객체지향설계및프로그래밍 실습과제#6

– 원주율 계산기 –

원주율이 초월수라는 사실은 이미 1873년에 수학적으로 증명되었으므로, 유한 시간 안에 실수계에서 원주율 값을 정확히 찾는 것은 불가능합니다. 오늘 우리가 만들어 볼 원주율 계산기는 이러한 원주율의 특성을 감안하여 ‘적어도 무한 번 돌렸을 때는 원주율 값이 정확히 나오는’ 수준을 유지한 채로 가장 멍청한 방법을 써서 연산을 진행해 볼 것입니다(이전에 언급했듯, 우리가 가지고 있는 Random class는 진짜 random은 아니므로 이 프로그램을 진짜 무한 번 돌려 볼 생각은 안 해도 좋습니다).

물론 이번 과제의 목표가 ‘진짜 원주율 찾기’인 것은 아닙니다. 우리가 ‘거의 무한하게 실행해야 할 code’를 다루어야 할 때 이를 Java에서 어떤 방식으로 접근할 수 있는지 살펴 보고, 또한, 그런 접근 방식이 하나가 아님을 느껴 보는 것이 이번 과제의 진짜 목표라 할 수 있을 것입니다.

# ‘직접’ 계산해 보기 (60점)

첫 부분과제는 조교와 함께 수행해 볼 예정이며, 미리 정해 둔 횟수만큼(유한하게) 점을 찍고 그 결과를 출력하는 클래스를 main thread에서 ‘직접’ 사용해 봅니다.

## 과제 수행 순서

* 이번 부분과제에서는 두 가지 클래스를 만듭니다:
  + 프로그램에서 사용할 각종 상수들을 제공할 Constants class
    - 형상화할 정사각형의 너비(Width), 그리고 이 값에서 파생되는 각종 추가 상수들
    - 그리고, 원주율 계산에서 점을 몇 번 찍을지(NumberOfIterations)
  + 원주율 계산 및 출력 기능을 제공할 PiCalculator class
    - 원주율 계산을 시작하는 Start(), 현재까지 계산된 원주율을 출력하는 Print()
* 위의 두 클래스가 완성되면, main()에서 PiCalculator class의 instance를 만든 다음 두 가지 메서드를 순서대로 호출함으로써 프로그램을 완성합니다.
* 순서가 대강 이해되었으면, 이제 다음 페이지부터 나열되는 step들에 따라 과제를 수행해 봅시다.

## 과제 수행 조건

* 이 부분과제는 조교에게 검사받지 않으며, 만약 자신이 이것만 하고 집에 가고 싶은 경우 느낀 점을 제출하고 바로 집에 가도 좋습니다.

## 과제 수행을 위한 세부 step들

* 먼저 Eclipse를 켠 다음 새 Java project를 만들고 능숙한 솜씨로 Program class를 만듭니다.
* Constants class를 만들고 아래의 상수(static final 필드)들을 추가해 줍니다:

|  |
| --- |
| static final int Width = 100;  static final int Center\_X = Width / 2;  static final int Center\_Y = Width / 2;  static final int Radius\_Square = Width / 2 \* Width / 2;  static final int NumberOfIterations = 100000; |

* PiCalculator class를 만들고 아래의 필드들을 추가해 줍니다:

|  |
| --- |
| Random rand; //복붙하고 Random의 m 오른쪽에 커서를 둔 다음 Ctrl + Space로 import부분 추가  Double pi; |

* 이제 **class 안 빈 공간**에서 Ctrl + Space, Enter를 눌러 기본 생성자를 만들고, rand 필드를 새로운 Random class의 instance로 초기화해 줍니다.
* 만들어야 할 두 메서드들 중 더 간단한 Print()를 먼저 만듭니다. Argument나 반환값은 필요 없을 것이고, sysout으로 pi 값을 출력해 주는 한 줄이면 충분하겠군요.
* 이제 가장 어려운 Start()가 남았습니다. 하지만 이미 우리는 여기에 무슨 코드를 적어야 할 지 수학적으로 이해하고 있으므로, 그냥 아래에 있는 내용을 복붙해 주도록 합시다:

|  |
| --- |
| public void Start()  {  int x, y, count\_in = 0;    for ( int i = 0; i < Constants.NumberOfIterations; ++i )  {  x = rand.nextInt(Constants.Width);  y = rand.nextInt(Constants.Width);    x -= Constants.Center\_X;  x \*= x;    y -= Constants.Center\_Y;  y \*= y;    if ( x + y <= Constants.Radius\_Square )  ++count\_in;    pi = (double)count\_in / (i + 1) \* 4.0;  }  } |

* 이제 마지막으로 Program class에 있는 main()을 꾸밀 시간입니다. 아래 세 줄이면 충분합니다:
  + 새로운 PiCalculator class의 instance 만들기
  + Start() 호출하기
  + Print() 호출하기
* 여기까지 다 수행했으면 이제 Ctrl + F11을 눌러 실행해 봅시다. 우리가 이미 알고 있는 원주율과 비슷한 값이 나오면 성공입니다. 만약 3.6 이상 또는 2.7 이하의 값이 나왔다면 오늘 집에 갈 때 ‘로또 슈도랜덤으로 5개 주세요’ 하는 것을 강력하게 권장합니다!

# ‘작업자 thread’를 고용하여 계산을 시켜 보기 (30점)

방금 만든 프로그램은 Start()의 실행이 모두 끝날 때까지 Print()를 호출할 수 없으므로 계산 결과를 보기 위해서는 좀 기다려야 했습니다. 이번에는, 우리 과제의 목표에 좀 더 어울리게 ‘중단할 때까지 알아서’ 점을 찍으며 pi 값을 갱신하도록 PiCalculator class를 개량해 봅시다. 이 과제 역시 아래에 나열된 step들에 따라서 조교와 함께 천천히 진행해 보도록 하겠습니다.

## 과제 수행을 위한 세부 step들

* 먼저, 이번에 작성할 프로그램이 어떻게 동작할 지 이해해 봅시다:
  + 프로그램이 시작되면 즉시 원주율 계산을 ‘시작’합니다.
  + 사용자가 엔터 키를 칠 때마다(‘한 줄 입력’을 할 때마다) 현재 계산된 값을 ‘출력’합니다.
  + 만약 이 때 q를 누르고 엔터 키를 쳤다면 실행을 ‘중단’합니다.
* Start()와 Print()는 이미 있으니, 이제 PiCalculator class에 Stop()을 추가해 둡니다. 메서드 내용은 일단 그냥 빈 칸으로 둡시다.
* 이제 main()을 위의 동작에 맞게 다시 작성해 봅시다. 이 step은 우리 과제의 주 목표는 아니니 그냥 아래 코드를 복붙해 주도록 합시다:

|  |
| --- |
| public static void main(String[] args)  {  PiCalculator calc = new PiCalculator();  Scanner scanner = new Scanner(System.in); // 역시나 import 필요    calc.Start();    while ( true )  {  String input = scanner.nextLine();    calc.Print();    if ( input.equals("q") == true )  {  calc.Stop();  break;  }  }    scanner.close();  } |

* 참고: 위의 무한 루프는 사용자 입력을 ‘사용자가 원하는 만큼’ 받기 위해 사용되고 있습니다.
* 이제 남은 것은 PiCalculator class에 ‘심장’을 때려 넣고 연결해 주는 것입니다. 이를 통해 이제 PiCalculator class는 ‘실행을 당하는 클래스’에서 ‘실행을 하는 클래스’로 전직하게 됩니다.
* 먼저 thread 사용에 대한 약간의 관용적 표현을 알아 볼 필요가 있습니다. PiCalculator class에 아래의 필드들을 추가해 봅시다:

|  |
| --- |
| private Thread thr\_runner; // Thread 그 자체  private boolean isStarted; // 내 주인에 의해 Start()가 호출되었는지를 나타냄  private boolean isStopRequested; // 내 주인에 의해 Stop()이 호출되었는지를 나타냄 |

* 위의 세 필드들은 Java에서 thread를 사용할 때 일반적으로 만들어 쓰는 필드들입니다. isStarted 필드는 ‘이미 실행중인데 또 Start()를 실행하는 것을 방지’하기 위해 사용되며, isStop어쩌구 필드는 thread에게 하던 일을 정지할 것을 정중하게 요청하기 위해 사용됩니다(아래에서 또 나옴).
* 이제 thread의 진입점 메서드를 초기화해 줄 수 있도록 생성자를 수정할 시간입니다. 생성자 코드 안에 아래의 코드를 추가해 주세요:

|  |
| --- |
| thr\_runner = new Thread(() -> {  int x;  int y;  int count\_in;    count\_in = 0;    for ( int i = 0; **isStopRequested == false**; i++ )  {  x = rand.nextInt(Constants.Width);  y = rand.nextInt(Constants.Width);    x -= Constants.Center\_X;  x \*= x;    y -= Constants.Center\_Y;  y \*= y;    if ( x + y <= Constants.Radius\_Square )  ++count\_in;    pi = (double)count\_in / (i + 1) \* 4.0;  }  }); |

* 네, 굵은 글씨로 강조해 둔 부분만 빼면 이전에 만들었던 Start()와 완벽히 동일합니다!
* 좀 길지만 이 구문은 결국 새로운 Thread instance를 만드는 코드입니다. 아까 조교가 thread는 ‘진입점 메서드’가 필요하다고 했지요? 여기선 람다 식을 써서 해당 메서드를 정의하고 있습니다.
* 눈여겨 볼 점들 중 하나는, 명백히 PiCalculator instance가 갖는 필드(pi 등)를 람다 식 안에서 별 부담 없이 그냥 가져다가 쓰고 있다는 것입니다. 나중에 여유가 있을 때 이 부분에 대해 조금 더 고민해 보면 나름 다양한 상상들을 해 볼 수 있을 것입니다.
* 일단 남은 메서드들을 마저 작성해 봅시다. 이전에 적어 둔 메서드들(생성자 제외)을 모두 지우고 아래 코드를 복붙해 주세요:

|  |
| --- |
| public void Start()  {  if ( isStarted == false )  {  isStarted = true;    isStopRequested = false;  thr\_runner.start();  }  }    public void Print()  {  System.out.println(pi);  }  public void Stop()  {  isStopRequested = true;  } |

* 다 되었습니다. 이제 프로그램을 실행해 보고 앞에 적혀 있던 동작들을 잘 수행하고 있나 확인해 봅시다.

## 추가로 해 볼 만한 것들

아래에 있는 것들은 이번 과제에는 포함되지 않지만 지금 한 번 해 볼 만한 것들입니다:

* Eclipse의 debugger 기능 및 Debug perspective를 사용하여 thr\_runner.start()가 호출되기 전 / 후에 Debug 탭(좌측 상단에 있는 창)에 어떤 변화가 생기는지 확인해 보세요.
  + 각 thread의 이름, 그리고 현재 상태(Running인지 Suspended인지)를 주목해 봅시다.
* Debug 탭에서 Running 상태인 thread를 누르고 메뉴에 보이는 일시정지 버튼을 누르면 해당 thread를 suspend하고 call stack을 확인할 수 있습니다.
  + Main thread의 call stack 맨 아래에는 main()이 있고, Thread-0 thread의 경우에는 아래 두 줄을 제외하면 우리가 만든 람다 식이 있는 것을 확인할 수 있습니다.

# LOOT를 사용하여 계산 상황을 실시간으로 나타내 보기 (팀 과제, 10점)

이제까지 우리는 ‘계속 실행해야 할 필요가 있는’ 작업을 수행하는 두 가지 방법, main thread에서 직접 하는 것과 별도의 작업자 thread를 만들어 시키는 것을 살펴 보았습니다. 이 둘은 여러분이 프로그램 실행 전반에 대한 설계를 직접 수행할 때 (본격적으로 만들긴 귀찮아서) 자주 쓰게 될 방법들입니다.

이번 부분과제에서는, 이전과는 다르게, LOOT에 의해 1초에 60번 실행’당하는’ 환경에서 위와 동일한 작업을 수행하도록 프로그램을 작성해 봅시다.

## 과제 수행 조건

여러분이 만드는 프로그램은 아래 예시와 같이 동작할 수 있어야 합니다:

|  |  |
| --- | --- |
|  |  |
| 실행하고 잠시 뒤 찍은 화면 | 조금 더 기다린 다음 찍은 화면 (직접 만들어서 확인해 보세요!) |

* 구체적으로, 여러분의 프로그램은 다음 기능들을 제공해야 합니다:
  + 일단, **창 크기는 정사각형**이어야 합니다.
  + 실행하면 매 frame마다…
    - 새로운 점을 찍습니다. 이 때, 내접원에 들어가는 점과 그렇지 않은 점을 서로 다른 image로 표현합니다.
    - 현재 찍은 점 수와 내접원에 들어간 점 수, 그리고 현재 찾은 원주율을 출력합니다.
  + 이 작업을 창이 꺼질 때까지 계속 반복합니다.

## 세부 진행 방법

* 일단 지난 주 과제에서 했던 것처럼 새 Java project를 만들고 LOOT를 사용할 수 있도록 build path를 설정해 줍니다.
* GameFrame class를 상속받는 새 클래스를 만듭니다.
* Program class와 main()을 만들고, 방금 만든 창을 띄우는 코드를 추가해 줍니다.
* 설정 instance(예제의 settings)를 통해 창이 정사각형 영역을 갖도록 너비 / 높이를 맞춥니다.
* 아래 필드들을 추가해 주고, 생성자 및 Initialize()를 통해 적절히 초기화해 줍니다:
  + Random 하나
  + Layer 하나(Viewport가 3차원 좌표계의 것을 그려 준다면 Layer는 2차원 것을 그려 줍니다)
  + TextBox 하나(만들자 마자 Layer에 미리 담아 둡니다)
  + 점 수를 집계하기 위한 정수 필드 두 개(전체 점 수, 내접원 안에 들어간 점 수)
  + 각종 상수들 – 점의 너비, 점이 가질 수 있는 x / y좌표의 최대값, ‘중점’의 x / y좌표
    - Note: 2차원 평면에서 점의 x, y좌표는 해당 점 그림의 ‘맨 왼쪽 위’ 픽셀의 좌표입니다.
* Update()를, 아래 작업을 순서대로 수행하도록 구성합니다:
  + 임의의 x, y좌표(최소값은 0, 최대값은 위에서 정한 상수)를 갖는 새로운 점을 하나 만듭니다.
  + 해당 점이 내접원 안에 들어가는지 여부를 검사합니다.
    - 그 결과에 따라 점을 그릴 image를 서로 다르게 부여해 줍니다.
    - 동시에 집계용 정수 필드 또는 필드들을 ++해 줍니다.
  + 점을 Layer에 추가합니다. Layer가 가진 children 필드에 add()를 써서 추가해 주면 됩니다.
  + 마지막으로, TextBox의 내용을 갱신해 줍니다.
* Draw()를 적당히 복붙하여 Layer를 그리도록 지정해 줍니다. 이번 프로그램의 경우 각 점들의 위치가 변하지 않으므로 ClearScreen()은 매 번 호출하지 않아도 됩니다. 다만 이 경우 TextBox의 크기를 너무 작게 지정해 두었다면 글씨가 겹쳐 써질 수 있으니 미리 적당히 크게 잡아 두세요.
* 이제 실행하면 전 페이지 예시와 유사하게 예쁘게 실행되는 우리 프로그램을 볼 수 있습니다!
  + 만약 그렇지 않은 경우 조심스럽게 코드를 다시 살펴 보거나 조교의 도움을 요청해 주세요.

## 주의사항

* 어떤 개발자는 GUI에 익숙하지만, 다른 개발자는 그렇지 않을 수 있습니다. 따라서, 과제 수행에 시간이 좀 걸리는 것은 그러한 개발자마다의 성향 차이를 극복해 가는 과정으로 볼 수 있습니다.
* 다 완성되면 조교에게 검사를 요청해 주세요. 조교가 ㅇㅋ 하면 성공입니다.