```java
// @RequiredArgsConstructor -> 생성자 자동생성 어노테이션
@Service
public class ArticleService {

    private final BlogRepository blogRepository;
    // BlogRepository 타입의 변수를 선언하고 final로 지정함.
    // final로 선언했기 때문에 한 번 초기화하면 다른 객체로 바꿀 수 없고,
    // 이 변수는 ArticleService 내에서 Repository를 통해 DB 접근에만 사용됨.
    // Repository는 DB 연동 로직을 캡슐화한 객체이므로 Service 계층에서는 직접 SQL을 작성하지 않고 여기서 제공하는 메서드(save, findAll 등)를 활용함.

    public ArticleService(BlogRepository blogRepository) {
        this.blogRepository = blogRepository;
    }
    // 생성자 정의. ArticleService 객체가 생성될 때 BlogRepository를 주입받음.
    // Spring 컨테이너는 ArticleService Bean을 만들 때 이미 생성된 BlogRepository Bean을 전달함.
    // 이렇게 하면 의존성 주입(DI) 방식으로 Service와 Repository를 연결할 수 있음.
    // 생성자를 통한 주입은 final 변수 초기화가 가능하고, 테스트용 Mock 객체 주입도 용이함.

    public ArticleDto createArticle(ArticleDto dto) {
        Article article = new Article(dto.getTitle(), dto.getContent());
        // 클라이언트나 Controller에서 전달받은 ArticleDto의 데이터를 기반으로 Entity 객체를 생성함.
        // Entity는 실제 DB에 저장 가능한 객체이며, DTO는 계층 간 데이터 전달용이므로 변환 필요.

        Article saved = blogRepository.save(article);
        // Repository의 save() 메서드를 호출하여 DB에 Entity를 저장함.
        // save()는 DB에 삽입 후, ID가 자동 생성된 완전한 Entity 객체를 반환함.
        // 이 반환 객체는 DB 저장 상태와 ID를 포함하므로 이후 로직에서 그대로 활용 가능.

        return new ArticleDto(saved.getId(), saved.getTitle(), saved.getContent());
        // 저장된 Entity 객체를 다시 DTO로 변환함.
        // Controller나 클라이언트에게 반환할 때 DTO를 사용하면 DB Entity 노출을 막고, 계층 간 독립성을 유지할 수 있음.
        // 반환된 DTO는 ID, 제목, 내용 필드를 포함하며, Service에서 처리 후 Controller로 전달됨.
    }

    public List<ArticleDto> getAllArticles() {
        return blogRepository.findAll()
                // Repository의 findAll() 호출로 DB에 있는 모든 Article Entity를 조회함.
                // findAll()은 List<Article> 형태로 반환되며, DB 상태 그대로 가져옴.

                .stream()
                .map(a -> new ArticleDto(a.getId(), a.getTitle(), a.getContent()))
                // 스트림을 이용해 각 Entity 객체를 DTO로 변환함.
                // Entity 내부 구조를 외부에 그대로 노출하지 않고, 필요한 데이터만 추출해 DTO에 담음.
                // 이렇게 하면 Controller나 View 계층에서 DB Entity에 직접 접근할 필요가 없어지고, 계층 분리가 명확해짐.
```

```
                .toList();
                // 변환된 DTO 객체들을 리스트로 수집하여 반환함.
                // 결과적으로 Service 계층에서 DB 조회 → Entity → DTO 변환 →
Controller 반환까지 전체 흐름을 처리함.
    }
}
```