

1.PI Calculator

برای محاسبه عدد پی الگوریتم های مختلفی وجود دارد که من ابتدا از سری Nilakantha با رابطه زیر استفاده کردم :

$$\pi = 3 + 4 / (2*3*4) - 4 / (4*5*6) + 4 / (6*7*8) - \dots$$

چرا که این سری همگراست و با تکرار و جمع ترم های بیشتر محاسبه عدد پی دقیق تر میشود. همچنین باید توجه داشت که اگر سری مورد نظر واگرا باشد تجدید آرایش می توند منجر به نتیجه متفاوت و حتی اشتباه شود! در اینجا چون قرار است هر ترم توسط ترد های مختلفی محاسبه و در نهایت با ترتیب های متفاوتی باهم جمع زده شوند حتما باید از سری های همگرا استفاده نمود. از بیگ دسیمال برای دقت بیشتر در محاسبات و جلوگیری از تخمین، هم چنین از متغیر mc برای تعریف دقت در عملیات جمع، ضرب و... تا سه رقم اعشار استفاده شده است.

اما برای دقت خواسته شده در یونیت تست (بجز تست اول) به 1000,000 تکرار نیاز بود که به لحاظ زمانی مناسب نبود و دستگاه من قادر به اتمام کامپایل کردن کد نشد. به همین منظور از الگوریتم (Bailey–Borwein–Plouffe formula) BBP که دارای سرعت بسیار بیشتری است استفاده کردم .

$$\pi = \sum_{k=0}^{\infty} \left[\frac{1}{16^k} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right) \right]$$

جزئیات کد: از کلاس Math Context متغیر mc برای دقت محاسبات در هر ترم (عملیات جمع، ضرب، تقسیم و... تا چند رقم اعشار دقیق محاسبه شود) هم چنین از کلاس BigDecimal تعریف شده تا با اضافه کردن هر ترم به آن، عدد تخمین زده نشود و مقدار دقیق و بدون تقریب گزارش شود. هم چنین قرار است محاسبه هر ترم به تردهای مختلفی واگذار شود (برای این کار ۸ ترد تعریف کردیم). در خط ۷۱ میزان اعشار خروجی را یک رقم بیشتر از تعداد اعشار خواسته شده (floating point) تعریف کردیم تا مطمئن باشیم که عدد خروجی تا یک رقم اعشار بیشتر درست محاسبه گردیده اما چون تعداد ارقام بعد از

اعشار در خروجی باید دقیقا مطابق تعداد خواسته شده در Unite Test باشد با دستور خط ۷۳ رقم آخر اعشار که اضافی بود را حذف کردیم. هم چنین برای اینکه اضافه کردن ترم ها به مقدار pi می تواند منجر به race condition شود برای جلوگیری از آن در متد add Term از کلمه کلیدی synchronized استفاده کرده ایم. در قسمت main هم میتوانستیم کدی اضافه نکنیم و کد اضافه شده تنها برای دیدن خروجی و مقایسه آن با مقدار اصلی pi بود تا بتوان با توجه به آن کد را اصلاح نمود.

2.Priority Simulator

همان طور که در راهنما هم اشاره شده بود بهترین ابزار برای دادن اولویت به دسته ای از ترد ها استفاده از کلاس Countdown Latch است. زمانی که نیاز است پیش از انجام تسک خاصی حتما دسته از ترد ها شروع به کار کرده و یا حتی کار را به اتمام رسانده باشند بجای اینکه برای تعداد زیادی ترد به صورت دستی از sleep استفاده کنیم که ایمن هم نیست برای آن دسته از ترد ها که دارای اولویت هستند یک شیء از کلاس Countdown Latch می سازیم و ترد ها را به آن واگذار می کنیم سپس با استفاده از متد await به آن ها اولویت می دهیم. متد await به ترد اصلی در حال اجرا اعلام میکند که حتما منتظر بماند تا تردی که متد روی آن اعمال شده ابتدا شروع به کار کند. برای مثال در این کد در کلاس runner پس از اینکه به تعداد تمام black/blue/white ترد ها شیء از کلاس Countdown Latch ساختیم (خط ۳۶، ۳۷ و ۳۸ کد کلاس runner) به تعداد black count ترد از کلاس Black thread می سازیم و آن را به لیست color thread ها اضافه میکنیم و چون اولویت با black thread است نیاز نیست از متد await استفاده کنیم. مشابه همین کار را برای برای blue thread ها انجام می دهیم فقط این بار از متد await. قبل از ساخت whitethread ها استفاده می کنیم تا مطمئن باشیم حتی قبل از ساخت whitethread ها تمام black thread ها کاملاً execute شده

اند. مجددا این کار برای whitethread ها نیز تکرار میشود با این تفاوت که متد await. علاوه بر blackLatch ها روی blueLatch ها نیز اعمال میشود چرا که اولویت آن ها نیز بیشتر است. در نهایت با همین متد، ترد اصلی منتظر execute شدن تمام تردهای رنگی می ماند سپس شروع به کار میکند. در هر سه کلاس black/blue/white thread هم فقط شی CDL از کلاس Countdown Latch تعریف و به کانسراکتور اضافه میشود.

3.Semaphore

درمفاهیم اولیه ترد دیدیم که گاهی برای دسترسی به منبع مشترک گاهی رقابت بین تردها ایجاد میشود که میتواند منجر به خروجی اشتباه گردد. برای این کار از کلاس lock یک شی مانند قفل تعریف کردیم تا تنها تردی که قفل را در اختیار داشت به منبع مشترک دسترسی داشته باشد و فقط وقتی کار آن ترد تمام شد قفل را در اختیار ترد دیگری بگذارد. اما زمانی که مانند این تمرین از ما خواسته شده به بیش از یک ترد دسترسی داده شود از کلاس Semaphore استفاده میشود. پس برای اینکار ابتدا در کلاس Controller یک شی semaphore می سازیم و تعداد تردهایی که اجازه دسترسی به آن (و به تبع دسترسی به منبع مشترک به طور همزمان) را دارند را مطابق با خواسته سوال ۲ قرار میدهیم. سپس در تعریف هریک از اپراتورهای تعریف شده semaphore را هم به عنوادی ورودی اضافه می کنیم. حال در کلاس Operator در متد run ۱۰ بار و در هر مرتبه semaphore به دو اپراتور اجازه دسترسی به منبع را میدهد سپس با متد accessResource. که خودمان آن را در کلاس Resource کامل کردیم بلافاصله پس از دادن دسترسی، نام اپراتوری که به منبع دسترسی دارد همراه زمان دقیق دسترسی اعلام میشود سپس آن دو اپراتور آزاد میشوند تا دو تا اپراتور دیگر دسترسی پیدا کنند. در کلاس Resource هم تنها یک فرمت قابل فهم برای اعلام زمانی دسترسی اپراتورها به منبع تعریف شده و فقط زمان sleep را به ۱۰۰۰ میلی ثانیه افزایش دادم تا در خروجی که دقت زمان بر حسب یک ثانیه است این نکته که به طور همزمان تنها دو تا ترد به منبع

مشترک دسترسی دارند کاملاً مشهود و قابل درک باشد. (البته میشد فرمت
زمان را تبدیل نکرد، در این صورت زمان دسترسی برحسب میلی ثانیه بیان
میشد و هدف کد هم که دسترسی همزمان تنها دو ترد به منبع مشترک بود
قابل مشاهده میشد اما با این تغییرات، کد و نتیجه آن قابل درک تر
میشود.)