Hw1_1

2차원 유클리드 좌표계에서 중심의 좌표가 (x, y)이고 반지름이 r인 원 두개를 입력 받는다.

여기서 나의 프로그램은 입력으로 $x, y, r \in \left\{ z \in \mathbb{Z} \mid -2^{31} \leq z < 2^{31} \right\}$만 허용할 것이다.

```
int x1 = sc.nextInt(), y1 = sc.nextInt(), r1 = sc.nextInt();
```

처음에는 입력을 위와 같이 comma separator로 받았는데, 수행 순서가 보장되는지 확실하지 않아서 찾아보았는데, Stack Overflow에 다음과 같은 결과가 나왔다. 하지만 Java 공식 문서에는 정확한 명세가 없어서 안전하게 다음과 같이 바꾸었다.

```
int x1 = sc.nextInt(); int y1 = sc.nextInt(); int r1 = sc.nextInt();
```

이렇게 하니까 코드가 지저분해져서 원을 객체화 해야겠다는 생각이 들었다.

Circle 클래스를 정의하자.

```
class Circle {
  int x, y, r;
  Circle(Scanner sc) {
    x = sc.nextInt();
    y = sc.nextInt();
    r = sc.nextInt();
  }
}
```

쉽게 입력을 받기 위해서 위와 같이 생성자 안에서 입력을 받을 것이다.

두 원이 겹치는지 판단하기 위해 생각할 점이 두가지가 있다.

원과 원판은 엄연히 다르다. 위상학적으로 원은 1차원 공간이고 원판은 2차원 공간이다.

따라서 두 원이 겹친다는 것은 두 원의 교집합이 공집합이 아니라는 뜻이다.

정의에 따라서 intersect 함수를 작성해보자.

두 원의 중심 사이의 거리를 d, 두 원의 반지름을 각각 r_1, r_2 라고 하면,
$|r_1 - r_2| \leq d \leq r_1 + r_2$ 와 두 원이 겹치는 것은 동치이다.

```
static boolean intersect(Circle c1, Circle c2) {
  long d = (long)(c1.x - c2.x) * (c1.x - c2.x) + (long)(c1.y - c2.y) * (c1.y - c2.y);
  return ((long)c1.r - c2.r) * ((long)c1.r - c2.r) <= d && d <= ((long)c1.r + c2.r) * ((long)c1.r + c2.r);
}
```

제곱을 계산하면 int 범위를 넘어갈 수 있으므로 long으로 type casting을 해주었다.

Hw1_2

문제의 입력으로 주어진 모든 식은 다음과 같이 나타낼 수 있다.

expression ::= value | expression ‘ + ’ value | expression ‘ - ’ value
value ::= double | value ‘ * ’ double | value ‘ / ’ double
double $\in \{\text{String s} \mid \text{Double.parseDouble(s) does not throw an exception}\}$

위 정의를 따르면 쉽게 재귀함수로 값을 구할 수 있다. 오른쪽부터 처리하면서 원하는 연산자가 있는지를 찾으면 된다.

여기서 만약 0으로 나누게 된다면 "0으로 나눌 수 없습니다."를 출력해야 하는데, Double로 처리할 경우 실수 오차가 발생할 수 있어서 단순히 0이 아닌지 확인하면 위험하다.

하지만 위 식 정의에서 나눗셈 기호 다음에 오는 것은 double 뿐이다. 이는 다른 말로 하면, double이 아닌 value로는 나눌 수 없다는 뜻이다. 즉, 0인지 확인할 때, double의 최소 절대치보다 절대치가 작은지만 확인해도 논리적으로 오류가 없다는 뜻이다. 이 경우는 0밖에 없으므로, 그냥 0과 비교해도 무관하다.

한 줄을 통으로 문자열로 입력 받은 후, evalExpr의 인자로는 받은 문자열과 처리해야 할 부분문자열의 구간을 두 개의 int로 받았다.

String은 클래스이므로, 인자로 받을 때 참조를 할 것이므로 String 클래스를 계속 생성하는 일은 없을 것이다. 만약에 부분 문자열 자체를 인자로 넘겨준다면, 새로운 String을 생성하므로 메모리를 더 많이 쓸 것이다. 이 부분에서 메모리 사용을 줄였다.

각 실수는 evalDouble을 한번만 부르도록 설계되어있다. 따라서 재귀 스택을 무시한다면 이론적으로 프로그램은 선형 시간에 동작한다.

Hw1_3

랜덤하게 삽입하라는 뜻은 정확히 무엇일까?

우리가 1 이상 10 이하의 자연수 중에 하나를 랜덤하게 선택하라고 하면 보통 각각의 수가 나올 확률이 1 / 10 이라고 생각한다.

만약 1만 나오고 다른 수가 나오지 않는다면, 이를 랜덤하게 선택했다고 할 수 있을까?

이에 대한 답은 관점에 따라 다르지만 나는 정의에 따르면 YES라고 대답할 것이지만 일반적으로는 NO라고 할 것이다.

1 이상 100 이하의 자연수 중 $0 \leq k \leq 100$개의 서로 다른 자연수를 나열하는 방법은 $100! / (100-k)!$ 가지이다.

나는 Math.floor(Math.random() * 100) + 1의 값을 확률변수 X 라고 할 때 모든 $1 \leq i \leq 100$ 에 대하여 $P(X=i) = 1 / 100$ 을 만족시킨다는 가정하에 각각의 방법이 모두 같은 확률로 나오게 프로그램을 작성할 것이다.

```
static int genNextInt() {
  return (int)(Math.floor(Math.random() * 100)) + 1;
}
```

알고리즘은 다음과 같다.

```
1   for (int i = 0; i < gen; i++) {
2     boolean valid;
3     do {
4       int x = genNextInt();
5       arr[i] = x;
6       valid = true;
7       for (int j = 0; j < i && valid; j++) {
8         if (arr[j] == x) valid = false;
9       }
10    } while (!valid);
11  }
```

각 시행마다 남은 수 중에서 균등하게 하나의 수를 뽑기 때문에 균등하게 k개를 나열한다는 사실은 자명하다.

기대 실행 시간은 어떻게 될까?

$$T(n) = \sum_{k=0}^{n-1} \frac{100 \cdot k}{100-k}$$

더 간단히 나타내기는 어렵지만 $n = 100$ 일 때 적분으로 어림잡아 계산하면, 평균 50000번 정도의 기본 연산에 프로그램이 정상적으로 종료됨을 알 수 있다.

Hw1_4

위 문제와 비슷한 방법으로 풀면 마찬가지로 문제에 조건에 맞는 출력들 중 균등한 확률로 뽑힌다는 것을 쉽게 증명할 수 있다.

Java에서는 배열을 생성할 때 모두 메모리가 0으로 초기화 됨이 보장되기 때문에 따로 0으로 모두 초기화 시킬 필요가 없다.