МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

НГТУ НЭТИ | Факультет прикладной математики и информатики

Кафедра прикладной математики

Лабораторная работа № 3

по дисциплине «Операционные системы и компьютерные сети»

РАЗРАБОТКА ПРИЛОЖЕНИЯ ИНТЕРАКТИВНОЙ ПЕРЕПИСКИ

Бригада 9

Группа ПМ-24 ГЕРАСИМЕНКО ВАДИМ

ПАРАСКУН ИВАН

Преподаватели СИВАК МАРИЯ АЛЕКСЕЕВНА

КОБЫЛЯНСКИЙ ВАЛЕРИЙ ГРИГОРЬЕВИЧ

Новосибирск, 2024

1. Цель работы

Изучить основные принципы разработки многопользовательских приложений, построенных на основе технологии клиент-сервер с использованием стека протоколов TCP/IP.

2. Условие задачи

С помощью API-интерфейса реализовать простой чат. Каждая бригада должна написать chat-сервер и chat-клиента. Сервер должен поддерживать соединение сразу от нескольких клиентов. Обмен между клиентами осуществляется через сервер. При получении сообщения от какого-либо клиента, сервер дублирует его на своем экране и оповещает всех подсоединенных клиентов, отправляя каждому из них данное сообщение. При подсоединении нового клиента к chat-серверу, сервер оповещает каждого клиента о новом пользователе, посылая им его IP-адрес, номер порта и имя. При отключении также необходимо сформировать оповещение.

3. Ход работы

С полным текстом программы можно ознакомиться в приложении.

Основная логика реализации серверной части:

Класс “ChatServer” запускает сервер, принимает соединения и создает для каждого клиента экземпляр класса “Client”:

```
public class ChatServer implements Server {
private final int port;
private final InetAddress inetAddress;
private final List<Client> activeClients;
private boolean isRunning = true;
```

...

```
@Override
public void start() {
    try (ServerSocket serverSocket = new ServerSocket(port)) {
        System.out.println(new InfoMessage(null,
                ServerTime.getTime(), "Server started at " + inetAddress
                + ", port: " + serverSocket.getLocalPort()));

        while (isRunning) {
            try {
                Socket socket = serverSocket.accept();
                new Client(socket, activeClients).start();
            } catch (IOException e) {
                System.out.println(new ErrorMessage(null, ServerTime.getTime(),
                        "An I/O error occurred during connection establish-
ment."));
            }
        }
    } catch (IOException e) {
```

```
        System.out.println(new ErrorMessage(null, ServerTime.getTime(),
                "An I/O error occurred while opening a socket: " + e.getMes-
sage()));
        System.exit(1);
    }
}
```

…

Класс “Client” принимает сообщение от клиента и рассылает его на сокеты остальных клиентов:

```
public class Client extends Thread implements User {
    private String name;
    private final Socket socket;
    private final List<Client> activeClients;
```

…

```
@Override
public void run() {
    activeClients.add(this);

    try (BufferedReader in = new BufferedReader(
            new InputStreamReader(socket.getInputStream()))) {
        name = in.readLine();
        Message connectionMessage = new InfoMessage(null, ServerTime.getTime(),
                "User \"" + name + "\" from " + socket.getRemoteSocketAddress()
+ " connected.");

        System.out.println(connectionMessage);
        sendMessage(connectionMessage);

        try {
            String line;

            while ((line = in.readLine()) != null) {
                Message message = new UserMessage(this, ServerTime.getTime(),
line);

                System.out.println(message);
                sendMessage(message);
            }
        } catch (SocketException e) {
            Message disconnectionMessage = new InfoMessage(null, ServerTime.get-
Time(),
                    "User \"" + name + "\" from " + socket.getRemoteSocket-
Address() + " disconnected.");

            System.out.println(disconnectionMessage);
            sendMessage(disconnectionMessage);
        } finally {
            socket.close();
            activeClients.remove(this);
        }
    } catch (IOException e) {
        System.out.println(new WarningMessage(null, ServerTime.getTime(),
                "An error occurred while creating an I/O stream: " + e.getMes-
sage()));
    }
}
```

```
@Override
public void sendMessage(Message message) {
    synchronized (activeClients) {
        for (Client client : activeClients) {
            try {
                PrintWriter out = new PrintWriter(cli-
ent.getSocket().getOutputStream(), true);
                out.println(message);
            } catch (IOException e) {
                System.out.println(new WarningMessage(null, ServerTime.get-
Time(),
                        "An error occurred while creating an I/O stream: " +
e.getMessage()));
            }
        }
    }
}

…
```

Основная логика реализации клиентской части:

Model:

Класс “ChatClient” отдельной нитью создает подключение к серверу и оставляет открытыми потоки ввода и вывода на время работы клиентской части программы; Имеет метод получения сообщения с сервера и отправки сообщения на сервер:

```
public class ChatClient implements Model {
    private String username;
    private InetAddress serverAddress;
    private int port;

    private boolean isRunning;
    private boolean isConfigured;

    private BufferedReader in;
    private PrintWriter out;
```

…

```
@Override
public void configure(InetAddress serverAddress, int port, String name) {
    if (isRunning) {
        throw new RuntimeException("Error: it is forbidden to reconfigure a run-
ning model");
    }

    setServerAddress(serverAddress);
    setPort(port);
    setUsername(name);
    isConfigured = true;
}

@Override
public void start() {
    if (isRunning) {
        throw new RuntimeException("Error: re-running the model is prohibited");
    }
```

```
    if (!isConfigured) {
        throw new RuntimeException("Error: the model is not configured");
    }

    try (Socket socket = new Socket(serverAddress, port)) {
        try (BufferedReader in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));
             PrintWriter out = new PrintWriter(
                     socket.getOutputStream(), true)) {
            this.in = in;
            this.out = out;
            out.println(username);

            isRunning = true;

            while (isRunning) {
                try {
                    Thread.sleep(10000);
                } catch (InterruptedException e) {
                    System.out.println(e.getMessage());
                    System.exit(7);
                }
            }
        } catch (IOException e) {
            System.out.println("An error occurred while creating an I/O stream:
"
                    + e.getMessage());
            System.exit(4);
        }
    } catch (IOException e) {
        System.out.println("An I/O error occurred while creating the socket: "
                + e.getMessage());
        System.exit(3);
    }
}

@Override
public String receiveMessage() throws IOException {
    validateForIllegalCall();

    try {
        return in.readLine();
    } catch (IOException e) {
        System.out.println("An error occurred while reading from the socket. "
                + e.getMessage());
        throw e;
    }
}

@Override
public void sendMessage(String message) {
    validateForIllegalCall();
    out.println(message);
}
```

…

Controller:

Контроллер позволяет настроить и запустить модель, а также отдельным потоком запускает цикл получения новых сообщений:

```
public class Controller {
    private final Model model;
    private final View view;

    private boolean isModelRunning;
…
public void startModel(InetAddress serverAddress, int port, String username) {
    if (isModelRunning) {
        throw new RuntimeException("Error: re-running the model is prohibited");
// create exception
    }

    model.configure(serverAddress, port, username);
    new Thread(model::start).start();

    do {
        isModelRunning = model.isRunning();

        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {
            System.out.println(e.getMessage());
            System.exit(7);
        }
    } while (!isModelRunning);

    enableAutoUpdate();
}
…

private void enableAutoUpdate() {
    new Thread(() -> {
        while (isModelRunning) {
            receiveMessage();
        }
    }).start();
}


…
```

View:

Класс “ConsoleView” взаимодействует с пользователем, имеет методы отправки сообщения и отображения сообщения на консоль:

```
public class ConsoleView implements View {
    private Controller controller;


…

    @Override
    public void start() {
        if (controller == null) {
            throw new NullPointerException("Controller must be specified");
        }
```

```
        Scanner scanner = new Scanner(System.in);

        InetAddress serverAddress = getServerAddress(scanner);
        int port = getPort(scanner);

        System.out.print("Enter your name: ");
        String username = scanner.nextLine();

        controller.startModel(serverAddress, port, username);

        String message;

        try {
            while ((message = scanner.nextLine()) != null) {
                sendMessage(message);
            }
        } catch (Exception e) {
            System.out.println("You have disconnected from the chat.");
            System.exit(0);
        }
    }
```

…

```
    @Override
    public void showMessage(String message) {
        System.out.println(message);
    }

    @Override
    public void sendMessage(String message) {
        controller.sendMessage(message);
    }
```

4. Вывод

В ходе лабораторной работы были изучены основные принципы разработки многопользовательских приложений, построенных на основе технологии клиент-сервер с использованием стека протоколов TCP/IP.

Приложение

Текст программы:

```
package ru.nstu.networks.chat.server;

public interface Server {
    void start();

    void stop();
}
```

```
package ru.nstu.networks.chat.server;
import ru.nstu.networks.chat.message.ErrorMessage;
import ru.nstu.networks.chat.message.InfoMessage;
import ru.nstu.networks.chat.user.Client;

import java.io.IOException;
import java.net.*;
import java.util.*;

public class ChatServer implements Server {
    private final int port;
    private final InetAddress inetAddress;
    private final List<Client> activeClients;
    private boolean isRunning = true;

    public ChatServer(int port) {
        if (port < 1024 || port > 49151) {
            throw new IllegalArgumentException("Ports available to users from
1024 to 49151."
                    + " Current port: " + port);
        }

        System.out.println(new InfoMessage(null,
                ServerTime.getTime(), "Attempt to create a new server."));

        validatePort(port);

        this.port = port;
        inetAddress = getInetAddress();
        activeClients = new LinkedList<>();

        System.out.println(new InfoMessage(null,
                ServerTime.getTime(), "A new server has been created at " +
inetAddress
                + ", assigned port: " + port));
    }

    @Override
    public void start() {
        try (ServerSocket serverSocket = new ServerSocket(port)) {
            System.out.println(new InfoMessage(null,
                    ServerTime.getTime(), "Server started at " + inetAddress
                    + ", port: " + serverSocket.getLocalPort()));

            while (isRunning) {
                try {
```

```
                    Socket socket = serverSocket.accept();
                    new Client(socket, activeClients).start();
                } catch (IOException e) {
                    System.out.println(new ErrorMessage(null, ServerTime.get-
Time(),
                            "An I/O error occurred during connection establish-
ment."));
                }
            }
        } catch (IOException e) {
            System.out.println(new ErrorMessage(null, ServerTime.getTime(),
                    "An I/O error occurred while opening a socket: " + e.getMes-
sage()));
            System.exit(1);  //TODO: const
        }
    }

    @Override
    public void stop() {
        isRunning = false;
        System.exit(0); //TODO: const
    }

    private static InetAddress getInetAddress() {
        try (DatagramSocket socket = new DatagramSocket()) {
            socket.connect(InetAddress.getByName("8.8.8.8"), 10000);
            return socket.getLocalAddress();
        } catch (Exception e) {
            System.out.println(new ErrorMessage(null,
                    ServerTime.getTime(), "Failed to get server inet-ad-
dress."));
        }

        System.exit(4);
        return null;
    }

    private static void validatePort(int port) {
        if (port < 1024 || port > 49151) {
            throw new IllegalArgumentException(
                    "The port number must be in the range from 1024 to 49151. "
                            + "Current port number: " + port);
        }
    }
}
```

```
package ru.nstu.networks.chat.server;

import java.time.LocalDateTime;
import java.time.ZoneId;
import java.time.format.DateTimeFormatter;

public final class ServerTime {
    public static String getTime() {
        return LocalDateTime.now(ZoneId.of("Europe/Moscow"))
                .format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss:SS"));
    }
}
```

```java
package ru.nstu.networks.chat.message;

import ru.nstu.networks.chat.user.User;

import java.io.Serializable;

public abstract class Message implements Serializable {
    protected final User sender;
    protected final String time;
    protected final String text;

    public Message(User sender, String time, String text) {
        this.sender = sender;
        this.time = time;
        this.text = text;
    }

    public abstract String getType();

    public User getSender() {
        return sender;
    }

    public String getDispatchTime() {
        return time;
    }

    public String getText() {
        return text;
    }

    public String toString() {
        return text;
    }
}
```

```java
package ru.nstu.networks.chat.message;

import ru.nstu.networks.chat.user.User;

public class ErrorMessage extends Message {
    private static final String type = "[ERROR]";

    public ErrorMessage(User sender, String time, String text) {
        super(sender, time, text);
    }

    @Override
    public String getType() {
        return type;
    }

    @Override
    public String toString() {
        return type + " " + time + ": " + text;
    }
}
```

```java
package ru.nstu.networks.chat.message;

import ru.nstu.networks.chat.user.User;
```

```java
public class InfoMessage extends Message {
    private static final String type = "[INFO]";

    public InfoMessage(User sender, String time, String text) {
        super(sender, time, text);
    }

    @Override
    public String getType() {
        return type;
    }

    @Override
    public String toString() {
        return type + " " + time + ": " + text;
    }
}
```

```java
package ru.nstu.networks.chat.message;

import ru.nstu.networks.chat.user.User;

public class UserMessage extends Message {
    private static final String type = "[MESSAGE]";

    public UserMessage(User sender, String time, String text) {
        super(sender, time, text);
    }

    @Override
    public String getType() {
        return type;
    }

    @Override
    public String toString() {
        return type + " " + time + " " + sender + ": " + text;
    }
}
```

```java
package ru.nstu.networks.chat.message;

import ru.nstu.networks.chat.user.User;

public class WarningMessage extends Message {
    private static final String type = "[WARNING]";

    public WarningMessage(User sender, String time, String text) {
        super(sender, time, text);
    }

    @Override
    public String getType() {
        return type;
    }

    @Override
    public String toString() {
        return type + " " + time + ": " + text;
```

```
    }
}
```

```
package ru.nstu.networks.chat.user;

import ru.nstu.networks.chat.message.Message;

import java.net.InetAddress;
import java.net.Socket;

public interface User {
    Socket getSocket();

    InetAddress getAddress();

    void sendMessage(Message message);
}
```

```
package ru.nstu.networks.chat.user;

import ru.nstu.networks.chat.message.InfoMessage;
import ru.nstu.networks.chat.message.Message;
import ru.nstu.networks.chat.message.UserMessage;
import ru.nstu.networks.chat.server.ServerTime;
import ru.nstu.networks.chat.message.WarningMessage;

import java.io.*;
import java.net.InetAddress;
import java.net.Socket;
import java.net.SocketException;
import java.util.List;

public class Client extends Thread implements User {
    private String name;
    private final Socket socket;
    private final List<Client> activeClients;

    public Client(Socket socket, List<Client> activeClients) {
        this.socket = socket;
        this.activeClients = activeClients;
    }

    @Override
    public Socket getSocket() {
        return socket;
    }

    @Override
    public InetAddress getAddress() {
        return socket.getInetAddress();
    }

    @Override
    public void run() {
        activeClients.add(this);

        try (BufferedReader in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()))) {
            name = in.readLine();
            Message connectionMessage = new InfoMessage(null, ServerTime.get-
Time(),
```

```
                    "User \"" + name + "\" from " + socket.getRemoteSocket-
Address() + " connected.");

            System.out.println(connectionMessage);
            sendMessage(connectionMessage);

            try {
                String line;

                while ((line = in.readLine()) != null) {
                    Message message = new UserMessage(this, ServerTime.get-
Time(), line);

                    System.out.println(message);
                    sendMessage(message);
                }
            } catch (SocketException e) {
                Message disconnectionMessage = new InfoMessage(null, Server-
Time.getTime(),
                        "User \"" + name + "\" from " + socket.getRemoteSocket-
Address() + " disconnected.");

                System.out.println(disconnectionMessage);
                sendMessage(disconnectionMessage);
            } finally {
                socket.close();
                activeClients.remove(this);
            }
        } catch (IOException e) {
            System.out.println(new WarningMessage(null, ServerTime.getTime(),
                    "An error occurred while creating an I/O stream: " + e.get-
Message()));
        }
    }

    @Override
    public void sendMessage(Message message) {
        synchronized (activeClients) {
            for (Client client : activeClients) {
                try {
                    PrintWriter out = new PrintWriter(cli-
ent.getSocket().getOutputStream(), true);
                    out.println(message);
                } catch (IOException e) {
                    System.out.println(new WarningMessage(null, ServerTime.get-
Time(),
                            "An error occurred while creating an I/O stream: " +
e.getMessage()));
                }
            }
        }
    }

    @Override
    public String toString() {
        return name + " from " + getAddress();
    }
}
```

```
package ru.nstu.networks.chat;

import ru.nstu.networks.chat.server.ChatServer;
```

```
import ru.nstu.networks.chat.server.Server;

public class Main {
    public static void main(String[] args) {
        Server server = new ChatServer(2009);
        server.start();
    }
}
```

```
package ru.nstu.networks.client.model;

import java.io.IOException;
import java.net.InetAddress;

public interface Model {
    void configure(InetAddress serverAddress, int port, String name);

    void start();

    boolean isRunning();

    String receiveMessage() throws IOException;

    void sendMessage(String message);

    void stop();
}
```

```
package ru.nstu.networks.client.model;

import java.io.*;
import java.net.InetAddress;
import java.net.Socket;

public class ChatClient implements Model {
    private String username;
    private InetAddress serverAddress;
    private int port;

    private boolean isRunning;
    private boolean isConfigured;

    private BufferedReader in;
    private PrintWriter out;

    public ChatClient() {
    }

    public void setUsername(String username) {
        if (username == null) {
            throw new NullPointerException("Error: client username must be spec-
ified");
        }

        this.username = username;
    }

    private void setServerAddress(InetAddress serverAddress) {
        if (serverAddress == null) {
            throw new NullPointerException("Error: server address cannot refer
to null");
```

```
        }

        this.serverAddress = serverAddress;
    }

    private void setPort(int port) {
        if (port < 1024 || port > 49151) {
            throw new IllegalArgumentException("Ports available to users from
1024 to 49151."
                    + " Current port: " + port);
        }

        this.port = port;
    }

    @Override
    public void configure(InetAddress serverAddress, int port, String name) {
        if (isRunning) {
            throw new RuntimeException("Error: it is forbidden to reconfigure a
running model");
        }

        setServerAddress(serverAddress);
        setPort(port);
        setUsername(name);
        isConfigured = true;
    }

    @Override
    public void start() {
        if (isRunning) {
            throw new RuntimeException("Error: re-running the model is prohib-
ited");
        }

        if (!isConfigured) {
            throw new RuntimeException("Error: the model is not configured");
        }

        try (Socket socket = new Socket(serverAddress, port)) {
            try (BufferedReader in = new BufferedReader(
                    new InputStreamReader(socket.getInputStream()));
                 PrintWriter out = new PrintWriter(
                         socket.getOutputStream(), true)) {
                this.in = in;
                this.out = out;
                out.println(username);

                isRunning = true;

                while (isRunning) {
                    try {
                        Thread.sleep(10000);
                    } catch (InterruptedException e) {
                        System.out.println(e.getMessage());
                        System.exit(7);
                    }
                }
            } catch (IOException e) {
                System.out.println("An error occurred while creating an I/O
stream: "
                        + e.getMessage());
                System.exit(4); //TODO: create a constant
            }
```

```
        } catch (IOException e) {
            System.out.println("An I/O error occurred while creating the socket:
"
                    + e.getMessage());
            System.exit(3); //TODO: create a constant
        }
    }

    @Override
    public boolean isRunning() {
        return isRunning;
    }

    @Override
    public String receiveMessage() throws IOException {
        validateForIllegalCall();

        try {
            return in.readLine();
        } catch (IOException e) {
            System.out.println("An error occurred while reading from the socket.
"
                    + e.getMessage());
            throw e;
        }
    }

    @Override
    public void sendMessage(String message) {
        validateForIllegalCall();
        out.println(message);
    }

    @Override
    public void stop() {
        validateForIllegalCall();
        isRunning = false;
    }

    private void validateForIllegalCall() {
        if (!isRunning) {
            throw new RuntimeException("Calling the method is prohibited: the
model is not running");
        }
    }
}
```

```
package ru.nstu.networks.client.controller;

import ru.nstu.networks.client.model.Model;
import ru.nstu.networks.client.view.View;

import java.io.IOException;
import java.net.InetAddress;

public class Controller {
    private final Model model;
    private final View view;

    private boolean isModelRunning;

    public Controller(Model model, View view) {
```

```
        this.model = model;
        this.view = view;
    }

    public void startModel(InetAddress serverAddress, int port, String username)
{
        if (isModelRunning) {
            throw new RuntimeException("Error: re-running the model is prohib-
ited"); // create exception
        }

        model.configure(serverAddress, port, username);
        new Thread(model::start).start();

        do {
            isModelRunning = model.isRunning();

            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                System.out.println(e.getMessage());
                System.exit(7);
            }
        } while (!isModelRunning);

        enableAutoUpdate();
    }

    public void stopModel() {
        validateForIllegalCall();
        model.stop();
    }

    public void sendMessage(String message) {
        validateForIllegalCall();
        model.sendMessage(message);
    }

    private void receiveMessage() {
        try {
            view.showMessage(model.receiveMessage());
        } catch (IOException e) {
            System.out.println("An error occurred: " + e.getMessage());
        }
    }

    private void enableAutoUpdate() {
        new Thread(() -> {
            while (isModelRunning) {
                receiveMessage();
            }
        }).start();
    }

    private void validateForIllegalCall() {
        if (!isModelRunning) {
            throw new RuntimeException("Calling the method is prohibited: the
model is not running");
        }
    }
}
```

```java
package ru.nstu.networks.client.view;

import ru.nstu.networks.client.controller.Controller;

public interface View {
    void start();

    void setController(Controller controller);

    void showMessage(String message);

    void sendMessage(String message);
}
```

```java
package ru.nstu.networks.client.view;

import ru.nstu.networks.client.controller.Controller;

import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.Scanner;

public class ConsoleView implements View {
    private Controller controller;

    public ConsoleView() {
    }

    @Override
    public void setController(Controller controller) {
        this.controller = controller;
    }

    @Override
    public void start() {
        if (controller == null) {
            throw new NullPointerException("Controller must be specified");
        }

        Scanner scanner = new Scanner(System.in);

        InetAddress serverAddress = getServerAddress(scanner);
        int port = getPort(scanner);

        System.out.print("Enter your name: ");
        String username = scanner.nextLine();

        controller.startModel(serverAddress, port, username);

        String message;

        try {
            while ((message = scanner.nextLine()) != null) {
                sendMessage(message);
            }
        } catch (Exception e) {
            System.out.println("You have disconnected from the chat.");
            System.exit(0);
        }
    }

    @Override
    public void showMessage(String message) {
        System.out.println(message);
```

```
    }

    @Override
    public void sendMessage(String message) {
        controller.sendMessage(message);
    }

    private InetAddress getServerAddress(Scanner scanner) {
        while (true) {
            try {
                System.out.print("Enter the server address: ");
                InetAddress serverAddress = InetAddress.getByName(scanner.next-
Line());

                if (serverAddress == null || serverAddress.getHostAddress().is-
Empty()) {
                    throw new UnknownHostException();
                }

                return serverAddress;
            } catch (UnknownHostException e) {
                System.out.println("Error: Unknown server address. Try again.");
            }
        }
    }

    private int getPort(Scanner scanner) {
        while (true) {
            try {
                System.out.print("Enter the port number: ");
                int port = scanner.nextInt();

                if (port < 1024 || port > 49151) {
                    throw new IllegalArgumentException("Ports available to users
from 1024 to 49151."
                            + " Current port: " + port);
                }

                return port;
            } catch (IllegalArgumentException e) {
                System.out.println("Error: " + e.getMessage() + ". Try again.");
            } catch (Exception e) {
                System.out.println("An error occurred while reading int number
from console");
            } finally {
                scanner.nextLine();
            }
        }
    }
}
```

```
package ru.nstu.networks.client;

import ru.nstu.networks.client.controller.Controller;
import ru.nstu.networks.client.model.ChatClient;
import ru.nstu.networks.client.model.Model;
import ru.nstu.networks.client.view.ConsoleView;
import ru.nstu.networks.client.view.View;

public class Main {
    public static void main(String[] args) {
        Model model = new ChatClient();
        View view = new ConsoleView();
```

```
        Controller controller = new Controller(model, view);
        view.setController(controller);

        view.start();
    }
}
```