가장 긴 팰린드롬 부분 문자열

문제 : 가장 긴 팰린드롬 부분 문자열을 출력하라.
예시1)
Input: s = "dcbabcdd"
Output: "dcbabcd"

"최장 공통 부분 문자열"이라는 문제가 있다
여러 개의 문자열이 있을 때 서로 공통된 가장 긴 부분 문자열을 찾는 문제로, DP(Dynamic Programing)로 풀 수 있다.
그러나 이 문제의 경우 DP를 이용한 풀이는 직관적으로 이해가 어렵고, 무엇보다 일반적인 예상과 달리 실행 속도가 느리다. 따라서 여기서는 좀 더 직관적이고 성능이 훨씬 더 좋은 "투 포인터"가 팰린드롬을 발견하면 확장하는 형태로 풀이해 볼 수 있다.
팰린드롬만 판별하면 된다는 점에서 착안해서, 매칭이 될 때 중앙을 중심으로 점점 확장해 나가면서 가장 긴 팰린드롬을 판별하는 알고리즘을 구현해보자.

d	c	b	a	b	c	d	d

2칸, 3칸으로 구성된 투포인터가 슬라이딩 윈도우 처럼 계속 앞으로 전진해 나간다. 이때 윈도우에 들어온 문자열이 팰린드롬인 경우 그 자리에 멈추고, 투 포인터는 점점 확장해 나가는 방식이다.
위 표에서 dd 처럼 짝수일 때도 있고, bab, dcd 처럼 홀수일 때도 있다.

- 짝수는 (2칸, 2 -> 4 -> 6)으로 증가해 나가고, 홀수는 (3칸, 1 -> 3 -> 5)으로 증가해 나간다.
  따라서 짝수와 홀수 둘 다 판별이 가능하다.

2개의 투 포인터가 계속 우측으로 이동하다 3칸짜리 투 포인터는 bab에서 앞,뒤 b가 팰린드롬으로 매칭되어 cbabc, dcbabcd까지 확장되면서 가장 긴 값으로 저장된다. 이 외에 2칸짜리 투포인터도 마지막 부분의 dd에서 매칭되지만 마지막 문자이기 때문에 오른쪽 방향으로 더 이상 확장되지 못하고 그대로 종료된다.

홀수(3칸)	d	c	b	a	b	c	d	d
짝수(2칸)	d	c	b	a	b	c	d	d

코드로 한번 구현해보자.
먼저 길이가 1인 문자열도 입력값이 될 수 있기 때문에 그대로 돌려주는 예외처리를 하고 가자.

```
int len = s.length();
if ( len < 2){
        return s;
}
```

이제 입력 문자열을 두고 우측으로 한칸씩 이동하면서 짝수(2칸)와 홀수(3칸) 2개의 투포인터로 조사해 나간다.

```
for (int i = 0; i < len - 1; i++){
        extendPalindrome(s, i, i + 1);
        extendPalindrome(s, i, i + 2);
}
```

extendPalindrome()으로 정의한 함수는 짝수(2칸)와 홀수(3칸) 2개의 투 포인터가 팰린드롬 여부를 판단하변서 슬라이딩 윈도우처럼 계속 우측으로 이동한다.
extendPalindrome() 함수 내부에서는 양쪽 끝 문자가 일치하는 경우 팰린드롬으로 보고 윈도우 크기를 점점 더 키운다. 이렇게 판별한 최대 길이에 해당하는 문자열이 최종 결과가 된다.

```
int left, maxLen;

public void extendPalindrome(String s, int j, int k){
        // 투 포인터가 유효한 범위 내에 있고, 양쪽 끝 문자가 일치하는 팰린드롬인 경우 범위 확장
        while (j >= 0 && k < s.length() && s.charAt(j) == s.charAt(k)){
                j--;
                k++;
        }
        // 기존 최대 길이보다 큰 경우 값 교체
        if (maxLen < k - j - 1){
                left = j + 1;
                maxLen = k - j - 1;
        }
}

public String longestPalindrome(String s){
        // 문자 길이 저장
        int len = s.length();
        // 길이가 1인 경우 예외 처리
        if (len < 2){
                return s;
        }
        //
        for (int i = 0; i < len - 1; i++){
                extendPalindrome(s, i, i + 1); // 짝수(2칸) 투 포인터
                extendPalindrome(s, i, i + 2); // 홀수(3칸) 투 포인터
        }
```

```
        return s.substring(left, left + maxLen);
    }
```