1.PI Calculator

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

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

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 درنهایت 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.

البته 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 های دیگر مثل فرمول لایب نیز یا BBP هم استفاده کردم اما آن ها هم قادر به محاسبه عدد پی با این دقت بالا نبودند!

2.PrioritySimulator

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 تردها 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 شئ 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 درحال 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/whiteترد ها شئ 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 colorthreadها اضافه میکنیم و چون اولویت باblackthreadاست نیاز نیست از متد await.استفاده کنیم. مشابه همین کار را برای برایbluethreadها انجام می دهیم فقط این بار از متد await. قبل از ساخت whitethreadها استفاده می کنیم تا مطمئن باشیم حتی قبل از ساخت whitethreadها تمام blackthreadها کاملا execute شده اند.مجددا این کار برای whitethreadih ها نیز تکرار میشود با این تفاوت که متدawait.علاوه برblackLatchها روی blueLatchها نیز اعمال میشود چرا که اولویت آن ها نیز بیشتر است. در نهایت با همین متد، ترد اصلی منتظر executeشدن تمام تردهای رنگی می ماند سپس شروع به کار میکند. در هر سه کلاسblack/blue/white threadهم

فقط شئCDL از کلاس Countdown Latch تعریف و به کانسراکتور اضافه میشود.

3.Semaphore

درمفاهیم اولیه ترد دیدیم که گاهی برای دسترسی به منبع مشترک گاهی رقاب بین تردها ایجاد میشود که میتواند منجر به خروجی اشتباده گردد. برای این کار از کلاس lock یک شئ مانند قفل تعریف کردیم تا تنها تردی که قفل را درختیار داشت به منبع مشترک دستترسی 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 شئ 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 درتعریف هریک از اپراتورهای تعریف شده semaphoreرا هم به عنوادی ورودی اضافه می کنیم.حال در کلاس Operator در متد run ۱۰بار و در هرمرتبه semaphoreبه دو اپراتور اجازه دسترسی به منبع را میدهد سپس با متد accessResourse.که خودمان آن را در کلاس Resourse کامل کردیم بلافاصه 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 Resourse هم تنها یک فرمت قابل فهم برای اعلام زمانی دسترسی اپراتورها به منبع تعریف شده و فقط زمان sleep را به ۱۰۰۰میلی ثانیه افزایش دادم تا درخروجی که دقت زمان برحسب یک ثانیه است این نکته که به طور همزمان تنها دوتا ترد به منبع مشترک دسترسی دارند کاملا مشهود و قابل درک باشد. (البته میشد فرمت زمان را تبدیل نکرد، دراین 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.)