



LẬP TRÌNH JAVA 2

BÀI 7: NETWORKING, SOCKET, SOCKET TCP, SOCKET UDP

PHẦN 1

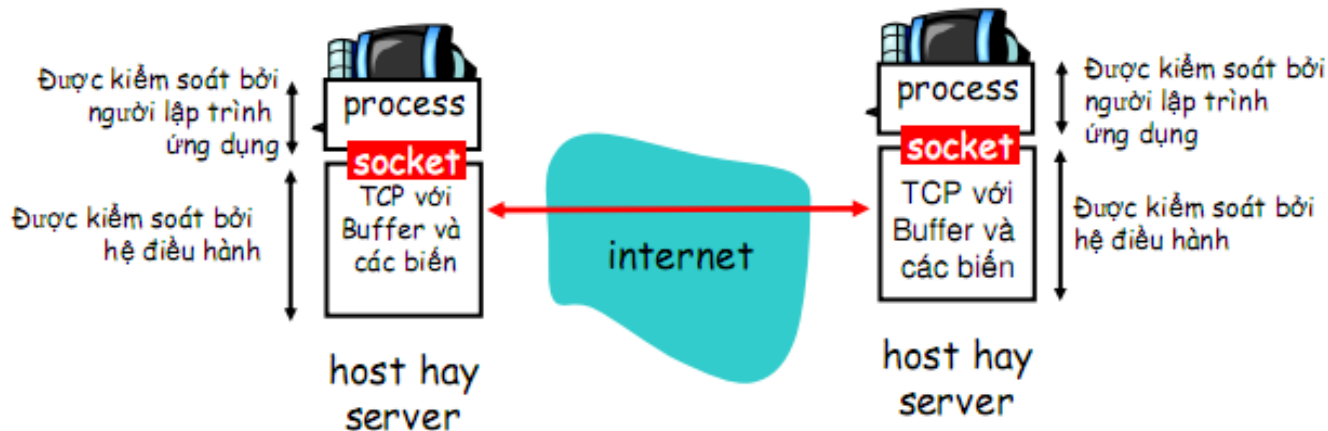
- ◎ Kết thúc bài học này bạn có khả năng
 - ❖ Giới thiệu Socket
 - ❖ Các lớp cần thiết của gói thư viện java.net
 - ❖ Lập trình Socket TCP
 - ❖ Lập trình Socket UDP
 - ❖ Bài tập



- ❑ Socket là giao diện lập trình ứng dụng (API) hay bộ thư viện hàm hỗ trợ, dùng để nối kết chương trình ứng dụng với lớp mạng trong hệ thống mạng TCP/IP.
- ❑ Được giới thiệu trong BSD4.1 UNIX, 1981
- ❑ Được khởi tạo, sử dụng và hủy một cách tường minh bởi ứng dụng
- ❑ Mô hình client/Server

❑ Cơ chế giao tiếp

- ❖ Một trong hai quá trình phải công bố số hiệu cổng của socket mà mình sử dụng để nhận và gửi dữ liệu.
- ❖ Các quá trình khác có thể giao tiếp với quá trình đã công bố cổng cũng bằng cách tạo ra một socket.



- ❑ Socket: Như là cửa thông giữa các quá trình ứng dụng và giao thức truyền tải end-to-end(UDP hay TCP)
- ❑ TCP: là dịch vụ truyền tin cậy theo **bytes** từ tiến trình này đến tiến trình khác.

- ❑ Hai loại dịch vụ truyền tải qua socket API:
 - ❖ Datagram không bảo đảm (UDP)
 - ❖ Connection-oriented bảo đảm (TCP)
- ❑ So sánh giữa TCP và UDP:

Có nối kết (TCP)	Không nối kết (UDP)
Tồn tại kênh giao tiếp ảo giữa 2 quá trình	Không tồn tại kênh giao tiếp ảo giữa 2 quá trình
Dữ liệu được gửi đi theo chế độ bảo đảm : có kiểm tra lỗi, truyền lại gói tin lỗi hay mất, bảo đảm thứ tự đến của các gói tin ...	Dữ liệu được gửi đi theo chế độ không bảo đảm : Không kiểm tra lỗi, không phát hiện và không truyền lại gói tin bị lỗi hay bị mất, không bảo đảm thứ tự đến của các gói tin ...
Dữ liệu chính xác Tốc độ truyền chậm	Dữ liệu không chính xác Tốc độ truyền nhanh
Thích hợp cho các ứng dụng cần độ chính xác cao: truyền file, thông tin điều khiển ...	Thích hợp cho các ứng dụng cần tốc độ, không cần chính xác cao: truyền âm thanh, hình ảnh ...

- ❑ Cổng (port): là 1 số 16 bit
 - ❖ Từ 0 – 1023: cổng hệ thống
 - ❖ Từ 1024 – 49151: cổng đã đăng ký (registered port)
 - ❖ Từ 49152 – 65535: cổng dùng riêng (private port).
- ❑ Một số cổng thông dụng
 - ❖ Echo: cổng 7 (TCP, UDP)
 - ❖ Web: cổng 80 (TCP)
 - ❖ FTP: cổng 21 cho nối kết và 20 cho dữ liệu (TCP)
 - ❖ SMTP: cổng 25 (TCP)
 - ❖ POP: cổng 110 (TCP)
 - ❖ Telnet: cổng 23 (TCP)
 - ❖ DNS: cổng 53 (TCP và UDP)
 - ❖ SNMP: cổng 161 (UDP)
 - ❖ RIP: cổng 520 (UDP)





- ☐ InetAddress
- ☐ ServerSocket
- ☐ Socket

❑ Các phương lớp của lớp InetAddress:

- ❖ `static InetAddress getLocalHost():` trả về địa chỉ máy cục bộ
- ❖ `static InetAddress getByName(String host):` nhận địa chỉ máy kiểu chuỗi, trả về đối tượng `InetAddress` thay mặt cho địa chỉ máy này.
- ❖ `public String getHostName():` trả về tên của đối tượng `InetAddress` theo dạng `String`.
- ❖ `public byte[] getAddress():` trả về địa chỉ IP của đối tượng `InetAddress` theo dạng mảng các byte.
- ❖ `public String.getHostAddress():` trả về địa chỉ IP của đối tượng `InetAddress` theo dạng `String`.

❑ Ví dụ: Tìm địa chỉ IP của localhost

```
public class Demo1 {  
    public static void main(String[] args) {  
        try{  
            InetAddress myHost = InetAddress.getLocalHost();  
            System.out.println("Host address: "+myHost.getHostAddress());  
            System.out.println("Host name: "+myHost.getHostName());  
        } catch (UnknownHostException ex) {  
            ex.printStackTrace();  
        }  
    }  
}
```

Output - DemoSOF203 (run) ☒	
	run:
	Host address: 192.168.72.1
	Host name: SCD050718
	BUILD SUCCESSFUL (total time: 16 seconds)

❑ Ví dụ: Tìm các địa chỉ IP của "dantri.com.vn"

```
public class Demo2 {  
    public static void main(String[] args) {  
        try{  
            InetAddress []address = InetAddress.getAllByName("dantri.com.vn");  
            for(int i=0;i<address.length;i++){  
                System.out.println("Address "+(i+1)+" : "+address[i]);  
            }  
        }catch(UnknownHostException ex){  
            ex.printStackTrace();  
        }  
    }  
}
```

Output - DemoSOF203 (run) ☒



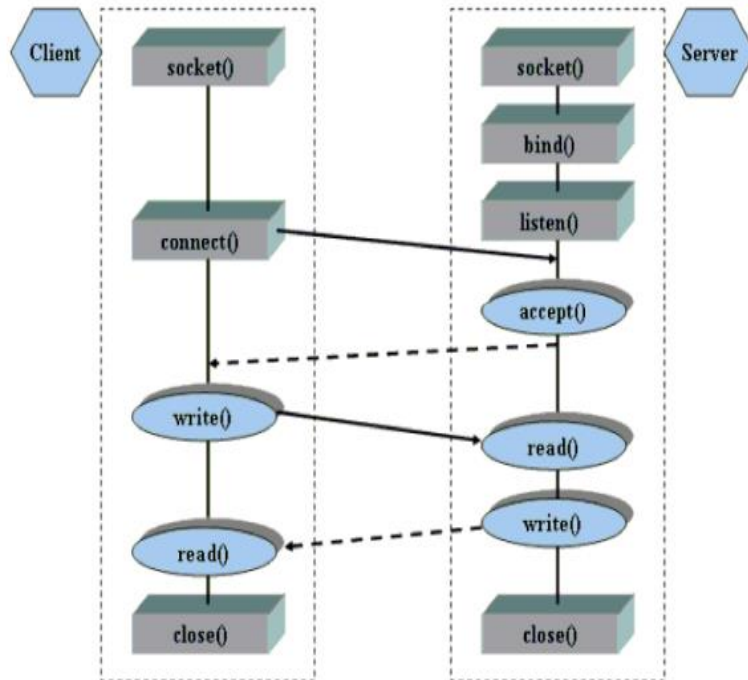
```
run:  
Address 1: dantri.com.vn/123.30.215.27  
Address 2: dantri.com.vn/123.30.215.63  
BUILD SUCCESSFUL (total time: 0 seconds)
```

❑ Các phương lớp của lớp Socket:

- ❖ `public Socket(String host, int port)`: tạo một kết nối theo địa chỉ host và số cổng port.
- ❖ `public Socket(InetAddress address, int port)`: tạo một kết nối theo địa chỉ là đối tượng `InetAddress` và số cổng port.
- ❖ `public Socket(String host, int port, boolean stream)`: tạo một kết nối theo địa chỉ host và số cổng port, `stream = true` để quy định kết nối theo TCP, ngược lại, kết nối theo UDP (User Datagram Protocol).
- ❖ `InputStream getInputStream()`: Lấy về luồng nhập để nhận dữ liệu.
- ❖ `OutputStream getOutputStream()`: Lấy về luồng xuất để gửi dữ liệu.
- ❖ `int getPort()`: Lấy về số hiệu cổng kết nối của máy chủ.
- ❖ `synchronized void close()`: Cắt đứt kết nối với máy

❑ Các phương lớp của lớp ServerSocket:

- ❖ `public ServerSocket(int port)`: tạo một đối tượng lắng nghe những kết nối từ máy khách theo số cổng port.
- ❖ `Socket accept()`: dừng lại chờ cho đến khi nhận được kết nối và trả về đối tượng Socket của máy khách.
- ❖ `synchronized void close()`: Cắt đứt kết nối với máy khách.

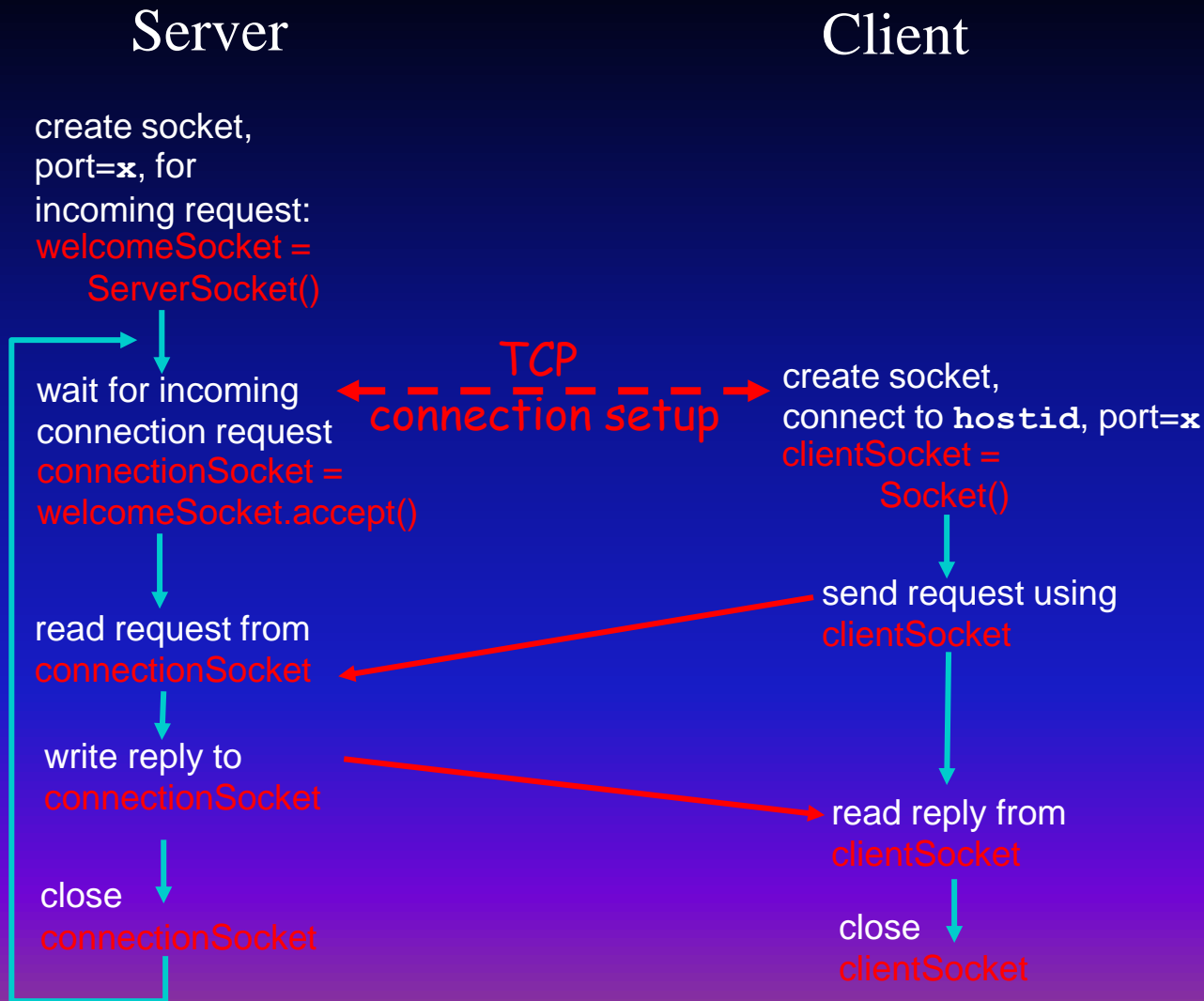


❑ Server

- ❖ Server process phải chạy trước
- ❖ Server phải tạo một socket để lắng nghe và chấp nhận các kết nối từ client

❑ Client

- ❖ Khởi tạo TCP socket
 - ❖ Xác định IP address, port của server
 - ❖ Thiết lập kết nối đến server.
- ❑ Khi server nhận yêu cầu kết nối, nó sẽ chấp nhận yêu cầu và khởi tạo socket mới để giao tiếp với client
- ❖ Server có thể chấp nhận nhiều yêu cầu client tại một thời điểm



Ví dụ: tạo ứng dụng client-server như sau

- ❑ Client đọc một dòng kí tự từ input chuẩn (**inFromUser** stream) , gửi tới server qua socket (**outToServer** stream)
- ❑ server đọc dòng kí tự trong sockets
- ❑ server biến đổi dòng kí tự đó thành dòng kí tự chỉ gồm các chữ hoa và gửi trả về cho client.
- ❑ client đọc, in ra dòng kí tự đã biến đổi từ socket (**inFromServer** stream)

❑ Client

```
import java.io.*;  
import java.net.*;  
class TCPClient {
```

```
    public static void main(String argv[]) throws Exception  
    {
```

```
        String sentence;  
        String modifiedSentence;
```

Tạo input stream



```
        BufferedReader inFromUser =  
            new BufferedReader(new InputStreamReader(System.in));
```

Tạo client socket,
kết nối tới server



```
        Socket clientSocket = new Socket("hostname", 6789);
```

Tạo output stream,
đính kèm vào socket



```
        DataOutputStream outToServer =  
            new DataOutputStream(clientSocket.getOutputStream());
```


❑ Client

Tạo input stream,
đính kèm vào trong
socket

```
BufferedReader inFromServer =  
    new BufferedReader(new  
        InputStreamReader(clientSocket.getInputStream()));  
  
sentence = inFromUser.readLine();  
  
Gửi dòng kí tự  
đến server
```

```
outToServer.writeBytes(sentence + '\n');
```

Đọc dòng kí tự
(đã biến đổi) gửi
về từ server

```
modifiedSentence = inFromServer.readLine();  
System.out.println("FROM SERVER: " + modifiedSentence);  
  
clientSocket.close();  
  
}  
}
```

❑ Server

```
import java.io.*;  
import java.net.*;
```

```
class TCPServer {
```

```
    public static void main(String argv[]) throws Exception  
    {
```

```
        String clientSentence;  
        String capitalizedSentence;
```

Tạo sẵn Socket
ở cổng 6789

```
        ServerSocket welcomeSocket = new ServerSocket(6789);
```

```
        while(true) {
```

Đợi đến khi có socket
từ client gửi đến

```
            Socket connectionSocket = welcomeSocket.accept();
```

Tạo input stream,
đính kèm vào socket

```
            BufferedReader inFromClient =  
                new BufferedReader(new  
                    InputStreamReader(connectionSocket.getInputStream()));
```

❑ Server

Tạo output stream,
đính kèm vào
socket

```
DataOutputStream outToClient =  
    new DataOutputStream(connectionSocket.getOutputStream());
```

Đọc dòng kí tự
trong socket

```
clientSentence = inFromClient.readLine();
```

```
capitalizedSentence = clientSentence.toUpperCase() + '\n';
```

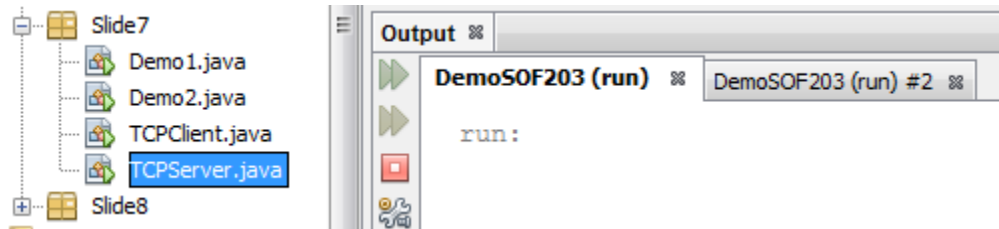
Ghi dòng kí tự đã
biến đổi vào socket

```
outToClient.writeBytes(capitalizedSentence);
```

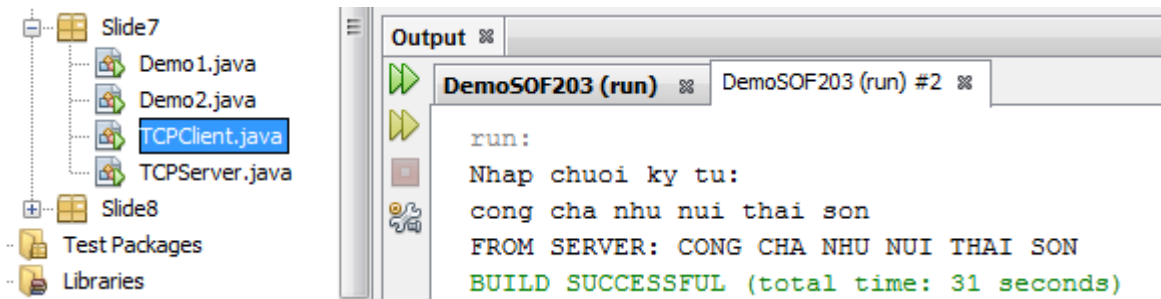
```
}  
}  
}
```

Kết thúc vòng lặp while,
quay trở về vòng lặp cha,
đợi kết nối khác

❑ Chạy bên server trước (TCPServer)



❑ Chạy bên client sau (TCPClient)





DEMO

Chạy và giải thích





LẬP TRÌNH JAVA 2

BÀI 7: NETWORKING, SOCKET, SOCKET TCP, SOCKET UDP

PHẦN 2

- ❑ Gửi email là hoạt động thường xuyên xảy ra của 1 ứng dụng. Có thể người dùng gửi cho 1 ai đó, hoặc có thể là thông báo của hệ thống
- ❑ Để gửi email trong Java bạn cần 2 thư viện
 - ❖ Mail.jar
 - ❖ Activation.jar



❑ Các phương thức được sử dụng của MimeMessage trong việc gửi email.

Phương thức	Mô tả
setFrom(InternetAddress)	Cung cấp địa chỉ email người gửi
setReplyTo(InternetAddress[])	Cung cấp địa chỉ email người nhận phản hồi
addRecipients(RecipientType, Address[])	Cung cấp danh sách địa chỉ email người cùng nhận
setSubject(String)	Cung cấp tiêu đề mail
setContent(String, String)	Cung cấp nội dung mail
setSentDate(Date)	Cung cấp ngày gửi mail

Properties

Chuẩn bị thông số cấu hình



Authenticator

Cung cấp tài khoản kết nối mail server



Session

Mở một session dựa vào cấu hình và authenticator ở trên



MimeMessage

Xây dựng mail: người nhận, người gửi, tiêu đề, nội dung...



Transport

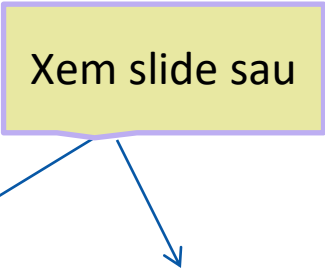
Gửi mail

```
try{
    String fromName = "Nguyễn Văn Sender";
    String fromEmail = "sender@gmail.com";
    String toEmails = "a@gmail.com,b@fpt.edu.vn";
    String subject = "Tiêu đề mail";
    String body = "Nội dung mail";

    Session session = Session.getInstance(config, authenticator);
    MimeMessage mail = new MimeMessage(session);

    InternetAddress sender = new InternetAddress(fromEmail, fromName, "utf-8");
    mail.setFrom(sender);
    mail.setReplyTo(new InternetAddress[]{sender});
    mail.addRecipients(Message.RecipientType.TO, toEmails);
    mail.setSubject(subject, "utf-8");
    mail.setContent(body, "text/html; charset=utf-8");
    mail.setSentDate(new Date());

    Transport.send(mail);
}
catch (Exception e) {
    throw new RuntimeException(e);
}
```



CODE GỬI EMAIL QUA GMAIL


```
//các thông số gmail
Properties prop = new Properties();
prop.put(key:"mail.smtp.auth", value: "true");
prop.put(key:"mail.smtp.starttls.enable", value: "true"); //TLS
prop.put(key:"mail.smtp.host", value: "smtp.gmail.com");
prop.put(key:"mail.smtp.port", value: "587");
prop.put(key:"mail.smtp.socketFactory.port", value: "587");
prop.put(key:"mail.smtp.socketFactory.class", value: "javax.net.ssl.SSLSocketFactory");
prop.put(key:"mail.smtp.ssl.protocols", value: "TLSv1.2");
//xác thực qua gmail
String senderEmail = txtSenderEmail.getText();
String password = new String(value: txtPass.getPassword());
Session session = Session.getInstance(props: prop,
    new javax.mail.Authenticator() {
        protected PasswordAuthentication getPasswordAuthentication() {
            return new PasswordAuthentication(userName: senderEmail, password);
        }
    });
```

❑ Viết code cho button “Send”

The screenshot displays an IDE interface for a Java project named **SendMail.java**. The **Projects** pane on the left shows the project structure, including **Test Packages**, **Libraries** (with **sqljdbc4.jar** and **javax.mail.jar**), and **Other Components - Navigator**. The **Other Components - Navigator** pane is highlighted with a red box, showing a tree view of the UI components: **JFrame** contains **jPanel1 [JPanel]** (with **jLabel1 [JLabel]**, **txtUser [JTextField]**, **jLabel2 [JLabel]**, and **txtPass [JPasswordField]**) and **jPanel2 [JPanel]** (with **jLabel3 [JLabel]**, **jLabel4 [JLabel]**, **jLabel5 [JLabel]**, **txtTo [JTextField]**, **txtSubject [JTextField]**, **jScrollPane1 [JScrollPane]** containing **txtMessage [JTextArea]**, **btnSend [JButton]**, and **jLabel6 [JLabel]**). The **Design** view on the right shows the visual layout of the application. It features a title bar with **Start Page** and **SendMail.java**. The main content area is titled **Send Mail** and contains two sections: **Send** and **Receive**. The **Send** section has **Username:** and **Password:** labels followed by text input fields. The **Receive** section has **To:** and **Subject:** labels followed by text input fields, and a **Message:** label followed by a text area. A **Send** button is located at the bottom of the **Receive** section.

❑ Thiết kế form như hình

```
public void sendEmail(){
    //các thông số gmail
    Properties prop = new Properties();
    prop.put(key:"mail.smtp.auth", value: "true");
    prop.put(key:"mail.smtp.starttls.enable", value: "true"); //TLS
    prop.put(key:"mail.smtp.host", value: "smtp.gmail.com");
    prop.put(key:"mail.smtp.port", value: "587");
    prop.put(key:"mail.smtp.socketFactory.port", value: "587");
    prop.put(key:"mail.smtp.socketFactory.class", value: "javax.net.ssl.SSLSocketFactory");
    prop.put(key:"mail.smtp.ssl.protocols", value: "TLSv1.2");
    //xác thực qua gmail
    String senderEmail = txtSenderEmail.getText();
    String password = new String(value: txtPass.getPassword());
    Session session = Session.getInstance(props: prop,
        new javax.mail.Authenticator() {
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(userName: senderEmail, password);
            }
        });
    try {
        Message message = new MimeMessage(session);
        message.setFrom(new InternetAddress(address:senderEmail));
        message.setRecipients(
            rt: Message.RecipientType.TO,
            adrss: InternetAddress.parse(addresslist:txtReceiveEmail.getText())
        );
        message.setSubject(string: txtSubject.getText());
        message.setContent(o: txtMessage.getText(), string: "text/html; charset=utf-8");
        Transport.send(msg:message);
        System.out.println(x: "Gửi thành công");
    } catch (MessagingException e) {
        e.printStackTrace();
    }
}
```



Send Mail

Send

Username:

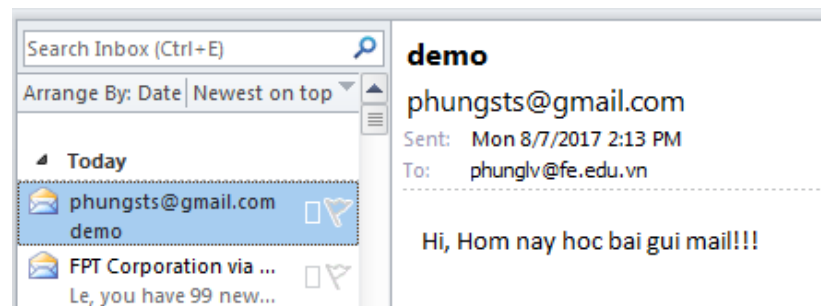
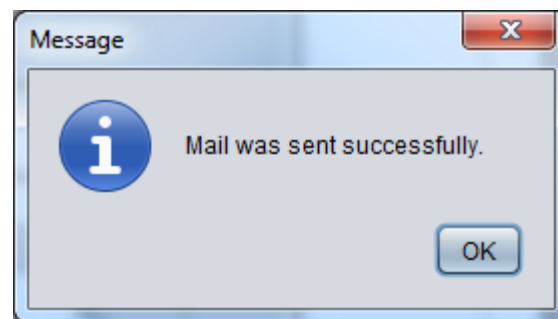
Password:

Receive

To:

Subject:

Message:



- ❑ Để gửi mail cho nhiều người cùng nhận bạn chỉ cần bổ sung các email ấy vào CC hoặc BCC.
- ❑ Sự khác biệt giữa CC và BCC là CC là những email sẽ được người nhận nhìn thấy trên mail còn BCC thì không.
- ❑ Mã bổ sung email cùng nhận:

```
String ccEmails = "cc1@gmail.com,cc2@yahoo.com,cc3@hotmail.com";  
mail.addRecipients(Message.RecipientType.CC, ccEmails);
```

```
String bccEmails = "bcc1@gmail.com,bcc2@yahoo.com";  
mail.addRecipients(Message.RecipientType.BCC, bccEmails);
```

```
Transport.send(mail);
```

- ☐ Bài tập: Nâng cấp bài Send Email trên có thêm chức năng CC hoặc BCC


```
// Phần nội dung mail chính
MimeBodyPart contentPart = new MimeBodyPart();
contentPart.setContent(body, "text/html; charset=utf-8");

// Phần file đính kèm
MimeBodyPart filePart = new MimeBodyPart();
File file = new File("c:/temp/a.gif");
FileDataSource fds = new FileDataSource(file);
filePart.setDataHandler(new DataHandler(fds));
filePart.setFileName(file.getName());

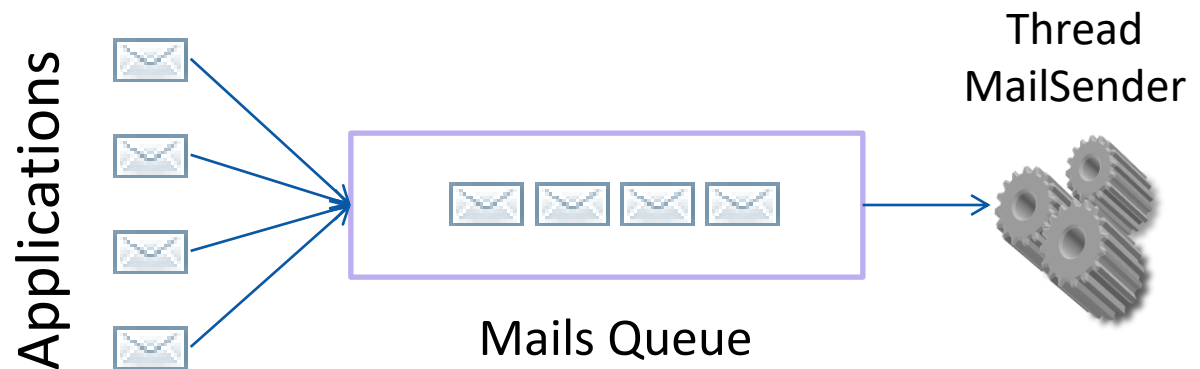
// Bổ sung các phần vào multi-part
MimeMultipart multiPart = new MimeMultipart();
multiPart.addBodyPart(contentPart);
multiPart.addBodyPart(filePart);

// Thiết lập nội dung mail là multi-part
mail.setContent(multiPart);

Transport.send(mail);
```

- ☐ Bài tập: Nâng cấp bài Send Email trên có thêm chức năng đính kèm theo file

- ❑ Chi phí (CPU và bộ nhớ) dành cho gửi mail là rất cao. Đôi khi hệ thống phải thực hiện gửi nhiều email một lúc dẫn đến tài nguyên cạn kiệt dễ gây tê liệt hệ thống.
- ❑ Giải pháp khắc phục là tại queue nắm giữ các mail và có 1 thread chuyển trách gửi tuần tự từng email.



```
public class MailSender extends Thread{
    static {
        MailSender sender = new MailSender();
        sender.start();
    }
    static final List<MimeMessage> queue = new ArrayList<>();

    public static void queue(MimeMessage mail) {
        synchronized(queue) {
            queue.add(mail);
            queue.notify();
        }
    }

    @Override
    public void run() { ...24 lines }
}
```

Xem slide sau

```
while (true) {
    try {
        synchronized (queue) {
            if (queue.size() > 0) {
                try {
                    MimeMessage mail = queue.remove(0);
                    Transport.send(mail);
                    System.out.println("The mail was sent.");
                }
                catch (MessagingException e) {
                    System.out.println("Unable to send mail.");
                }
            }
            else {
                queue.wait();
            }
        }
    }
    catch (InterruptedException e) {
        break;
    }
}
```

```
Session session = Session.getInstance(config, authenticator);
MimeMessage mail = new MimeMessage(session);

InternetAddress sender = new InternetAddress(fromEmail, fromName, "utf-8");
mail.setFrom(sender);
mail.setReplyTo(new InternetAddress[]{sender});
mail.addRecipients(Message.RecipientType.TO, toEmails);
mail.setSubject(subject, "utf-8");
mail.setContent(body, "text/html; charset=utf-8");
mail.setSentDate(new Date());

//Transport.send(mail);
MailSender.queue(mail);
```

Thay vì gọi ~~Transport.send(mail)~~ để gửi email thì sử dụng **MailSender.queue(mail)** để bổ sung email vào queue và sẽ được gửi bởi 1 Thread khác

- ❖ Giới thiệu Socket
- ❖ Các lớp cần thiết của gói thư viện java.net
- ❖ Lập trình Socket TCP
- ❖ Lập trình Socket UDP
- ❖ Bài tập





Cảm ơn