}
```

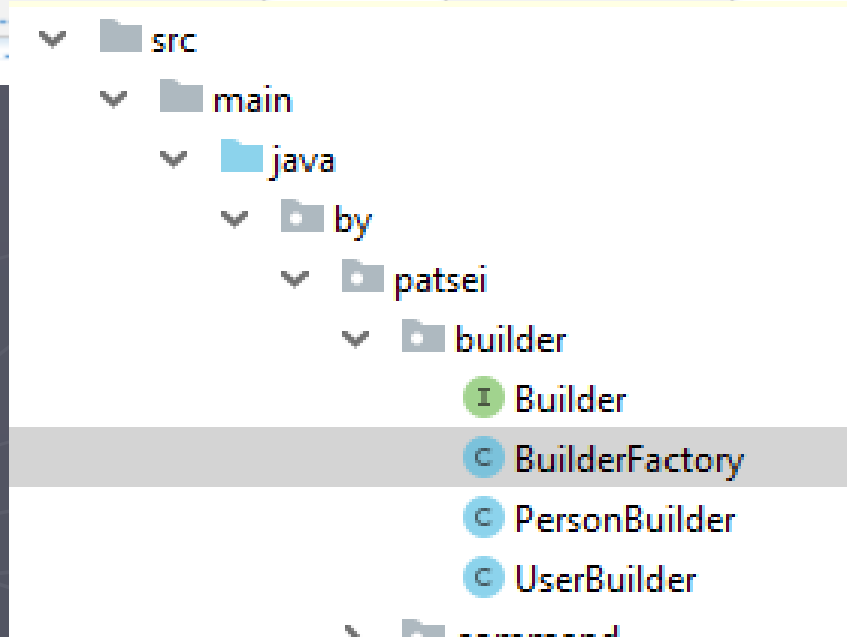



```java
public class UserRepository  extends AbstractRepository <User>{

    public UserRepository(Connection connection){
        super(connection);
    }

    @Override
    protected String getTableName() {
        return SQLHelper.USER_TABLE;
    }


    @Override
    public List<User> query(String sqlString, Parameter paramater) throws RepositoryException {

        List<User> users = executeQuery(sqlString,new UserBuilder(), paramater.getParameters());
        return users;
    }

    @Override
    public Optional<User> queryForSingleResult(String sqlString, Parameter parameter) throws RepositoryException {
        List<User> user = query(sqlString, parameter);
        return user.size() == 1 ?
                Optional.of(user.get(0)) :
                Optional.empty();
    }

    public Map<String,Object> getFields(User user) {
        Map<String,Object> fields = new HashMap<>();
        fields.put(UserTableConstants.LOGIN.getFieldName(), user.getLogin());
        fields.put(UserTableConstants.PASSWORD.getFieldName(), user.getPassw());
        return fields;
    }
}
```


<<Интерфейс>>
Repository
RepositoryCreator
AbstractRepository
UserRepository<User>
PersonRepository<Person>
Repository
БД

8. Builder



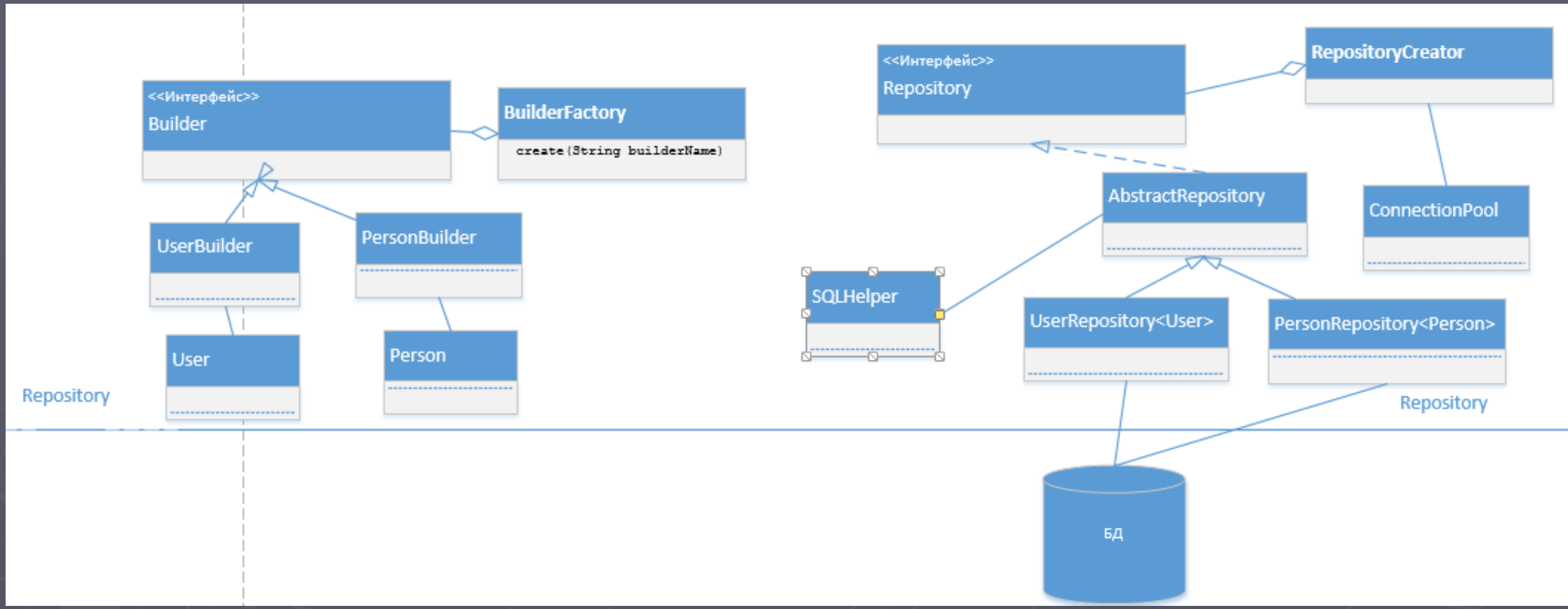


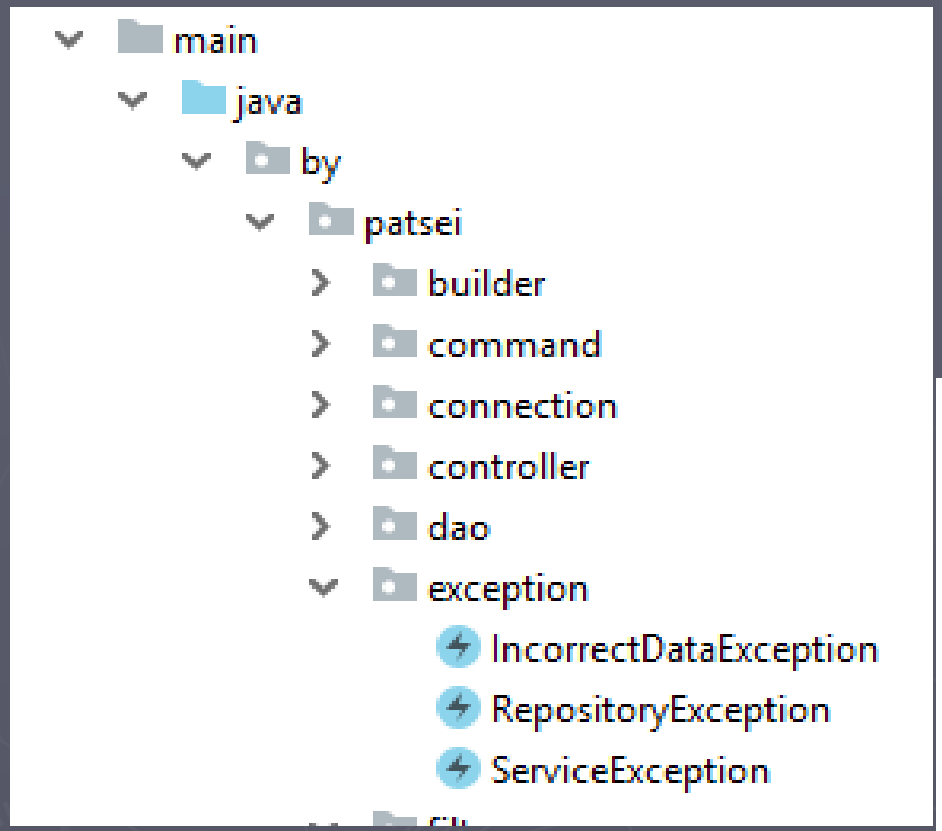

```
public interface Builder <T> {
    T build(ResultSet resultSet) throws
RepositoryException, RepositoryException;
}
```

```java
public class BuilderFactory {

    private static final String USER = "user";
    private static final String PERSON = "persons";
    private static final String MESSAGE= "Unknown Builder name!";

    public static Builder create(String builderName) {
        switch (builderName) {
            case USER: {
                return new UserBuilder();
            }
            case PERSON: {
                return new PersonBuilder();
            }
            default:
                throw new IllegalArgumentException(MESSAGE);
        }
    }
}
```

```java
// ResultSet --> Person
public class PersonBuilder implements Builder <Person>{
    @Override
    public Person build(ResultSet resultSet) throws RepositoryException {
        try {
            int id = resultSet.getInt(PersonTableConstants.ID.getFieldName());
            String name = resultSet.getString(PersonTableConstants.NAME.getFieldName());
            String phone = resultSet.getString(PersonTableConstants.PHONE.getFieldName());
            String email = resultSet.getString(PersonTableConstants.EMAIL.getFieldName());
            return new Person(id, name, phone, email);
        } catch (SQLException exception) {
            throw new RepositoryException(exception.getMessage(), exception);
        }
    }
}
```

```java
// ResultSet--> User

public class UserBuilder implements Builder<User> {
    @Override
    public User build(ResultSet resultSet) throws RepositoryException {
        try {
            int id = resultSet.getInt(UserTableConstants.ID.getFieldName());
            String login = resultSet.getString(UserTableConstants.LOGIN.getFieldName());
            byte[] password = resultSet.getBytes(UserTableConstants.PASSWORD.getFieldName());
            return new User(id, login, password);
        }

        catch (SQLException exception) {
            throw new RepositoryException(exception.getMessage(), exception);
        }
    }
}
```


<<Интерфейс>>
Builder
BuilderFactory
create(String builderName)
UserBuilder
PersonBuilder
User
Person
Repository
<<Интерфейс>>
Repository
RepositoryCreator
AbstractRepository
ConnectionPool
SQLHelper
UserRepository<User>
PersonRepository<Person>
Repository
БД



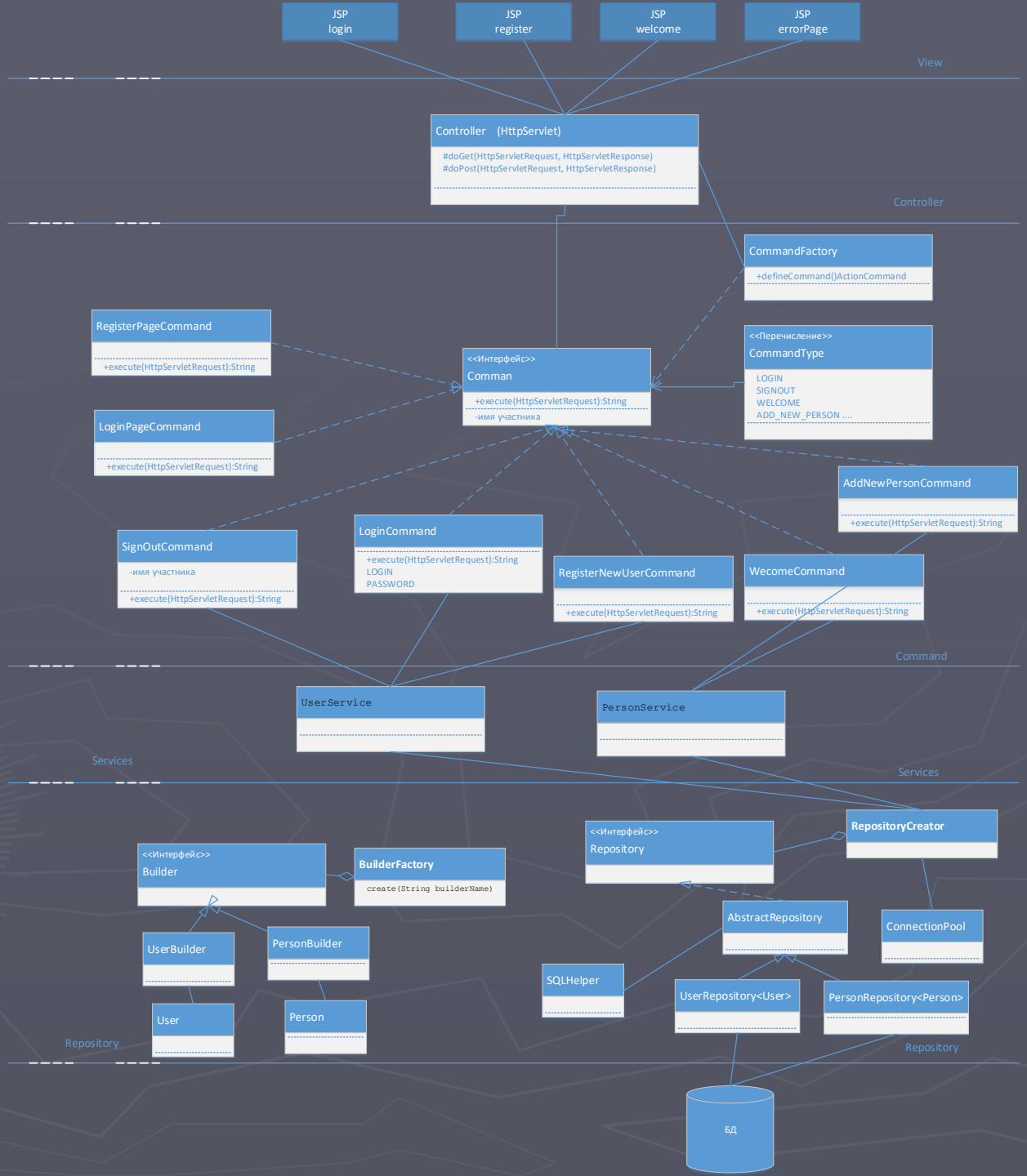
9. ИСКЛЮЧЕНИЯ

```
    public class IncorrectDataException extends
Exception{
    public IncorrectDataException(String
message) {
        super(message);
    }
    public IncorrectDataException(String
message, Throwable cause) {
        super(message, cause);
    }
    public IncorrectDataException(Throwable
cause) {
        super(cause);
    }
}
```

10. FILTER

http://localhost:8080/Servlet_war_exploded/controller?command=welcome

```
@WebFilter(urlPatterns = "/controller")
public class LoginRequiredFilter implements Filter {
    private static final String COMMAND = "command";
    private static final String WELCOME = "welcome";
    private static final String ERROR_MESSAGE = "error_message";
    private static final String ERROR_TEXT = "Нет авторизации для выполнения данной команды";
    private static final Logger LOGGER = Logger.getLogger(LoginRequiredFilter.class.getName());


    public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws ServletException, IOException
        HttpServletRequest request = (HttpServletRequest) req;
        String command = request.getParameter(COMMAND);
        LOGGER.info("Filter is working  " + COMMAND + "= " + command);
        if (!command.equals(WELCOME)) {
            chain.doFilter(req, resp);
        } else {
            if (request.getSession().getAttribute(SessionAttribute.NAME) != null) {
                chain.doFilter(req, resp);
            } else {
                request.setAttribute(ERROR_MESSAGE, ERROR_TEXT);
                request.getRequestDispatcher(Page.ERROR_PAGE.getPage()).forward(req, resp);
            }
        }
    }
}
```


JSP
login
JSP
register
JSP
welcome
JSP
errorPage
View
Controller (HttpServlet)
#doGet(HttpServletRequest, HttpServletResponse)
#doPost(HttpServletRequest, HttpServletResponse)
Controller
CommandFactory
+defineCommand()ActionCommand
RegisterPageCommand
+execute(HttpServletRequest):String
<<Интерфейс>>
Comman
+execute(HttpServletRequest):String
-имя участника
<<Перечисление>>
CommandType
LOGIN
SIGNOUT
WELCOME
ADD_NEW_PERSON ....
LoginPageCommand
+execute(HttpServletRequest):String
AddNewPersonCommand
+execute(HttpServletRequest):String
SignOutCommand
-имя участника
+execute(HttpServletRequest):String
LoginCommand
+execute(HttpServletRequest):String
LOGIN
PASSWORD
RegisterNewUserCommand
+execute(HttpServletRequest):String
WecomeCommand
+execute(HttpServletRequest):String
Command
UserService
PersonService
Services
Services
<<Интерфейс>>
Builder
BuilderFactory
create(String builderName)
<<Интерфейс>>
Repository
RepositoryCreator
UserBuilder
PersonBuilder
AbstractRepository
ConnectionPool
SQLHelper
UserRepository<User>
PersonRepository<Person>
User
Person
Repository
Repository
БД