СПбНИУ ИТМО
Кафедра ВТ

Лабораторная работа №5
по дисциплине
«Программирование интернет-приложений»
Вариант 2048

Выполнил
Широков О.И
гр.2120

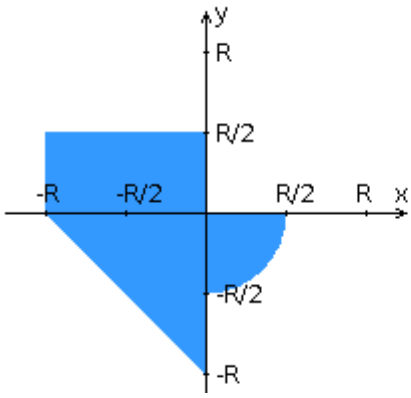
Санкт-Петербург
г.2013

1. Разделить приложение из лабораторной работы №4 на две составляющие - клиентскую и серверную, обменивающиеся сообщениями по заданному протоколу.
На стороне клиента осуществляются ввод и передача данных серверу, прием и отображение ответов от сервера и отрисовка области. В сообщении клиента должна содержаться вся необходимая информация для определения факта попадания/непопадания точки в область.

Сервер должен принимать сообщения клиента, обрабатывать их в соответствии с заданной областью и отправлять клиенту ответное сообщение, содержащее сведения о попадании/непопадании точки в область.

Приложение должно удовлетворять следующим требованиям:

- Для передачи сообщений необходимо использовать протокол TCP.
- Для данных в сообщении клиента должен использоваться тип double.
- Для данных в ответном сообщении сервера должен использоваться тип Boolean.
- Каждое сообщение на сервере должно обрабатываться в отдельном потоке. Класс потока должен быть унаследован от класса Thread.
- Приложение должно быть локализовано на 2 языка - английский и шведский.
- Строки локализации должны храниться в отдельном классе.
- Приложение должно корректно реагировать на "потерю" и "восстановление" связи между клиентом и сервером; в случае недоступности сервера клиент должен показывать введённые пользователем точки серым цветом.



2. Исходный код

```
package RegLocation.Net;


import java.io.IOException;
import java.net.ServerSocket;
import java.net.SocketAddress;

public class SilhouetteServer
{
    public SilhouetteServer(SocketAddress addr)
    {
        try
        {
            ServerSocket mySocket = new ServerSocket();
            mySocket.bind(addr);
            while (true)
            {
                ClientSession session = new ClientSession(mySocket.accept());
                System.out.println("Session started!");
                session.start();
```

```
            }
        }
        catch (IOException ex)
        {
            ex.printStackTrace();
        }
    }
}
package RegLocation.Net;


import java.net.Socket;
import java.util.HashMap;

public class SilhouetteProcThread extends  Thread
{
    Socket _sck;
    Request request;
    public SilhouetteProcThread(Socket cli_socket, Request request1)
    {
        this._sck = cli_socket;
        this.request = request1;
    }

    @Override
    public void run()
    {
        double MarkX = request.MarkX;
        double MarkY = request.MarkY;
        double R     = request.R;
        int MHC = request.MarkHashCode;
        boolean IRR = Silhouette.InRegion((float)MarkX,(float)MarkY,(float)R);

        Response response = new Response(MHC, IRR, R);

        SilhouetteChannel.sendResponse(response,_sck);
    }
}


package RegLocation.Net;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;
import java.net.SocketException;


public class SilhouetteChannel
{
    public static void sendRequest(Request request,Socket _sck)  throws IOException
    {
            OutputStream sckStream = _sck.getOutputStream();
            request.getBytesStream(sckStream);

    }

    public static void sendResponse(Response response,Socket _sck)// throws SocketException
    {
        try
        {
            OutputStream sckStream = _sck.getOutputStream();
            response.getBytesStream(sckStream);
        }
   /*     catch (SocketException ex)
        {
            throw ex;
        }
        */
        catch (IOException ex)
        {
            ex.printStackTrace();
        }

    }


    public static Response receiveResponse(Socket _sck)
    {
```

```
        Response ret = null;
        try
        {
            InputStream sckStream = _sck.getInputStream();
            ret = Response.toResponse(sckStream);
        }
        catch (SocketException ex)
        {
            return null;
        }
        catch (IOException ex)
        {
            ex.printStackTrace();
        }
        return ret;
    }

    public  static  Request receiveRequest(Socket _sck)
    {
        Request ret = null;
        try
        {
            InputStream sckStream = _sck.getInputStream();
            ret = Request.toRequest(sckStream);
        }
        catch (SocketException ex)
        {
            return null;
        }
        catch (IOException ex)
        {
            ex.printStackTrace();
        }
        return ret;
    }

}
package RegLocation.Net;

import java.io.*;

public class Response implements Serializable
{
    public int MarkHashCode;
    public boolean InRegionResult;
    public double R;

    /**
     *
     * @param MHC Mark HashCode
     * @param IRR Result of execution InRegion
     * @param R   Current R value
     */
    public Response(int MHC, boolean IRR,double R)
    {
        this.MarkHashCode   = MHC;
        this.InRegionResult = IRR;
        this.R              = R;
    }

    public static Response toResponse(InputStream BytesStream)
    {
        Response ret = null;
        try
        {
            ObjectInputStream deserialize = new ObjectInputStream(BytesStream);
            ret = (Response)deserialize.readObject();

        }
        catch (IOException ex)
        {
            return null;
        }
        catch (ClassNotFoundException ex)
        {
            ex.printStackTrace();
        }

        return ret;
    }
```

```
    public void getBytesStream(OutputStream outStream)
    {
        try
        {
            ObjectOutputStream serializer = new ObjectOutputStream(outStream);
            serializer.writeObject(this);
        }
        catch (IOException ex)
        {
            ex.printStackTrace();
        }

        return;
    }
}
```

```
package RegLocation.Net;

import java.io.IOException;
import java.net.Socket;


public class ClientSession extends  Thread
{
    Socket socket;
    public ClientSession(Socket socketr)
    {
        this.socket = socketr;
    }

    @Override
    public void run()
    {
        while(true)
        {
            try
            {
                Request request;

                request = Request.toRequest(socket.getInputStream());
                if(request == null)
                    continue;
                new SilhouetteProcThread(socket,request).start();
            }
            catch (IOException ex)
            {
                ex.printStackTrace();
            }
        }
    }
}
```

```
package RegLocation;

import java.util.ListResourceBundle;


public class MyBundle_en_US extends ListResourceBundle
{
    @Override
    protected Object[][] getContents()
    {
        return content;
    }
    static final Object[][] content =
            {
                    {"AddButton"  , "Press me, please"},
                    {"DefaultSTR" , "Nothing pressed"},
                    {"JBText", new String[]{ "0", "0.25", "0.5", "0.75", "1"} }
            };
}
```

```
package RegLocation;

import RegLocation.Net.Mark;
import RegLocation.Net.Request;
import RegLocation.Net.Response;
import RegLocation.Net.SilhouetteChannel;
import org.omg.PortableServer.THREAD_POLICY_ID;

import java.io.IOException;
import java.lang.reflect.Array;
import java.net.ConnectException;
import java.net.Socket;
import java.net.SocketAddress;
import java.net.SocketException;
import java.util.HashMap;
import java.util.Vector;

public class MarkProcessingPool
{
    Vector<Integer> pool;
    Vector<Mark> MAL;
    HashMap<Integer, MarkStates> colors;
    SocketAddress sckAddr;
    Socket        sck;

    public MarkProcessingPool(HashMap<Integer, MarkStates> mcolors, Vector<Mark> mal,SocketAddress
lsckAddr)
    {
        this.colors  = mcolors;
        this.MAL     = mal;
        this.pool    = new Vector<Integer>();
        this.sckAddr = lsckAddr;
        sck = new Socket();

        Thread PoolProcessing = new Thread(new Runnable()
        {
            @Override
            public void run()
            {
                while(true)
                {
                    Mark[] MAL_Array = MAL.toArray(new Mark[MAL.size()]);
                    Integer[] pool_array = pool.toArray(new Integer[pool.size()]);
                    for(Integer hash : pool_array)
                    {
                        Mark found = null;
                        for(Mark i : MAL)
                            if(i.hashCode() == hash)
                            {
                                found = i;
                                break;
                            }

                        Request myRequest = new Request(found,SilhouetteComponent.R);
                        HashMap<Integer, MarkStates> myHashMap = colors;

                        boolean flag = false;
                        do
                        {
                            try
                            {
                                SilhouetteChannel.sendRequest(myRequest, sck);
                                flag = false;
                            }
                            catch (SocketException ex)
                            {
                                tryConnect();
                                flag = true;
                            }
                            catch (IOException ex)
                            {
                                ex.printStackTrace();
                            }
                        } while (flag);
                        Response response = SilhouetteChannel.receiveResponse(sck);

                        if(response != null)
                        {
                            if((float)response.R != SilhouetteComponent.R)
```

```
                            continue;
                        MarkStates state;
                        if(response.InRegionResult)
                            state = MarkStates.InRegion;
                        else
                            state = MarkStates.OutOfRegion;

                        myHashMap.put(response.MarkHashCode, state);
                        pool.remove(hash);
                    }
                }
            }
        }
    });

    PoolProcessing.start();
}

void tryConnect()
{
    boolean flag = false;
    boolean itFirst = true;
    do
    {
        if(!itFirst)
        try
        {
            Thread.sleep(1000);
        }
        catch (InterruptedException ex)
        {
            Thread.currentThread().interrupt();
        }
        itFirst = false;
        flag    = false;
        try
        {

            sck = new Socket();
            sck.connect(sckAddr,1000);
        }
        catch (SocketException ex)
        {
            flag = true;
        }
        catch (IOException ex)
        {
            ex.printStackTrace();
        }
    } while (flag);
    return;
}

public void addMarkInPool(int MarkHashCode)
{
    pool.add(MarkHashCode);
}

}
```

3. Вывод

В процессе выполнения лабораторной работы была изучена модель сетевого взаимодействия в Java, понятие сериализации и локализации.