پروژه‌ی شماره ۲

تراکنش‌های بانکی

۱. مقدمه

یک ساختار ساده از شبکه‌ی بانکی را در نظر بگیرید. در این ساختار تعدادی پایانه (Terminal) بانکی با اتصال به سرور مرکزی درخواست‌های واریز و یا برداشت را برای سرور ارسال می‌کنند. این درخواست‌ها پس از اعتبارسنجی توسط سرور اجرا شده و نتیجه‌ی انجام آنها برای پایانه‌ی مورد نظر ارسال می‌گردد.

گستردگی پایانه‌ها و همچنین حجم بالای تراکنش‌ها باعث افزایش احتمال ارسال همزمان چند درخواست برای سرور می‌شود که باید به نحو صحیحی مدیریت شود. به این صورت که از بروز خطا در اثر اجرای همزمان چند درخواست بر روی یک سپرده جلوگیری شود.

همچنین جهت پیگیری خطا و رفع مغایرت‌های احتمالی، لازم است که تمام وقایعی که در سرور و پایانه‌ها اتفاق می‌افتد در فایل‌های ثبت وقایع (Log File) نوشته‌شود.

۲. تعریف پروژه:

نحوه‌ی اجرای برنامه به این شکل است که ابتدا شماره پورت سرور در فایل تنظیمات آن به صورت دستی مشخص می‌شود. سپس سرور اجرا شده و تنظیمات و اطلاعات سپرده‌ها را از فایل خوانده و منتظر می‌ماند تا پایانه‌ها از طریق شبکه به آن متصل شوند.

در سمت پایانه، ابتدا آدرس و شماره پورت سرور در فایل تنظیمات به صورت دستی مشخص شده و سپس ترمینال اجرا می‌شود. ترمینال بر اساس تنظیمات به سرور متصل شده و درخواست‌ها را ارسال می‌کند.

سرور با دریافت هر درخواست، پس از اعتبارسنجی (صحت مقادیر داده‌شده، کفایت مقدار موجودی، عدم تخطی از سقف موجودی و ...)، آن را اجرا کرده و نتیجه را برای ترمینال ارسال می‌کند. نیازی به ثبت اطلاعات سپرده‌ها در پایگاه داده نیست و کافی است در حافظه نگهداری شوند. انتخاب و طراحی ساختار داده‌ی مورد استفاده برای نگهداری و تبادل اطلاعات به‌عهده‌ی برنامه‌نویس است.

پایانه با دریافت پاسخ درخواست خود، نتیجه را در فایلی با عنوان response.xml (با فرمت مناسب) ثبت می‌کند.

لازم است سرور و همه‌ی پایانه‌ها تمام وقایع مربوط به دریافت و ارسال پیام و عملیات داخلی خود را به میزان لازم و کافی در فایل‌های وقایع‌نگاری ثبت کنند.

در سمت سرور در هر زمان که از طریق console، پیغام "sync" به سرور داده شد، کلیه اطلاعات سپرده‌ها که بر روی حافظه موجود است در همان فایل core.json با همان فرمت اولیه به روز رسانی می‌شوند.

۳. فرمت ورودی‌ها

به ازای هر پایانه فایلی با عنوان terminal.xml وجود دارد که اطلاعات مربوط به پایانه و همچنین تراکنش‌های مالی مرتبط با آن پایانه را مشخص می‌کند. ساختار این فایل به شکل زیر است:

```
<terminal id="21374" type="ATM">
        <server ip="192.168.0.77" port="8080"/>
        <outLog path="term21374.log" />
        <transactions>
                <transaction id="1" type="deposit" amount="3,000" deposit="33227781" />
                <transaction id="2" type="withdraw" amount="10,000" deposit ="35527439" />
                ...
        </transactions>
</terminal>
```

برای سرور نیز فایلی به نام core.json وجود دارد که حاوی اطلاعات زیر است:

```
{
  “port”: 8080,
  “deposits”: [
    {
      “customer”: “Ali Alavi”,
      “id”: “33227781”,
      “initialBalance”: “1,000”,
      “upperBound”: “1,000,000”
    },
    {
      “customer”: “Reza Rezaei”,
      “id”: “35527439”,
      “initialBalance”: “0”,
      “upperBound”: “0”
    }
  ],
  “outLog”: “server.out”
}
```

۴. نکات قابل توجه

- ✓ برای parse کردن XML از parser های موجود در JDK استفاده کنید. اما برای parse کردن JSON لازم است از کتابخانه‌های متن‌باز استفاده کنید.
- ✓ برای مدیریت کتابخانه‌های مورد نیاز همچنین مدیریت نحوه‌ی تولید محصول نهایی از ابزار maven استفاده کنید.
- ✓ بدیهی است که برای انجام درست این پروژه نیاز به critical section handling دارید. شما باید پس از اتمام پروژه‌ی خود، یک سناریوی تست آماده کنید که نشان دهید بدون handling شما خروجی برنامه اشتباه خواهد بود.
- ✓ برای انجام این تمرین مدت ده روز مهلت دارید.

۵. کلیدواژه‌های مرتبط

- ✓ Thread
- ✓ Critical Section
- ✓ Synchronization
- ✓ Socket
- ✓ JSON
- ✓ XML
- ✓ Maven
- ✓ Java.util
- ✓ Git