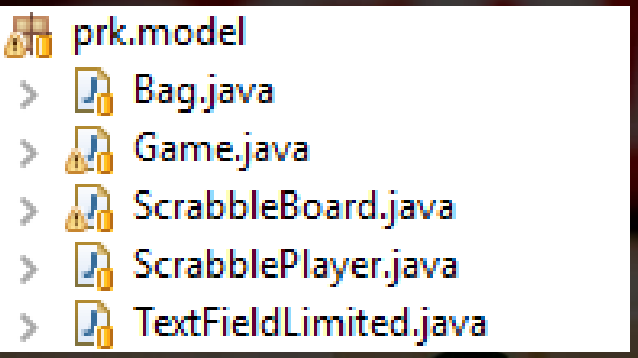
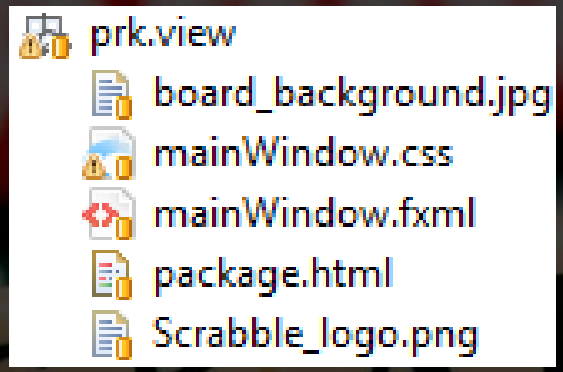
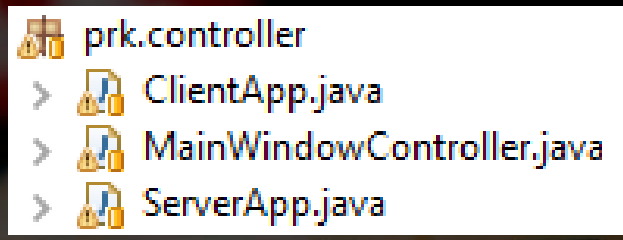
# Projekt końcowy

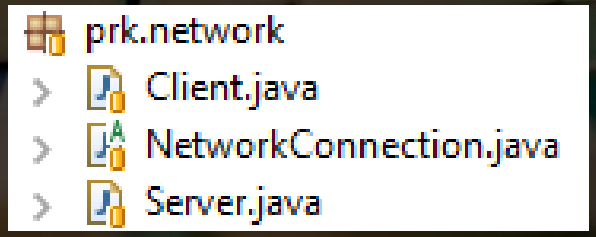

Wojciech Krzywiec
Maciej Gawłowski

# Realizacja
# Model – View - Controller



# Połączenie sieciowe
# Klasa abstrakcyjna NetworkConnection

```
public abstract class NetworkConnection {

    private ConnectionThread connThread = new ConnectionThread();
    private Consumer<Serializable> onReceiveCallback;

    public NetworkConnection(Consumer<Serializable> onReceiveCallBack) {
        this.onReceiveCallback = onReceiveCallBack;
        connThread.setDaemon(true);
    }

    public void startConnection() throws Exception {
        connThread.start();
    }

    public void send(Serializable data) throws Exception {
        connThread.out.writeObject(data);
    }

    public void closeConnection() throws Exception {
        connThread.socket.close();
    }

    protected abstract boolean isServer();

    protected abstract String getIP();

    protected abstract int getPort();

    private class ConnectionThread extends Thread {..
}
```

Instancja klasy prywatnej

Umożliwia odbiór wiadomości

Kiedy JVM się zatrzymuje, wątek jest natychmiast zatrzymany

Klasa prywatna

# Połączenie sieciowe
# Klasa prywatna ConnectionThread

```
    private class ConnectionThread extends Thread {
        private Socket socket;
        private ObjectOutputStream out;

        @Override
        public void run() {
            try (ServerSocket server = isServer() ? new ServerSocket(getPort()) : null;
                    Socket socket = isServer() ? server.accept() : new Socket(getIP(), getPort());
                    ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());
                    ObjectInputStream in = new ObjectInputStream(socket.getInputStream())) {
                this.socket = socket;
                this.out = out;
                socket.setTcpNoDelay(true);

                while (true) {
                    Serializable data = (Serializable) in.readObject();
                    onReceiveCallback.accept(data);
                }

            } catch (Exception e) {
                onReceiveCallback.accept("Connection closed");
            }
        }
    }
}
```

# Połączenie sieciowe
# Klasy dziedziczące po NetworkConnection

```
public class Client extends NetworkConnection {
    private String ip;
    private int port;

    public Client(String ip, int port, Consumer<Serializable> onReceiveCallBack) {
        super(onReceiveCallBack);
        this.ip = ip;
        this.port = port;
    }

    @Override
    protected boolean isServer() {
        return false;
    }

    @Override
    protected String getIP() {
        return ip;
    }

    @Override
    protected int getPort() {
        return port;
    }

}
```

# Połączenie sieciowe
# Klasy dziedziczące po NetworkConnection

```
public class Server extends NetworkConnection {
    private int port;

    public Server(int port, Consumer<Serializable> onReceiveCallBack) {
        super(onReceiveCallBack);
        this.port = port;
    }

    @Override
    protected boolean isServer() {
        return true;
    }

    @Override
    protected String getIP() {
        return null;
    }

    @Override
    protected int getPort() {
        return port;
    }

}
```

# Połączenie sieciowe
# Klasa uruchomieniowa ServerApp

```
public class ServerApp extends Application {

    private MainWindowController mainWindowController;

    private Stage primaryStage;
    private NetworkConnection connection = createServer();

    public void mainWindow() {

    @Override
    public void init() throws Exception {
        connection.startConnection();
    }

    public void start(Stage primaryStage) throws Exception {

    @Override
    public void stop() throws Exception {
        connection.closeConnection();
    }

    public static void main(String[] args) {

    private Server createServer() {
        return new Server(55555, data -> {
            mainWindowController.getMessage(data.toString());
        });
    }
```

Metoda wywołana natychmiast po załadowaniu klasy i wywołaniu konstruktora

Metoda wywołana kiedy aplikacja powinna się zatrzymać, zakończenie połączenia

# Połączenie sieciowe
# Klasa uruchomieniowa ClientApp

```
public class ClientApp extends Application {

    private MainWindowController mainWindowController;

    private Stage primaryStage;
    private NetworkConnection connection = createClient();

    public void mainWindow() throws Exception {

    @Override
    public void init() throws Exception {
        connection.startConnection();
    }

    public void start(Stage primaryStage) throws Exception {

    @Override
    public void stop() throws Exception {
        connection.closeConnection();
    }

    public static void main(String[] args) {

    private Client createClient() {
        return new Client("localhost", 55555, data -> {
            mainWindowController.getMessage(data.toString());
        });
    }
```