به نام خدا

تمرین سری –هفتم

مائده بادان فیروز - ۹۸۲۲۲۰۰۹

بهار ۱۴۰۲

مقدمه:

در این سری تمرین مفاهیم پیشرفته‌تر multi thread استفاده شده است.

طراحی و پیاده سازی:

۱) در بخش اول برای محاسبه عدد پی از فرمول لایب نیتز استفاده کردم اما این فرمول دقت بالایی ندارد و با محاسبه‌ی ۱ میلیون جمله از آن تنها تا ۵ رقم اعشار را دقیق محاسبه می کند!

فرمول لایب نیتز

$$\pi = \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \frac{4}{13} - \cdots$$

برای اینکه بتوانیم دقیق‌تر محاسبه کنیم به دنبال فرمول‌های بیشتری گشتم و به فرمول زیر رسیدم که آن هم درست محاسبه نمی‌کند و به طرز عجیبی با تعویض ترتیب ضرب‌ها در صورت به تنهایی یا در مخرج به تنهایی مقادیر متفاوت تولید می‌کند! مثلا اگر ۱۲ را در آخر ضرب کنیم یک جواب می‌گیریم و اگر در ابتدا ضرب کنیم یک جواب دیگر!!!!

$$\frac{1}{\pi} = 12 \sum_{k=0}^{\infty} \frac{(-1)^k (6k)! (13591409 + 545140134k)}{(3k)! (k!)^3 640320^{3k+3/2}}$$

نهایتا هدف استفاده از کلاس BigDecimal است که فرایند کدنویسی آن به درستی پیاده شده و اجرا می‌شود. ضعف از فرمول است که به اندازه‌ی کافی دقیق نیست.

۲) برای این بخش از کلاس CountDownLatch ۳ شیء ساخته شد که هر کدام برای به ترتیب اجرا شدن یک نوع thread قبل از انواع دیگر استفاده شدند. مثلا latch1 در واقع می‌شمارد که چند تا Blackthread دیگر باقی مانده که باقی threadها باید منتظر آن‌ها بماند تا تمام شوند و خودشان بتوانند start شوند. برای اینکه شمارنده بعد انجام کار هر thread یکی کم شود باید در تابع ()run آن thread در پایان کار تابع ()countDown. برای

CountDownLatch صدا شود تا از مقدار شمارنده یکی کم کند وگرنه هیچ گاه انتظار باقی threadها به پایان نمی‌رسد.

۳) برای این بخش از کلاس semaphore یک شی ساخته شد و به هر ۵ thread همان یک شیء پاس داده شد تا هماهنگی به درستی انجام شود. چون می خواستیم در یک زمان فقط دو thrad به ناحیه بحرانی دسترسی داشته باشند در constructor کلاس semaphore عدد ۲ را وارد کردیم و برای اینکه به طور عادلانه -یعنی به ترتیب و در یک صف که یکی به آن وارد می‌شود و یکی از آن خارج (FIFO) - به threadها اجازه‌ی دسترسی به ناحیه بحرانی داده شود attribute به نام fair را در آرگومان دوم از constructor، true وارد کردیم.

بعد از آن در تابع ()run از threadها دقیقا ابتدای ناحیه بحرانی با استفاده از تابع ()acquire. یکی از ظرفیت‌های semaphore را رزرو کردیم. در انتها در یک بلاک finally این ظرفیت گرفته شده را با تابع ()release. آزاد کردیم تا threadهای بعدی بتوانند ظرفیت را بگیرند.

سنجش و ارزیابی:

برای دیدن درستی کارکرد semaphore اطلاعاتی را جهت پرینت در صفحه نمایش تنظیم کردیم که نشان دهد چه threadهایی در انتظار هستند و چه threadهایی در چه زمانی اجازه‌ی دسترسی پیدا کرده‌اند.

برای سنجش دو قسمت دیگر از همان unit testها استفاده شد.

نتیجه‌گیری:

۱) همیشه برای ترتیب و توالی شروع و پایان threadها نیاز به sleep یا join نیست. CountDownLatch قابلیت خیلی خوبی برای کنترل اجرای threadها در زمان درست است.

۲) Semaphore مثل یک قفل است که به جای یک کلید به تعداد دلخواه ما می‌تواند کلید داشته باشد و به همان تعداد اجازه ورود و حضور در یک زمان را بدهد. شیوه عملکرد آن هم این است که بلافاصله بعد از خروج یک عضو، اجازه‌ی ورود بعدی را می‌دهد.