به نام خدا

دستور کار کارگاه برنامه‌نویسی پیشرفته

جلسه دهم

آشنایی با ریسه‌ها و کار با شبکه در جاوا

مقدمه

در این جلسه قرار است با ریسه‌ها و برنامه‌نویسی شبکه در جاوا آشنایی پیدا کنیم. از شبکه برای برقراری ارتباط میان چند کامپیوتر استفاده می‌شود. می‌توان برنامه‌هایی نوشت که روی چند سیستم اجرا می‌شوند و با استفاده از شبکه به صورت از راه دور به یکدیگر سرویس می‌دهند. ریسه‌ها نیز امکان اجرای همروند چند کار را در یک برنامه می‌دهند.

برنامه‌نویسی تحت شبکه

برای برنامه‌نویسی تحت شبکه روش‌های مختلفی وجود دارد. یکی از این روش‌ها بر مبنای معماری کلاینت-سرور[1] است. نمونه کلاینت و سرور را هر روز وقتی در حال استفاده از اینترنت هستید تجربه می‌کنید. مثلا وقتی به سایت گوگل وصل می‌شوید، یک کامپیوتر در نقش سرور (سرویس‌دهنده) وب‌سایت گوگل در یک نقطه از دنیا قرار دارد و کامپیوتر شما در نقش کلاینت (سرویس‌گیرنده) با آن ارتباط برقرار می‌کند. در روش برنامه‌نویسی سوکت[2] در معماری کلاینت-سرور دو برنامه وجود دارد؛ یکی از این برنامه‌ها در نقش سرور اجرا می‌شود که در آن یک ServerSocket تعریف می‌گردد، و دیگری در نقش کلاینت اجرا می‌شود که یک Socket در آن تعریف می‌شود. پس یک سوکت بر روی سرور گوگل وجود دارد و یک سوکت بر روی مرورگر کامپیوتر شما. در ادامه توضیح می‌دهیم که چگونه می‌توان یک سرور به زبان جاوا نوشت و در آن با برنامه نویسی سوکت سمت سرور آشنا می‌شویم.

---

[1] Client-Server

[2] Socket Programming

برای ایجاد یک سوکت سرور ابتدا باید یک نمونه از نوع ServerSocket بسازید. کد زیر نشان‌دهنده کد ساخت سوکت سرور است:

```
ServerSocket server = new ServerSocket(5000);
```

هنگام ساخت سوکت باید یک شماره پورت[3] نیز به آن اختصاص دهید. شماره پورت یک عدد یکتا بر روی کامپیوتر اجراکننده برنامه سرور است که نشان می‌دهد داده‌های ارسال‌شده بر روی شبکه مربوط به کدام برنامه است. برای مثال، فرض کنید نرم‌افزار پیام‌رسان تلگرام (یا سروش) و اینستاگرام همزمان بر روی موبایل شما در حال اجرا باشد. داده‌های دریافت‌شده توسط موبایل شما باید تفکیک شود تا مشخص شود که به کدام یک از این برنامه‌ها مربوط است. شماره پورت برای همین تفکیک داده‌ها تعیین می‌شود و هر برنامه با یک شماره پورت اختصاصی کار می‌کند. البته در درس شبکه‌های کامپیوتری بیشتر با این مفاهیم آشنا خواهید شد.

حالا که سوکت سرور را ساخته‌اید، باید برنامه سرور در حال انتظار باشد تا زمانی که درخواست برقراری ارتباطی بر روی آن سوکت داده شد، بتوان به طور مناسب نسبت به پذیرش یا عدم پذیرش ارتباط تصمیم‌گیری کرد. در کد بالا، برنامه در حال گوش‌دادن به پورت شماره ۵۰۰۰ است و منتظر است تا داده‌ای دریافت نماید. هنگامی که کلاینت درخواست برقراری ارتباط را می‌دهد، در صورت پذیرش ارتباط از طرف سرور، یک خط ارتباطی میان کلاینت و سرور برقرار می‌شود که این خط ارتباطی در نمونه‌ای از نوع Socket ذخیره می‌گردد.

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

حال که ارتباط برقرار شده است، می‌توان داده‌ها را از روی سوکت ایجادشده خوانده و داده‌هایی را به کلاینت ارسال کرد. داده‌های دریافتی یا ارسالی می‌تواند به صورت رشته‌ای از کاراکترها و یا یک شی باشد (مانند انواع خواندن از و نوشتن در فایل که در جلسه قبل دیدیم). پس برای دریافت اطلاعات باید یک جریان داده ساخته و به سوکت متصل کنیم. در این مثال می‌خواهیم داده به صورت رشته‌ای از کاراکترها را بخوانیم.

```
inFromClient = new DataInputStream(
                    new BufferedInputStream(socket.getInputStream()));
String line = inFromClient.readLine();
```

[3] Port Number

در قطعه کد بالا تنها یک خط از داده ارسال‌شده را از روی سوکت خوانده‌ایم. می دانیم که اگر قرار باشد تعداد خط بیشتری بخوانیم، لازم است یک حلقه ساخته و عملیات خواندن را درون آن انجام دهیم و این کار را تا زمانی که داده کلاینت تمام نشده باشد، باید ادامه دهیم.

حالا اگر نیاز باشد که سرور نیز در جواب کلاینت پیامی ارسال کند، از یک جریان داده خروجی باید استفاده کرد.

```
DataOutputStream outToClient =
            new DataOutputStream(connectionSocket.getOutputStream());
outToClient.writeBytes("Got it!");
```

و در پایان باید سوکت ایجادشده را ببندیم.

```
socket.close();
```

توجه! در نوشتن کدهای بالا باید exceptionهای مربوط به هر یک از خطوط را به درستی مدیریت کرد.

انجام دهید

سرور ذخیره متن

حالا که ساخت یک سرور را یاد گرفتیم، می خواهیم یک سرور ذخیره متن بنویسیم. در این سرور یک سوکت ایجاد شده و از کلاینت داده می‌گیرد. دریافت داده را تا جایی ادامه می‌دهد که سرور کلمه *over* را برای آن ارسال نماید. سرور متن‌های دریافتی در هر بار را در یک شی از نوع String ذخیره می‌کند. سپس تمام داده‌های دریافتی را برای کلاینت می‌فرستد. هر بار که کلاینت داده جدیدی می‌فرستد، به آن String اضافه شده و دوباره به کلاینت باز فرستاده می‌شود.

برنامه‌نویسی با ریسه‌ها

در قسمت قبل آموختیم که چگونه می‌توانیم یک سرور ایجاد کرده و تقاضاهایی که به آن می‌رسد را پردازش نماییم، همانطور که می‌توانید در کد قسمت قبل خود ببینید، این پردازش به صورت ترتیبی است؛ یعنی تا زمانی که پردازش یک تقاضا در یک ارتباط به پایان نرسیده است، نمی‌توان تقاضای بعدی را پردازش نمود. این محدودیت باعث می‌شود که سرور شما در هر لحظه فقط به یک کلاینت سرویس دهد. همانطور که می‌دانید ریسه‌ها امکان اجرای همروند دستورات را در یک برنامه

می‌دهند. پس یک راه حل برای محدودیت بیان‌شده پیاده‌سازی پردازش تقاضاهای کاربر در ریسه‌های جداگانه است.

یکی از روش‌های ساخت یک ریسه این است که اینترفیس Runnable را پیاده‌سازی کرده و یک نمونه از این کلاس را به سازنده‌ی Thread بدهیم.

```
Thread t = new Thread(new RunnableDemo(), “my thread name”);
```

اینترفیس Runnable دارای یک متد به نام run است که در یک ریسه به طور جدا اجرا می‌گردد. زمانی که یک ریسه ساخته می‌شود آغاز به کار نمی‌کند، بلکه می‌بایست به صورت مشخص با فراخوانی متد ()start آن را آغاز و به حالت running تغییر وضعیت داد.

انجام دهید

سرور ذخیره متن با سرویس‌دهی همزمان

در این قسمت قصد داریم که سروری که در قسمت قبل پیاده‌سازی کردیم را به گونه‌ای گسترش دهیم که بتواند از چند کلاینت به صورت همزمان پشتیبانی نماید. برای این کار نیاز است که از ریسه‌ها برای هر ارتباط استفاده کنیم. در قسمتی از کد قبل که Socket را از ServerSocket دریافت می‌کردیم و آن را برای سرویس‌دهی به یک تابع دیگر تحویل می‌دادیم، چنین می‌نویسیم:

```
Thread t = new Thread(new Handler(socket), “Handler thread”);
t.Start();
```

که در آن Handler به صورت زیر پیاده‌سازی شده است:

```
class Handler implements Runnable {

     Handler() {
                …
     }
     public void run() {
                …
     }
}
```