کارگاه برنامه‌نویسی پیشرفته

دستور کار شماره ده

اهداف

آشنایی با مفاهیم مقدماتی شبکه

آشنایی با سوکت[1] در جاوا

[1] Socket

فهرست مطالب

مقدماتی بر شبکه در جاوا

تعریف سوکت و پورت[1]

به هر یک از نقاط انتهایی[2] در اتصال بین دو برنامه یا دو وسیله، یک سوکت گفته می‌شود. هر کدام از این نقاط انتهایی، خود شامل یک جفت آی‌پی[3] و پورت (یک عدد صحیح ۱۶ بیتی) منحصر به فرد است. از آنجایی که یک کلاینت[4] یا سرور، خود می‌توانند به صورت همزمان با چندین برنامه و دستگاه دیگر متصل باشند، برای برقراری ارتباط با هر یک، نیاز به پورت‌های منحصر به فرد داریم.

مراحل ساخت کانکشن[5]

برای ساخت یک کانکشن، برای سرور یک ServerSocket با پورت دلخواه می‌سازیم (پورت نباید کمتر از ۱۰۲۴ باشد چرا که این پورت‌ها رزرو شده‌اند و به دسترسی ادمین نیاز دارند). سرور بر روی این پورت به درخواست‌های اتصال کلاینت گوش می‌کند:

```java
public class Server {
   public static void main(String[] args) {
       try {
           ServerSocket serverSocket = new ServerSocket(6000);
           System.out.println("Server is listening...");
           Socket client = serverSocket.accept();
           System.out.println("connected to a client");
           System.out.println(client.getPort());
       } catch (IOException exception) {
           exception.printStackTrace();
       }
   }
}
```

(ساخت سوکت در سمت سرور)

```
server is listening...
```

(خروجی تکه کد بالا)

[1] Port
[2] Endpoint
[3] IP Address
[4] Client
[5] Connection

متد accept، یک متد blocking است و تا زمانی که کلاینت به سرور متصل نشود، خط‌های بعدی اجرا نمی‌شوند. کلاینت برای وصل شدن به سرور باید آدرس آی‌پی و پورت سوکتی که سرور در حال گوش دادن به آن است را بداند و با ساختن سوکتی، به سرور وصل شود:

```java
public class Client {
    public static void main(String[] args) {
        try {
            Socket socket = new Socket("127.0.0.1", 6000);
            System.out.println(socket.getInetAddress()); // 127.0.0.1
        } catch (IOException exception) {
            exception.printStackTrace();
        }
    }
}
```

(ساخت سوکت در سمت کلاینت و وصل شدن به سرور)

آی‌پی «127.0.0.1» مربوط به خود کامپیوتری است که کد روی آن اجرا شده و به آن «localhost» هم گفته می‌شود.

```
server is listening...
connected to a client
11658
```

(خروجی کد در سمت سرور)

همان‌طور که مشاهده کردید، متد accept در سرور، یک سوکت جدید با پورت جدید ساخت تا بین سرور و کلاینت اتصال برقرار شود.

نحوه‌ی تعامل در سوکت

همانند فایل‌ها برای تبادل اطلاعات بین سوکت‌ها می‌توانیم از کلاس‌هایی مانند ObjectInputStream، DataOutputStream، DataInputStream و غیره استفاده کنیم. در ادامه به بررسی نحوه‌ی تبادل اطلاعات با استفاده از ObjectStreamها[1] می‌پردازیم.

کلاس Message به صورت شئ بین سرور و کلاینت مبادله می‌شود، به همین دلیل باید اینترفیس Serializable را پیاده‌سازی کند:

```java
public class Message implements Serializable {
    private String content;
    private Date date;

    public Message(String content, Date date) {
        this.content = content;
        this.date = date;
    }

    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    @Override
    public String toString() {
        return "Message: " + this.content + " (Date: " + this.date.toString() + ")";
    }
}
```

(پیاده‌سازی کلاس Message)

[1] ObjectInputStream & ObjectOutputStream

```java
public class Client {
   public static void main(String[] args) {
       System.out.println("\t".repeat(7) + "{CLIENT}\n");
       try {
           // Send a message to the server
           Socket socket = new Socket("127.0.0.1", 6000);
           Message message = new Message("Hey, how are doing? :D", new Date());
           ObjectOutputStream outputStream =
                   new ObjectOutputStream(socket.getOutputStream());
           outputStream.writeObject(message);

           // Receive a message from the server
           ObjectInputStream inputStream =
                   new ObjectInputStream(socket.getInputStream());
           Message serverMessage = (Message) inputStream.readObject();
           System.out.println(serverMessage.toString());
       } catch (IOException | ClassNotFoundException exception) {
           exception.printStackTrace();
       }
   }
}
```

(پیاده‌سازی کلاس Client)

```java
public class Server {
   public static void main(String[] args) {
       System.out.println("\t".repeat(7) + "{SERVER}\n");
       try {
           // Receive a message from the client
           ServerSocket serverSocket = new ServerSocket(6000);
           Socket client = serverSocket.accept();
           ObjectInputStream inputStream =
                   new ObjectInputStream(client.getInputStream());
           Message clientMessage = (Message) inputStream.readObject();
           System.out.println(clientMessage.toString());

           // Send a message to the client
           ObjectOutputStream outputStream =
                   new ObjectOutputStream(client.getOutputStream());
           Message message = new Message("Good, how about you?", new Date());
           outputStream.writeObject(message);

       } catch (IOException | ClassNotFoundException exception) {
           exception.printStackTrace();
       }
   }
}
```

(پیاده‌سازی کلاس Server)

```
                                {SERVER}

Message: Hey, how are doing? :D (Date: Thu Mar 03 16:46:56 IRST 2022)
```

(خروجی سرور)

```
                                {CLIENT}

Message: Good, how about you? (Date: Thu Mar 03 16:46:56 IRST 2022)
```

(خروجی کلاینت)

بدون استفاده از ObjectStreamها هم می‌توان اطلاعات را فرستاد. برای مثال، مانند کار با فایل‌ها، می‌توانستیم با استفاده از InputStream یا OutputStream پیام‌ها را به صورت باینری مبادله کنیم.

انجام دهید: چت‌روم

پیاده‌سازی چت‌روم

در این جلسه قصد داریم با استفاده از مفاهیم شبکه، یک چت‌روم در کنسول پیاده‌سازی کنیم. توضیح برنامه به صورت زیر است:

۱. در ابتدای اجرای برنامه، هر کاربر می‌تواند یک نام کاربری برای خود انتخاب کند.
۲. اگر کاربری به چت‌روم اضافه شد، به تمام کاربران اطلاع داده می‌شود.
۳. پیامی که هر کاربر می‌نویسد، برای تمامی کاربران (به جز خودش) ارسال می‌شود.
۴. پیام هر کاربر را می‌توان از کاربران دیگر، تمییز داد (برای مثال نام هر کاربر به همراه پیامی که فرستاده است، نمایش داده می‌شود).
۵. هر کاربر، در هر زمانی که خواست، با نوشتن عبارت «#exit»، می‌تواند از چت‌روم خارج شود.
۶. اگر کاربری از چت‌روم خارج شد (چه با استفاده از دستور exit و چه با بستن برنامه)، به تمام کاربران دیگر اطلاع داده می‌شود.

به مثال زیر توجه کنید:

```
********** WELCOME TO MY CHATROOM! **********

enter your username: Farid
Farid joined Chatroom!

Hey! How're you doing?
Arian: I'm pretty good!
Hamed: I have to go guys. see you later!
Hamed left the chatroom!
```

(مثالی از یک چت‌روم پیاده‌سازی شده)

نکات کلی:

- مثال بالا صرفاً برای فهم بهتر سؤال زده شده و شما می‌توانید هرگونه پیاده‌سازی‌ای را که شروط گفته شده را رعایت می‌کند، ارائه دهید.
- در مثال بالا از چاپ نوشته‌های رنگی در کنسول استفاده شده است که انجام این کار اختیاری می‌باشد و الزامی به پیاده‌سازی آن نیست. می‌توانید از این لینک کمک بگیرید.

نحوه‌ی تحویل

قبل از پیاده‌سازی این تمرین، لازم است که مخزنی جدید در گیت‌هاب با نام AP-Workshop10 برای خودتان بسازید. دقت کنید که مخزنی که می‌سازید، حتماً از نوع private باشد که باقی افراد به آن دسترسی نداشته باشند.

برای انجام این تمرین، می‌بایست طبق مراحل زیر عمل کنید:

۱. ابتدا برنچی با نام Server درست کنید و کلاس‌های مربوط به ساخت سرور را در آن توسعه دهید.

۲. برنچ کامل‌شده‌ی Server را با برنچ اصلی (Master) مرج کنید.

۳. پس از آن برنچی از برنچ اصلی با نام Client ساخته و کلاس‌های مربوط به ساخت کلاینت را در آن توسعه دهید.

۴. پس از کامل شدن برنچ Client آن را با برنچ اصلی مرج کرده و کار را به اتمام برسانید.