Algorithm 05 중복순열 (가위 바위 보)

```
1 usage  new *
public static ArrayList<String[]> rockPaperScissors(int rounds) {
    // TODO:
    // 결과를 담을 ArrayList를 선언한다.
    ArrayList<String[]> outcomes = new ArrayList<>();
    // 함수를 실행해 반환된 결과를 다시 반환한다.
    return permutation( curRounds: 0, rounds, new String[rounds], outcomes);
}

// 배열의 모든 요소의 경우의 수를 체크하기 위해 재귀 함수를 사용한다.
// rounds를 넣을 변수 roundsToGo, 일회용 배열인 playedSoFar 변수를 선언한다.
2 usages  new *
public static ArrayList<String[]> permutation(int curRounds, int roundsToGo, String[] playedSoFar, ArrayList<String[]> outcomes) {
    // 재귀 탈출조건 (base case)
    if (roundsToGo == curRounds) {
        String[] arr = Arrays.copyOf(playedSoFar, playedSoFar.length);
        outcomes.add(arr);
        return outcomes;
    }
    // [rock, paper, scissors]를 요소로 갖는 str 배열을 선언한다.
    String[] rps = new String[]{"rock", "paper", "scissors"};
    // str 배열을 한 번씩 순회한다.
    for (int i = 0; i < rps.length; i++) {
        // rps의 i번째 요소를 변수에 담는다.
        String currentPlay = rps[i];
        playedSoFar[curRounds] = currentPlay;
        // 일회용 배열의 크기를 rounds만큼 맞춰주기 위해, rounds에서 1을 뺀다. [rock, rock, rock]
        outcomes = permutation( curRounds: curRounds + 1, roundsToGo, playedSoFar, outcomes);
    }
    // 값을 반환한다.
    return outcomes;
}
```

outcomes = ... 과 같이 할당하지 않아도 return이 되는 이유 .
outcomes의 타입이 ArrayList (레퍼런스 타입) 이므로 주소값만 전달함 → 바로 할당이 가능함

0부터 시작

같아질 때 재귀 종료

1 입력 → 3개
2 → 9개
3 → 27개
4 → 81개
=> 총 개수: 3^{n} (n → rounds)

실제로 주소값만 들어감

깊은 복사

→ playedSoFar가 아닌 이유 (6:30)
playedSoFar를 넣으면 항상 동일한 배열이 들어감
재귀로 배열의 요소를 계속 변경하는데 배열을 새로 선언하거나 할당하지 않으면, 마지막에 바뀐 요소로 전부 바뀌게 됨

rps 순회 (재귀)

roundsToGo = 3

rps = [{ rock, paper, scissors }]
(0, 1, 2)

playedSoFar = [null, null, null]

	i	curRounds	currentPlay	playedSoFar
start	0	0	rock	[rock, null, null]
	0	1 (+1)	rock	[rock, rock, null]
	0	2	rock	[rock, rock, rock] → outcomes.add(arr)
depth를 하나 되돌아감	1	2 (3) ↓ curRounds(3) = roundsToGo(3)	paper	[rock, rock, paper] → outcomes.add(arr)
	2	2	scissors	[rock, rock, scissors] → outcomes.add(arr)
	1	1	paper	[rock, paper, null]
	0	2	rock	[rock, paper, rock] → outcomes.add(arr)
	1	2	papper	[rock, papper, scissors] → outcomes add(arr)

outcomes

[
{rock, rock, rock},
{rock, rock, paper}
{rock, rock, scissors}
{rock, paper, rock}
{rock, paper, paper}
{rock, paper, scissors}
{rock, scissors, rock}
{rock, scissors, paper}
{rock, scissors, scissors}

{paper, rock, rock}
{paper, rock, paper}
{paper, rock, scissors}
⋮
]