📖 자동차 경주 - 미니과제

미니과제 진행 방식

진행 방식

- 미션은 **과제 진행 요구 사항**, **기능 요구 사항**, **프로그래밍 요구 사항** 세 가지로 구성되어 있다.
- 세 개의 요구 사항을 만족하기 위해 노력한다. 특히 기능을 구현하기 전에 기능 목록을 만들고, 기능 단위로 커밋 하는 방식으로 진행한다.
- **기능 요구 사항에 기재되지 않은 내용은 스스로 판단하여 구현한다.**

미션 제출 방법

- 미션 구현을 완료한 후 GitHub을 통해 제출해야 한다.
  - GitHub을 활용한 제출 방법은 미니과제 진행 가이드 문서를 참고해 제출한다.
  - 풀 리퀘스트 제목은 '[OO대 트랙_본명] 미션 제출합니다.'로 작성한다.
- GitHub에 미션을 제출한 후 구글 폼에 PR 링크를 포함하여 최종 제출한다.

과제 진행 소감

과제를 수행하면서 느낀 점, 배운 점, 많은 시간을 투자한 부분 등 자유롭게 작성한다.

예시

```
이번 미션은 생각만큼 쉽지 않았습니다.
특히 기능을 분리해서 기능 단위로 커밋하는 것이 쉽지 않다는 것을 깨달았습니다.
기능 단위로 분리하고 커밋하기 위해 다양한 방법으로 연습하였습니다.
하지만 일단 기능을 분리해서 구현하고 나니 좀 더 명확하게 구현할 수 있다는 것을 느낄 수 있었습니다.
미션을 수행하면서 더욱 성장한 걸 느꼈어요. :)
```

과제 제출 전 체크 리스트

- 터미널에서 `java -version` 을 실행하여 Java 버전이 17인지 확인한다. Eclipse 또는 IntelliJ IDEA와 같은 IDE에서 Java 17로 실행되는지 확인한다.
- 터미널에서 Mac 또는 Linux 사용자의 경우 `./gradlew clean test` 명령을 실행하고, Windows 사용자의 경우 `gradlew.bat clean test` 또는 `./gradlew.bat clean test` 명령을 실행할 때 모든 테스트가 아래와 같이 통과하는지 확인한다.

```
BUILD SUCCESSFUL in 0s
```

자동차 경주

과제 진행 요구 사항

- 미션은 자동차 경주 저장소를 포크하고 클론하는 것으로 시작한다.
- **기능을 구현하기 전 `README.md` 에 구현할 기능 목록을 정리**해 추가한다.
- Git의 커밋 단위는 앞 단계에서 `README.md` 에 정리한 기능 목록 단위로 추가한다.
  - AngularJS Git Commit Message Conventions을 참고해 커밋 메시지를 작성한다.

- 자세한 과제 진행 방법은 미니과제 진행 가이드 문서를 참고한다.

기능 요구 사항

초간단 자동차 경주 게임을 구현한다.

- 주어진 횟수 동안 n대의 자동차는 전진 또는 멈출 수 있다.
- 각 자동차에 이름을 부여할 수 있다. 전진하는 자동차를 출력할 때 자동차 이름을 같이 출력한다.
- 자동차 이름은 쉼표(,)를 기준으로 구분하며 이름은 5자 이하만 가능하다.
- 사용자는 몇 번의 이동을 할 것인지를 입력할 수 있어야 한다.
- 전진하는 조건은 0에서 9 사이에서 무작위 값을 구한 후 무작위 값이 4 이상일 경우이다.
- 자동차 경주 게임을 완료한 후 누가 우승했는지를 알려준다. 우승자는 한 명 이상일 수 있다.
- 우승자가 여러 명일 경우 쉼표(,)를 이용하여 구분한다.
- 사용자가 잘못된 값을 입력할 경우 `IllegalArgumentException` 를 발생시키고, "[ERROR]"로 시작하는 에러 메시지를 출력 후 그 부분부터 입력을 다시 받는다.
  - `Exception` 이 아닌 `IllegalArgumentException` , `IllegalStateException` 등과 같은 명확한 유형을 처리한다.

실행 결과

```
경주할 자동차 이름을 입력하세요.(이름은 쉼표(,) 기준으로 구분)
pobi,woni,jun
시도할 회수는 몇회인가요?
5

실행 결과
pobi : -
woni :
jun : -

pobi : --
woni : -
jun : --

pobi : ---
woni : --
jun : ---

pobi : ----
woni : ---
jun : ----

pobi : -----
woni : ----
jun : -----

최종 우승자 : pobi, jun
```

프로그래밍 요구 사항 1

- JDK 17 버전에서 실행 가능해야 한다.
- 프로그램 실행의 시작점은 `Application` 의 `main()` 이다.
- `build.gradle` 파일은 변경할 수 없으며, 제공된 라이브러리 이외의 외부 라이브러리는 사용하지 않는다.
- 프로그램 종료 시 `System.exit()` 를 호출하지 않는다.
- 프로그래밍 요구 사항에서 달리 명시하지 않는 한 파일, 패키지 등의 이름을 바꾸거나 이동하지 않는다.

프로그래밍 요구 사항 2

- 자바 코드 컨벤션을 지키면서 프로그래밍한다.
  - 기본적으로 Google Java Style Guide을 원칙으로 한다.
  - 단, 들여쓰기는 '2 spaces'가 아닌 '4 spaces'로 한다.
- indent(인덴트, 들여쓰기) depth를 3이 넘지 않도록 구현한다. 2까지만 허용한다.
  - 예를 들어 while문 안에 if문이 있으면 들여쓰기는 2이다.

    - 힌트: indent(인덴트, 들여쓰기) depth를 줄이는 좋은 방법은 함수(또는 메서드)를 분리하면 된다.
- 3항 연산자를 쓰지 않는다.
- 함수(또는 메서드)가 한 가지 일만 하도록 최대한 작게 만들어라.
- JUnit 5와 AssertJ를 이용하여 정리한 기능 목록이 정상적으로 작동하는지 테스트 코드로 확인한다.
    - 테스트 도구 사용법이 익숙하지 않다면 아래 문서를 참고하여 학습한 후 테스트를 구현한다.
        - JUnit 5 User Guide
        - AssertJ User Guide
        - AssertJ Exception Assertions
        - Guide to JUnit 5 Parameterized Tests

프로그래밍 요구 사항 3

- 함수(또는 메서드)의 길이가 15라인을 넘어가지 않도록 구현한다.
    - 함수(또는 메서드)가 한 가지 일만 잘 하도록 구현한다.
- else 예약어를 쓰지 않는다.
    - else를 쓰지 말라고 하니 switch/case로 구현하는 경우가 있는데 switch/case도 허용하지 않는다.
    - 힌트: if 조건절에서 값을 return하는 방식으로 구현하면 else를 사용하지 않아도 된다.
- 도메인 로직에 단위 테스트를 구현해야 한다. 단, UI(System.out, System.in, Scanner) 로직은 제외한다.
    - 핵심 로직을 구현하는 코드와 UI를 담당하는 로직을 분리해 구현한다.
    - 힌트: MVC 패턴 기반으로 구현한 후, View와 Controller를 제외한 Model에 대한 단위 테스트 추가에 집중한다.