# **스프링 백엔드 개발자를 위한 Java 기초 다지기: 콘솔 할 일 목록 앱 만들기**

## **I. 서론: 여정의 시작**

### **A. 현재 상황과 목표 설정**

스프링 백엔드 개발자로의 여정을 시작하신 것을 환영합니다. 현재 2년차 개발자로서 프론트엔드 개발 경험을 포함하여 Arduino, Python, Express, Nest, React 등 다양한 기술 스택을 경험하며 프로젝트를 수행해 오셨습니다. 특히, 전문 인력 부재 및 체계적인 가이드 없이 AI의 도움을 받아 다양한 요구사항에 대응하며 파편화된 프로젝트를 진행해 오신 경험은 쉽지 않은 환경이었을 것입니다. 이러한 경험 속에서 백엔드 개발, 특히 한국 채용 시장에서 수요가 높은 스프링(Spring) 기반 기술로 방향을 설정하신 것은 매우 전략적인 결정입니다.

스프링 프레임워크는 Java 언어를 기반으로 구축되었습니다. 따라서 스프링을 효과적으로 활용하고 그 내부 동작 원리를 깊이 이해하기 위해서는 Java 언어 자체에 대한 탄탄한 기초가 필수적입니다.1 단순히 문법을 아는 것을 넘어, Java의 객체 지향 패러다임, 메모리 관리 방식, 핵심 API 등에 대한 이해는 향후 스프링의 고급 기능을 학습하고 실제 문제 해결 능력을 키우는 데 결정적인 역할을 합니다.

본 학습 가이드는 이러한 배경을 바탕으로, 스프링 백엔드 개발에 필요한 핵심 Java 역량을 체계적으로 다지는 첫걸음을 안내합니다. 복잡한 웹 프레임워크에 바로 뛰어들기 전에, 간단한 콘솔 애플리케이션 제작을 통해 Java의 기본기를 확실하게 다지는 것을 목표로 합니다.

### **B. 학습 프로젝트 소개: 콘솔 기반 할 일 목록 관리 앱**

이 가이드에서는 "콘솔 기반 할 일 목록 관리" 애플리케이션을 단계별로 함께 만들어가면서 Java의 핵심 개념을 실습합니다. 이 프로젝트는 다음과 같은 기능을 포함합니다.

* 할 일 추가
* 할 일 목록 조회
* 할 일 완료 처리
* 마감일이 있는 할 일 관리 (상속 활용)
* (선택 사항) 파일에 할 일 목록 저장 및 불러오기

이 간단한 프로젝트를 통해 Java 개발 환경 설정부터 시작하여 기본 문법, 객체 지향 프로그래밍(클래스, 객체, 캡슐화, 상속, 다형성), 컬렉션 프레임워크(ArrayList), 스트림 API, 그리고 기본적인 파일 입출력까지 Java의 필수 요소를 자연스럽게 익힐 수 있습니다.3 각 단계는 이전 단계의 지식을 바탕으로 점진적으로 진행되며, '왜 이 기술을 배워야 하는지'에 대한 이해를 돕는 데 중점을 둡니다.

## **II. 1단계: Java 개발 환경 설정**

Java 개발을 시작하기 위해 가장 먼저 필요한 것은 Java Development Kit(JDK)를 설치하고 코드를 작성하고 실행할 통합 개발 환경(IDE)을 설정하는 것입니다.

### **A. Java Development Kit (JDK) 설치**

JDK는 Java 코드를 컴파일(javac)하고 실행(java)하는 데 필요한 도구들을 포함합니다.5 다양한 JDK 배포판이 존재하지만, 학습 및 개발 목적으로는 무료로 사용 가능하며 안정적인 OpenJDK 기반의 배포판을 권장합니다. 여기서는 **Azul Zulu OpenJDK 17 (LTS)** 버전을 기준으로 설명합니다. Java 17은 현재 많은 스프링 부트 프로젝트에서 권장하는 LTS(Long-Term Support) 버전입니다.5

**설치 단계:**

1. **다운로드:** [Azul Zulu 다운로드 페이지](https://www.azul.com/downloads/?package=jdk#zulu)에 접속하여 사용하는 운영체제(Windows, macOS, Linux)와 아키텍처(x64 등)에 맞는 Java 17(LTS) 버전의 JDK 설치 파일을 다운로드합니다. (예: Windows는.msi 또는.zip, macOS는.dmg, Linux는.deb,.rpm 또는.tar.gz).5
2. **설치:**
   * **Windows:** 다운로드한.msi 설치 파일을 실행하고 화면의 지시에 따라 설치를 완료합니다..zip 파일의 경우, 원하는 위치(예: C:\Program Files\Java\)에 압축을 해제합니다.5
   * **macOS:** 다운로드한.dmg 파일을 열고 설치 프로그램을 실행합니다.5
   * **Linux:** 배포판에 따라.deb 또는.rpm 파일을 사용하여 설치하거나,.tar.gz 파일을 원하는 위치(예: /usr/lib/jvm/)에 압축 해제합니다.5
3. **환경 변수 설정 (필수):** Java 컴파일러와 런타임을 시스템 어디서든 인식하고, 다른 개발 도구(Maven, Gradle 등)가 JDK 위치를 찾을 수 있도록 환경 변수를 설정해야 합니다.
   * **JAVA\_HOME 설정:** JDK가 설치된 최상위 디렉토리 경로를 JAVA\_HOME 환경 변수로 설정합니다. (예: Windows C:\Program Files\Java\zulu17..., Linux/macOS /usr/lib/jvm/zulu17...).2
   * **PATH 설정:** JAVA\_HOME 환경 변수를 이용하여 JDK의 bin 디렉토리를 시스템 PATH 환경 변수에 추가합니다. (예: %JAVA\_HOME%\bin 또는 $JAVA\_HOME/bin) 이렇게 하면 터미널이나 명령 프롬프트 어디서든 java, javac 명령어를 실행할 수 있습니다.2
   * **설정 방법:**
     + Windows: 시스템 속성 → 고급 시스템 설정 → 환경 변수.5
     + macOS/Linux: ~/.bash\_profile, ~/.zshrc 또는 /etc/environment 파일 편집 후 source 명령어로 적용.2
4. **설치 확인:** 터미널 또는 명령 프롬프트를 새로 열고 다음 명령어를 입력하여 설치 및 환경 변수 설정이 올바르게 되었는지 확인합니다.5  
   Bash  
   java -version  
   javac -version  
   echo %JAVA\_HOME% # Windows  
   echo $JAVA\_HOME # macOS/Linux  
   각 명령어 실행 시 설치한 JDK 버전 정보와 설정한 경로가 올바르게 출력되어야 합니다.

### **B. 통합 개발 환경 (IDE) 선택 및 설치**

IDE는 코드 작성, 컴파일, 디버깅, 프로젝트 관리 등 개발 전반의 생산성을 크게 향상시키는 도구입니다. Java 및 스프링 개발에는 **IntelliJ IDEA**가 강력한 기능과 편의성으로 널리 사용됩니다. 무료 버전인 **Community Edition**으로도 본 학습 가이드를 충분히 따라올 수 있습니다.8 Eclipse 역시 훌륭한 무료 IDE이며 선호에 따라 선택할 수 있습니다.11

**IntelliJ IDEA Community Edition 설치:**

1. ([https://www.jetbrains.com/idea/download/)에](https://www.jetbrains.com/idea/download/)에) 접속하여 Community 버전을 다운로드합니다.9
2. 다운로드한 설치 파일을 실행하여 설치를 진행합니다.

### **C. IntelliJ IDEA에서 "ConsoleToDoList" 프로젝트 생성**

IDE 설치 후, 우리의 학습 프로젝트인 "ConsoleToDoList"를 생성합니다.

1. IntelliJ IDEA를 실행하고 시작 화면에서 **New Project**를 클릭합니다.8
2. **New Project** 창에서 왼쪽 메뉴의 **Java**를 선택합니다.8
3. **Project SDK** (또는 JDK) 항목에서 앞에서 설치한 JDK 17 버전을 선택합니다. 목록에 없다면 "Add JDK..."를 클릭하여 JDK 설치 경로를 직접 지정합니다.8
4. **Name** 필드에 프로젝트 이름(예: ConsoleToDoList)을 입력합니다.8
5. **Location** 필드에서 프로젝트를 저장할 경로를 지정합니다.8
6. **Create** 버튼을 클릭하여 프로젝트 생성을 완료합니다.8
7. 프로젝트가 생성되면 왼쪽의 Project 창에 기본적인 폴더 구조가 나타납니다. Java 소스 코드는 주로 src 폴더 아래에 작성하게 됩니다.9
8. src 폴더를 우클릭하고 **New > Java Class**를 선택하여 메인 애플리케이션 클래스 파일(예: ToDoApp)을 생성합니다.8

이제 Java 코드를 작성하고 실행할 준비가 모두 완료되었습니다.

## **III. 2단계: Java 기초 문법과 콘솔 앱 뼈대 만들기**

프로젝트 환경 설정이 끝났으니, 이제 Java의 기본적인 문법을 배우면서 콘솔 할 일 목록 앱의 기본 구조를 만들어 보겠습니다.

### **A. main 메소드: 프로그램의 시작점**

모든 Java 애플리케이션은 main 메소드에서 실행을 시작합니다. ToDoApp.java 파일 안에 다음과 같이 main 메소드를 작성합니다.1

Java

// ToDoApp.java  
public class ToDoApp {  
 public static void main(String args) {  
 System.out.println("할 일 목록 관리 앱 시작!");  
 // 여기에 프로그램 로직이 들어갑니다.  
 }  
}

* public static void main(String args): 이 정해진 형식은 Java Virtual Machine(JVM)이 프로그램을 시작할 때 호출하는 진입점입니다.1
* System.out.println(...): 괄호 안의 내용을 콘솔에 출력하고 줄을 바꿉니다.1 IntelliJ IDEA에서 녹색 실행 버튼(클래스나 메소드 옆)을 누르거나 Shift+F10 (기본 단축키)을 눌러 코드를 실행하고 콘솔 출력을 확인할 수 있습니다.8

### **B. 변수와 데이터 타입: 데이터 저장하기**

프로그램에서 데이터를 다루려면 변수를 사용해야 합니다. 변수는 데이터를 저장하는 메모리 공간에 이름을 붙인 것입니다.14 Java는 **정적 타입(Static Typing)** 언어로, 변수를 선언할 때 반드시 데이터 타입을 명시해야 합니다.

* **기본형 타입 (Primitive Types):** 숫자, 문자, 논리값 등 기본적인 데이터를 직접 저장합니다.
  + int: 정수 (예: int choice = 1;)
  + double: 실수 (예: double average = 95.5;)
  + boolean: 논리값 (true 또는 false) (예: boolean isRunning = true;)
  + char: 단일 문자 (예: char grade = 'A';)
* **참조형 타입 (Reference Types):** 객체를 가리키는 주소값을 저장합니다.
  + String: 문자열 (예: String message = "안녕하세요";)
  + 배열, 클래스(우리가 만들 Task 등), 인터페이스 등

JavaScript와 같은 동적 타입 언어와 달리, Java는 변수 선언 시 지정된 타입과 다른 타입의 값을 할당할 수 없습니다.1 이는 컴파일 시점에 오류를 발견하여 프로그램의 안정성을 높이는 데 도움을 줍니다.

### **C. 제어 흐름: 프로그램 로직 만들기**

프로그램의 실행 순서를 제어하기 위해 조건문과 반복문을 사용합니다.15

* **if-else 문:** 조건에 따라 다른 코드를 실행합니다.  
  Java  
  int age = 20;  
  if (age >= 19) {  
   System.out.println("성인입니다.");  
  } else {  
   System.out.println("미성년자입니다.");  
  }
* **while 문:** 조건이 참인 동안 코드를 반복 실행합니다. 앱의 메인 메뉴처럼 사용자가 종료를 원할 때까지 계속 실행해야 할 때 유용합니다.15  
  Java  
  boolean isRunning = true;  
  while (isRunning) {  
   // 메뉴 표시 및 사용자 입력 처리  
   // 사용자가 '종료' 선택 시 isRunning = false;  
  }
* **for 문:** 정해진 횟수만큼 또는 컬렉션의 모든 요소를 순회하며 코드를 반복 실행합니다. 나중에 할 일 목록을 출력할 때 사용됩니다.15
* **switch 문:** 특정 변수의 값에 따라 여러 경우 중 하나를 실행합니다. 메뉴 선택 처리에 if-else 대신 사용할 수 있습니다.

### **D. 사용자 입력 처리: Scanner**

콘솔로부터 사용자 입력을 받기 위해 java.util.Scanner 클래스를 사용합니다.15

1. **import 문 추가:** 파일 상단에 import java.util.Scanner;를 추가하여 Scanner 클래스를 사용할 수 있도록 합니다.
2. **Scanner 객체 생성:** main 메소드 내에서 Scanner scanner = new Scanner(System.in); 코드로 Scanner 객체를 생성합니다. System.in은 표준 입력(키보드)을 의미합니다.
3. **입력 받기:**
   * scanner.nextInt(): 정수 입력을 받습니다. 메뉴 선택 등에 사용됩니다.
   * scanner.nextLine(): 한 줄의 문자열 전체를 입력받습니다. 할 일 설명 입력 등에 사용됩니다.
4. **Scanner 닫기:** 프로그램 종료 전에 scanner.close();를 호출하여 사용한 자원을 해제하는 것이 좋습니다.15

**주의:** nextInt() 다음에 nextLine()을 바로 사용할 때 문제가 발생할 수 있습니다. nextInt()는 숫자만 읽고 입력 버퍼에 남아있는 줄바꿈 문자(\n)를 소비하지 않습니다. 이후 nextLine()이 이 남아있는 줄바꿈 문자를 읽어버려 입력을 건너뛰는 것처럼 보일 수 있습니다. 이를 해결하기 위해 nextInt() 호출 후 scanner.nextLine();을 한 번 더 호출하여 버퍼를 비워주는 것이 일반적인 방법입니다.15

### **E. 프로젝트 1단계: 기본 메뉴 루프 구현**

이제 배운 내용을 바탕으로 ToDoApp.java의 main 메소드 안에 기본적인 메뉴 루프를 구현합니다.

Java

// ToDoApp.java  
import java.util.Scanner;  
  
public class ToDoApp {  
 public static void main(String args) {  
 Scanner scanner = new Scanner(System.in);  
 boolean isRunning = true;  
  
 System.out.println("=== 콘솔 할 일 목록 관리 앱 ===");  
  
 while (isRunning) {  
 System.out.println("\n1. 할 일 추가");  
 System.out.println("2. 할 일 목록 보기");  
 System.out.println("3. 종료");  
 System.out.print("메뉴를 선택하세요: ");  
  
 int choice = scanner.nextInt();  
 scanner.nextLine(); // nextInt() 후의 줄바꿈 문자 소비  
  
 switch (choice) {  
 case 1:  
 System.out.println("[할 일 추가 - 구현 예정]");  
 // 할 일 추가 로직  
 break;  
 case 2:  
 System.out.println("[할 일 목록 보기 - 구현 예정]");  
 // 할 일 목록 보기 로직  
 break;  
 case 3:  
 System.out.println("앱을 종료합니다.");  
 isRunning = false; // 반복문 종료 조건 변경  
 break;  
 default:  
 System.out.println("잘못된 입력입니다. 다시 선택해주세요.");  
 break;  
 }  
 }  
 scanner.close(); // Scanner 자원 해제  
 }  
}

이 코드를 실행하면 콘솔에 메뉴가 표시되고, 사용자가 숫자를 입력하면 해당 메뉴에 대한 (아직 구현되지 않은) 메시지가 출력됩니다. 3번을 입력하면 앱이 종료됩니다.

## **IV. 3단계: 객체 지향 프로그래밍 - Task 클래스 정의**

이제 할 일(Task)이라는 개념을 Java의 객체로 모델링해 보겠습니다. 이를 통해 객체 지향 프로그래밍(OOP)의 핵심 개념인 클래스, 객체, 캡슐화를 이해합니다.

### **A. 클래스와 객체: 현실 세계 모델링**

OOP는 현실 세계의 사물이나 개념(예: 할 일)을 속성(데이터)과 행동(메소드)을 가진 객체(Object)로 표현하는 방식입니다. \*\*클래스(Class)\*\*는 이러한 객체를 만들기 위한 설계도 또는 템플릿입니다.16

### **B. Task 클래스 설계 및 필드 정의**

src 폴더에 Task.java 파일을 새로 만듭니다. Task 클래스는 할 일의 '설명'과 '완료 여부'라는 상태(데이터)를 가집니다. 이를 클래스 내부에 **필드(field)** 또는 \*\*인스턴스 변수(instance variable)\*\*로 선언합니다.

Java

// Task.java  
public class Task {  
 private String description; // 할 일 설명  
 private boolean completed; // 완료 여부  
}

### **C. 캡슐화: 데이터 보호와 접근 제어**

\*\*캡슐화(Encapsulation)\*\*는 객체의 데이터를 외부에서 직접 접근하지 못하도록 보호하고, 대신 객체가 제공하는 메소드를 통해서만 데이터에 접근하도록 하는 OOP의 중요한 원칙입니다.16 이는 데이터의 무결성을 지키고 객체의 내부 구현을 숨기는 데 목적이 있습니다.

* **private 접근 제한자:** 필드 앞에 private을 붙이면 해당 필드는 클래스 외부에서 직접 접근할 수 없게 됩니다.16 예를 들어 ToDoApp 클래스에서 task.description = "새 설명";과 같이 직접 수정할 수 없습니다. 이는 의도치 않은 값 변경(예: description을 null로 설정)을 방지합니다.16
* **public Getter와 Setter 메소드:** 외부에서 private 필드에 안전하게 접근할 수 있도록 public 접근 제한자를 가진 메소드를 제공합니다.16
  + **Getter:** 필드 값을 읽어서 반환하는 메소드. 일반적으로 get필드명() 형식을 따릅니다. boolean 타입 필드의 경우 is필드명() 형식을 관례적으로 사용합니다.16  
    Java  
    public String getDescription() {  
     return description;  
    }  
      
    public boolean isCompleted() { // boolean 타입은 is...  
     return completed;  
    }
  + **Setter:** 외부로부터 값을 받아 필드 값을 변경하는 메소드. 일반적으로 set필드명(타입 매개변수) 형식을 따릅니다.16 Setter 메소드 내부에 유효성 검사 로직을 추가하여 잘못된 값이 필드에 저장되는 것을 막을 수 있습니다. 이는 캡슐화의 중요한 장점입니다.16  
    Java  
    public void setDescription(String description) {  
     // 간단한 유효성 검사 예시  
     if (description!= null &&!description.trim().isEmpty()) {  
     this.description = description;  
     } else {  
     System.out.println("오류: 할 일 설명은 비워둘 수 없습니다.");  
     }  
    }  
      
    public void setCompleted(boolean completed) {  
     this.completed = completed;  
    }
  + **this 키워드:** 메소드 내에서 인스턴스 변수(필드)와 매개변수의 이름이 같을 때, this.필드명은 인스턴스 변수를 명확히 가리킵니다.16
  + IntelliJ IDEA에서는 코드 편집기에서 Alt+Insert (또는 Cmd+N on Mac) 단축키를 누르고 "Getter and Setter"를 선택하여 자동으로 생성할 수 있습니다.16

### **D. 생성자: 객체 초기화**

\*\*생성자(Constructor)\*\*는 new 키워드로 객체를 생성할 때 호출되어 객체의 초기 상태를 설정하는 특별한 메소드입니다. 클래스 이름과 동일하며 리턴 타입이 없습니다.16

Java

// Task.java 내부에 추가  
public Task(String description) {  
 // this.description = description; // 직접 필드에 할당하거나  
 this.setDescription(description); // Setter를 통해 유효성 검사 포함하여 초기화  
 this.completed = false; // 새로운 할 일은 기본적으로 미완료 상태  
}

위 생성자는 할 일 설명을 매개변수로 받아 description 필드를 초기화하고, completed 필드는 false로 설정합니다. Setter 메소드를 사용하여 초기화하면 생성 시점에도 유효성 검사를 적용할 수 있습니다.

### **E. toString() 메소드: 객체 정보 출력**

객체를 System.out.println() 등으로 직접 출력하면 기본적으로는 클래스명과 메모리 주소 정보가 나옵니다. 객체의 내용을 알아보기 쉽게 출력하려면 Object 클래스로부터 상속받은 toString() 메소드를 \*\*오버라이드(Override)\*\*해야 합니다.

Java

// Task.java 내부에 추가  
@Override  
public String toString() {  
 return "할 일: " + description + " (완료: " + (completed? "Y" : "N") + ")";  
 // 삼항 연산자: completed가 true이면 "Y", false이면 "N"을 반환  
}

@Override 어노테이션은 이 메소드가 부모 클래스의 메소드를 재정의한다는 것을 명시적으로 나타냅니다.

### **F. 프로젝트 2단계: Task 객체 사용하기**

이제 ToDoApp.java의 "할 일 추가" 로직에서 Task 클래스를 사용해 봅시다.

Java

// ToDoApp.java의 switch 문 case 1 내부 수정  
case 1:  
 System.out.print("추가할 할 일 내용을 입력하세요: ");  
 String description = scanner.nextLine();  
 Task newTask = new Task(description); // Task 객체 생성  
 System.out.println("새 할 일이 생성되었습니다: " + newTask); // toString() 호출됨  
 // TODO: 생성된 newTask를 목록에 추가하는 로직 (다음 단계에서 구현)  
 break;

사용자로부터 설명을 입력받아 Task 객체를 생성하고, 생성된 객체를 출력하여 toString() 메소드가 잘 동작하는지 확인합니다.

## **V. 4단계: 컬렉션 프레임워크 - ArrayList로 할 일 목록 관리**

하나의 할 일을 객체로 만드는 방법을 배웠으니, 이제 여러 개의 할 일을 효율적으로 관리하는 방법을 알아봅니다. Java에서는 \*\*컬렉션 프레임워크(Collections Framework)\*\*를 통해 다양한 종류의 데이터 그룹을 다룰 수 있는 클래스들을 제공합니다.

### **A. Java 컬렉션 프레임워크 소개**

컬렉션 프레임워크는 데이터를 저장하고 관리하는 표준화된 방법을 제공합니다. 주요 인터페이스는 다음과 같습니다.18

* **List 인터페이스:** 순서가 있는 데이터의 집합을 다룹니다. 데이터 중복을 허용합니다. 인덱스를 통해 특정 위치의 데이터에 접근할 수 있습니다. 대표적인 구현 클래스로 ArrayList, LinkedList가 있습니다.18
* **Set 인터페이스:** 순서가 중요하지 않고, 데이터의 중복을 허용하지 않는 집합을 다룹니다. 대표적인 구현 클래스로 HashSet, TreeSet이 있습니다.18
* **Map 인터페이스:** 키(Key)와 값(Value)의 쌍으로 데이터를 저장합니다. 키는 중복될 수 없지만 값은 중복될 수 있습니다. 특정 키를 통해 값을 빠르게 찾아야 할 때 유용합니다. 대표적인 구현 클래스로 HashMap, TreeMap이 있습니다.18

**백엔드 개발에서의 중요성:** 어떤 컬렉션을 선택하느냐는 애플리케이션의 성능과 로직 구현 방식에 큰 영향을 미칩니다. 예를 들어, 사용자의 고유 ID를 관리할 때는 Set이 적합하고, 특정 ID로 사용자 정보를 빠르게 조회해야 할 때는 Map이 효율적입니다. 할 일 목록처럼 순서대로 저장하고 조회하는 경우에는 List가 적합합니다.

### **B. ArrayList 사용하기**

ArrayList는 내부적으로 배열을 사용하여 데이터를 관리하는 List 인터페이스의 구현체입니다. 크기가 동적으로 조절되며, 인덱스를 통한 데이터 접근 속도가 빠릅니다.18

1. **ArrayList 선언 및 생성:** ToDoApp.java 클래스의 main 메소드 바깥 (클래스 멤버 변수 위치) 또는 main 메소드 시작 부분에 Task 객체를 저장할 ArrayList를 선언하고 생성합니다.  
   Java  
   // ToDoApp.java 상단 또는 main 메소드 시작 부분  
   import java.util.ArrayList;  
   import java.util.List; // List 인터페이스 import  
     
   //... 클래스 정의...  
   private static List<Task> tasks = new ArrayList<>(); // Task 객체들을 저장할 리스트  
   * **List<Task>:** List 인터페이스 타입으로 변수를 선언하는 것이 좋습니다. 이렇게 하면 나중에 필요에 따라 ArrayList가 아닌 다른 List 구현체(예: LinkedList)로 쉽게 변경할 수 있습니다 (다형성).
   * **new ArrayList<>():** ArrayList 객체를 생성합니다. <> 안의 타입(Task)은 \*\*제네릭(Generics)\*\*이라고 하며, 이 리스트가 Task 타입의 객체만 저장하도록 강제하여 타입 안정성을 높입니다.22
   * static 키워드는 main 메소드(static 메소드) 내에서 직접 접근하기 위해 사용되었습니다. (객체 생성 없이 사용 가능)
2. **ArrayList 주요 메소드:**
   * add(element): 리스트 끝에 요소를 추가합니다. 22  
     Java  
     // case 1: 할 일 추가 로직에서  
     Task newTask = new Task(description);  
     tasks.add(newTask); // 생성된 Task 객체를 리스트에 추가  
     System.out.println("할 일이 추가되었습니다.");
   * get(index): 지정된 인덱스의 요소를 반환합니다. 인덱스는 0부터 시작합니다. 21
   * size(): 리스트에 저장된 요소의 개수를 반환합니다. 22
   * remove(index): 지정된 인덱스의 요소를 삭제합니다. 22
   * remove(object): 지정된 객체와 동일한 첫 번째 요소를 삭제합니다. 객체 비교를 위해 Task 클래스에 equals() 메소드가 오버라이드되어 있어야 정확하게 동작합니다.23
   * isEmpty(): 리스트가 비어있는지 확인합니다.
   * clear(): 리스트의 모든 요소를 삭제합니다.23

### **C. 할 일 목록 순회 및 출력**

ArrayList에 저장된 모든 할 일을 화면에 출력하기 위해 반복문을 사용합니다.

* **향상된 for문 (for-each loop):** 리스트의 각 요소를 순서대로 접근하는 가장 간결한 방법입니다.21  
  Java  
  // case 2: 할 일 목록 보기 로직에서  
  System.out.println("\n--- 할 일 목록 ---");  
  if (tasks.isEmpty()) {  
   System.out.println("등록된 할 일이 없습니다.");  
  } else {  
   for (int i = 0; i < tasks.size(); i++) {  
   // (i + 1) 로 사용자에게 1부터 시작하는 번호 표시  
   System.out.println((i + 1) + ". " + tasks.get(i)); // get(i)로 Task 객체 접근  
   }  
  }  
  System.out.println("----------------");  
  위 코드는 get(index) 메소드를 사용하여 각 할 일에 1부터 시작하는 번호를 붙여 출력합니다. tasks.get(i)는 Task 객체를 반환하며, System.out.println은 이 객체의 toString() 메소드를 호출하여 내용을 출력합니다.

### **D. List, Set, Map 비교 요약**

다음 표는 Java 컬렉션의 주요 인터페이스 특징을 요약한 것입니다.19

| **특징** | **List (예: ArrayList)** | **Set (예: HashSet)** | **Map (예: HashMap)** |
| --- | --- | --- | --- |
| 순서 유지 | O (삽입 순서) | X (일반적) | X |
| 중복 허용 | O | X | Key: X / Value: O |
| 접근 방법 | 인덱스 | 반복자/포함 여부 | 키 |
| 주요 구현체 | ArrayList, LinkedList | HashSet, TreeSet | HashMap, TreeMap |
| 주요 사용 사례 | 순서 있는 목록 | 고유 요소 집합 | 키-값 쌍 관리 |

### **E. 프로젝트 3단계: 핵심 목록 관리 기능 구현**

이제 ArrayList를 사용하여 할 일 목록 관리 앱의 핵심 기능을 완성합니다.

1. **할 일 추가:** case 1에서 Task 객체를 생성한 후 tasks.add(newTask); 코드를 추가하여 리스트에 저장합니다.
2. **할 일 목록 보기:** case 2에 위에서 설명한 for 루프를 사용하여 tasks 리스트의 내용을 번호와 함께 출력하는 코드를 구현합니다. 리스트가 비어있을 경우 "등록된 할 일이 없습니다." 메시지를 출력합니다.
3. **(선택) 할 일 삭제:** 새로운 메뉴 옵션(예: 4번)을 추가하고, 사용자에게 삭제할 할 일 번호를 입력받아 tasks.remove(index - 1); 코드로 해당 할 일을 삭제하는 기능을 구현할 수 있습니다. 인덱스 유효성 검사(입력된 번호가 목록 범위 내에 있는지)를 추가하는 것이 좋습니다.

## **VI. 5단계: 스트림 API - 함수형 스타일로 데이터 처리**

Java 8부터 도입된 스트림 API(Stream API)는 컬렉션 데이터를 더 간결하고 효율적으로 처리할 수 있는 함수형 프로그래밍 스타일의 방법을 제공합니다. 반복문을 명시적으로 사용하지 않고 데이터 처리 과정을 선언적으로 표현할 수 있습니다.24

### **A. 스트림 API 소개**

스트림은 데이터 요소의 흐름(sequence)을 나타냅니다. 컬렉션(List, Set 등)이나 배열로부터 스트림을 생성한 후, 여러 \*\*중간 연산(Intermediate Operation)\*\*을 연결하여 데이터를 가공하고, 마지막으로 \*\*최종 연산(Terminal Operation)\*\*을 통해 결과를 생성합니다.24

* **특징:**
  + 원본 데이터를 변경하지 않습니다 (Non-mutating).24
  + 일회성입니다 (재사용 불가).24
  + 내부 반복(Internal Iteration)을 사용합니다 (개발자가 반복 로직을 직접 작성하지 않음).

### **B. filter(): 조건에 맞는 데이터 걸러내기**

filter() 중간 연산은 주어진 조건(Predicate, 람다식으로 표현)을 만족하는 요소만 포함하는 새로운 스트림을 반환합니다.24

* **람다 표현식 (Lambda Expression):** (매개변수) -> { 실행 코드 } 형태로, 익명 함수를 간결하게 표현하는 방법입니다.
* **예시:** 완료되지 않은(completed가 false인) 할 일만 필터링  
  Java  
  tasks.stream() // List<Task>로부터 스트림 생성  
   .filter(task ->!task.isCompleted()) // task 객체를 받아 completed가 false이면 통과  
   //... 이후 다른 연산...

### **C. map(): 데이터 변환하기**

map() 중간 연산은 스트림의 각 요소에 주어진 함수(Function, 람다식 또는 메소드 참조로 표현)를 적용하여 새로운 형태의 요소로 변환된 스트림을 반환합니다.24

* **메소드 참조 (Method Reference):** 람다 표현식을 더욱 간결하게 표현하는 방법입니다. 클래스명::메소드명 또는 객체참조::메소드명 형태로 사용됩니다.
* **예시:** 각 Task 객체에서 할 일 설명(description)만 추출  
  Java  
  tasks.stream()  
   .map(task -> task.getDescription()) // 람다식 사용  
   // 또는  
   .map(Task::getDescription) // 메소드 참조 사용 (더 간결함)  
   //... 이후 다른 연산...

### **D. collect(): 결과 수집하기**

collect() 최종 연산은 스트림 파이프라인을 통해 처리된 요소들을 모아서 원하는 형태(예: List, Set, Map 또는 단일 값)로 반환합니다.24 java.util.stream.Collectors 클래스가 다양한 컬렉터(Collector)를 제공합니다.

* **예시:** 완료되지 않은 할 일의 설명만 모아서 새로운 List<String>으로 만들기  
  Java  
  List<String> incompleteDescriptions = tasks.stream() // 스트림 생성  
   .filter(task ->!task.isCompleted()) // 미완료 필터링  
   .map(Task::getDescription) // 설명 추출  
   .collect(Collectors.toList()); // 결과를 List로 수집

### **E. 주요 스트림 연산 요약**

다음 표는 이 가이드에서 다루는 기본적인 스트림 연산을 요약한 것입니다.24

| **연산** | **종류** | **목적** | **예시 (람다/메소드 참조)** |
| --- | --- | --- | --- |
| stream() | 소스 생성 | 컬렉션으로부터 스트림 생성 | tasks.stream() |
| filter() | 중간 연산 | 조건에 맞는 요소 선택 | t ->!t.isCompleted() |
| map() | 중간 연산 | 각 요소를 다른 형태로 변환 | Task::getDescription |
| collect() | 최종 연산 | 결과를 컬렉션 등으로 수집 | Collectors.toList() |
| forEach() | 최종 연산 | 각 요소에 대해 특정 작업 수행 | System.out::println |

### **F. 프로젝트 4단계: 스트림 활용 기능 추가**

이제 스트림 API를 사용하여 할 일 목록 앱에 새로운 기능을 추가하고 기존 기능을 개선합니다.

1. **미완료 할 일 보기:** 새로운 메뉴 옵션(예: 5번)을 추가하고, filter와 collect를 사용하여 완료되지 않은 할 일 목록만 출력하는 기능을 구현합니다.  
   Java  
   // case 5: 미완료 할 일 보기 로직  
   List<Task> incompleteTasks = tasks.stream()  
    .filter(task ->!task.isCompleted())  
    .collect(Collectors.toList());  
   System.out.println("\n--- 미완료 할 일 목록 ---");  
   if (incompleteTasks.isEmpty()) {  
    System.out.println("모든 할 일을 완료했습니다!");  
   } else {  
    for (int i = 0; i < incompleteTasks.size(); i++) {  
    System.out.println((i + 1) + ". " + incompleteTasks.get(i));  
    }  
   }  
   System.out.println("-----------------------");  
   break;
2. **할 일 완료 처리:** 새로운 메뉴 옵션(예: 6번)을 추가합니다.
   * 먼저 "할 일 목록 보기"(case 2)를 실행하여 사용자에게 번호를 보여줍니다.
   * 사용자에게 완료 처리할 할 일 번호를 입력받습니다.
   * 입력받은 번호(taskNumber)가 유효한 범위 내에 있는지 확인합니다.
   * 유효하다면, 해당 Task 객체를 가져와서 setCompleted(true) 메소드를 호출합니다.  
     Java  
     // case 6: 할 일 완료 처리 로직  
     System.out.print("완료 처리할 할 일 번호를 입력하세요: ");  
     int taskNumberToComplete = scanner.nextInt();  
     scanner.nextLine(); // 줄바꿈 문자 소비  
       
     if (taskNumberToComplete > 0 && taskNumberToComplete <= tasks.size()) {  
      Task taskToComplete = tasks.get(taskNumberToComplete - 1); // 인덱스는 0부터 시작  
      taskToComplete.setCompleted(true);  
      System.out.println("'" + taskToComplete.getDescription() + "' 할 일을 완료 처리했습니다.");  
     } else {  
      System.out.println("잘못된 번호입니다.");  
     }  
     break;

## **VII. 6단계: 상속과 다형성 - 기능 확장하기**

OOP의 또 다른 강력한 특징인 상속과 다형성을 이용하여 할 일 목록 앱에 '마감일'이 있는 특별한 종류의 할 일을 추가해 보겠습니다.

### **A. 상속: 코드 재사용과 확장**

\*\*상속(Inheritance)\*\*은 기존 클래스(부모 클래스, Superclass)의 속성과 메소드를 물려받아 새로운 클래스(자식 클래스, Subclass)를 만드는 기능입니다. 이를 통해 코드 재사용성을 높이고 클래스 간의 계층 구조를 형성할 수 있습니다.27

1. **DeadlineTask 클래스 생성:** Task 클래스를 상속받는 DeadlineTask.java 파일을 새로 만듭니다. extends 키워드를 사용합니다.  
   Java  
   // DeadlineTask.java  
   public class DeadlineTask extends Task {  
    private String deadline; // 마감일 필드 추가  
   }
2. **자식 클래스 생성자:** 자식 클래스의 생성자에서는 반드시 부모 클래스의 생성자를 호출하여 부모 클래스의 필드를 초기화해야 합니다. super() 키워드를 사용합니다.27  
   Java  
   // DeadlineTask.java 내부에 추가  
   public DeadlineTask(String description, String deadline) {  
    super(description); // 부모(Task) 클래스의 생성자 호출하여 description 초기화  
    this.deadline = deadline;  
   }
3. **새로운 필드 Getter/Setter 추가:** deadline 필드에 대한 Getter와 Setter를 추가합니다.

### **B. 메소드 오버라이딩: 동작 방식 변경**

\*\*메소드 오버라이딩(Method Overriding)\*\*은 자식 클래스에서 부모 클래스로부터 상속받은 메소드를 동일한 이름과 매개변수로 재정의하여 동작 방식을 변경하는 것입니다.27

DeadlineTask 객체를 출력할 때 마감일 정보도 함께 보이도록 toString() 메소드를 오버라이드합니다.

Java

// DeadlineTask.java 내부에 추가  
@Override  
public String toString() {  
 // 부모 클래스(Task)의 toString() 결과를 활용하고 마감일 정보 추가  
 return super.toString() + ", 마감일: " + deadline;  
}

super.toString()은 부모 클래스인 Task의 toString() 메소드를 호출하여 기존 출력 형식("할 일:..., 완료:...")을 가져옵니다.

### **C. 다형성: 다양한 형태를 다루는 능력**

\*\*다형성(Polymorphism)\*\*은 '여러 형태를 가질 수 있는 능력'을 의미합니다. 상속 관계에서 자식 클래스 객체는 부모 클래스 타입으로 다루어질 수 있습니다.27 즉, DeadlineTask 객체는 Task 타입의 변수에 할당될 수 있습니다.

Java

Task myTask = new DeadlineTask("보고서 작성", "2024-12-31");

이것이 가능한 이유는 DeadlineTask가 Task를 상속했기 때문에 "DeadlineTask **is a** Task" 관계가 성립하기 때문입니다.29

우리가 Task 객체를 저장하기 위해 만든 List<Task> tasks에는 Task 객체뿐만 아니라 DeadlineTask 객체도 함께 저장할 수 있습니다.

Java

tasks.add(new Task("일반 회의 준비"));  
tasks.add(new DeadlineTask("기획안 제출", "2024-11-30"));

리스트를 순회하며 각 요소의 toString() 메소드를 호출하면, JVM은 실제 객체의 타입(컴파일 시점의 변수 타입이 아닌, 런타임 시점의 실제 객체 타입)에 맞는 오버라이드된 toString() 메소드를 찾아 실행합니다 (이를 **동적 메소드 디스패치**라고 합니다).27 따라서 Task 객체는 Task의 toString()이, DeadlineTask 객체는 DeadlineTask의 toString()이 호출되어 각기 다른 형식으로 출력됩니다.

**스프링에서의 중요성:** 스프링 프레임워크는 인터페이스 기반 설계를 적극 활용합니다. 다형성 덕분에 개발자는 인터페이스 타입으로 의존성을 주입받고, 스프링 컨테이너는 실제 어떤 구현 클래스의 객체를 주입할지 유연하게 결정할 수 있습니다. 이는 코드의 유연성과 확장성을 크게 높여줍니다.30

### **D. 프로젝트 5단계: 마감일 있는 할 일 추가 기능 구현**

1. **메뉴 수정:** ToDoApp.java의 case 1 (할 일 추가) 로직을 수정하여 사용자에게 일반 할 일인지 마감일 있는 할 일인지 묻습니다.
2. **입력 분기:** 사용자의 선택에 따라 필요한 정보(설명 또는 설명+마감일)를 추가로 입력받습니다.
3. **객체 생성 및 추가:** 입력받은 정보를 바탕으로 Task 또는 DeadlineTask 객체를 생성하고, 이를 tasks 리스트에 추가합니다.

Java

// case 1: 할 일 추가 로직 수정 예시  
System.out.println("할 일 종류를 선택하세요 (1: 일반, 2: 마감일): ");  
int taskType = scanner.nextInt();  
scanner.nextLine(); // 줄바꿈 문자 소비  
  
System.out.print("할 일 내용을 입력하세요: ");  
String description = scanner.nextLine();  
  
if (taskType == 1) {  
 tasks.add(new Task(description));  
 System.out.println("일반 할 일이 추가되었습니다.");  
} else if (taskType == 2) {  
 System.out.print("마감일을 입력하세요 (YYYY-MM-DD): ");  
 String deadline = scanner.nextLine();  
 tasks.add(new DeadlineTask(description, deadline));  
 System.out.println("마감일 있는 할 일이 추가되었습니다.");  
} else {  
 System.out.println("잘못된 할 일 종류입니다.");  
}  
break;

이제 "할 일 목록 보기"(case 2)를 실행하면, Task와 DeadlineTask 객체가 각자의 toString() 형식에 맞게 출력되는 것을 확인할 수 있습니다.

## **VIII. (선택) 7단계: 파일 입출력 - 할 일 목록 저장 및 불러오기**

콘솔 앱은 종료하면 메모리에 있던 데이터가 사라집니다. 할 일 목록을 파일에 저장하고 앱 시작 시 다시 불러와서 유지하는 기능을 구현해 봅시다. Java의 기본 I/O(Input/Output) 스트림을 사용합니다.

### **A. Java 기본 I/O 스트림 소개**

파일의 내용을 읽거나 쓰기 위해 스트림(Stream)을 사용합니다. 문자 기반의 텍스트 파일을 다룰 때는 Reader와 Writer 계열의 클래스를 사용합니다.32 성능 향상을 위해 버퍼(Buffer)를 사용하는 BufferedReader와 BufferedWriter를 함께 사용하는 것이 일반적입니다.32

### **B. 파일에 할 일 목록 쓰기 (BufferedWriter)**

tasks 리스트의 내용을 텍스트 파일(예: todolist.txt)에 저장합니다. 각 할 일의 정보를 특정 구분자(예: 쉼표 ,)로 구분하여 한 줄에 하나씩 저장하는 간단한 CSV(Comma-Separated Values) 형식을 사용해 봅시다.

Java

// ToDoApp.java에 추가할 메소드 예시  
import java.io.\*; // 필요한 I/O 클래스 import  
  
private static void saveTasksToFile(String filename) {  
 // try-with-resources: 블록 종료 시 자동으로 writer.close() 호출  
 try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {  
 for (Task task : tasks) {  
 String taskType = (task instanceof DeadlineTask)? "D" : "T"; // 타입 구분자  
 String completedStatus = task.isCompleted()? "true" : "false";  
 String line = taskType + "," + task.getDescription() + "," + completedStatus;  
  
 if (task instanceof DeadlineTask) {  
 line += "," + ((DeadlineTask) task).getDeadline(); // 마감일 추가  
 }  
 writer.write(line); // 파일에 한 줄 쓰기  
 writer.newLine(); // 줄바꿈  
 }  
 System.out.println("할 일 목록을 '" + filename + "' 파일에 저장했습니다.");  
 } catch (IOException e) {  
 System.err.println("파일 저장 중 오류 발생: " + e.getMessage());  
 }  
}

* **try-with-resources:** try 괄호 안에 생성된 리소스(BufferedWriter)는 try 블록이 끝나면 자동으로 close() 메소드가 호출됩니다. 이는 리소스 누수를 방지하는 가장 안전하고 권장되는 방법입니다.32
* **instanceof 연산자:** 객체가 특정 클래스의 인스턴스인지 확인합니다. 이를 이용해 Task와 DeadlineTask를 구분하여 저장합니다.
* **형 변환 (Casting):** (DeadlineTask) task는 task 변수가 실제로는 DeadlineTask 객체임을 명시하여 getDeadline() 메소드를 호출할 수 있게 합니다.
* **IOException 처리:** 파일 입출력 작업 중에는 오류가 발생할 수 있으므로 try-catch 블록으로 예외를 처리해야 합니다.

### **C. 파일에서 할 일 목록 읽기 (BufferedReader)**

앱 시작 시 todolist.txt 파일이 존재하면 내용을 읽어 tasks 리스트를 복원합니다.

Java

// ToDoApp.java에 추가할 메소드 예시  
private static void loadTasksFromFile(String filename) {  
 File file = new File(filename);  
 if (!file.exists()) {  
 System.out.println("저장된 할 일 파일이 없습니다.");  
 return;  
 }  
  
 // try-with-resources  
 try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {  
 String line;  
 tasks.clear(); // 기존 목록 초기화 후 불러오기  
 while ((line = reader.readLine())!= null) { // 파일 끝까지 한 줄씩 읽기  
 String parts = line.split(","); // 쉼표로 데이터 분리  
 if (parts.length >= 3) {  
 String taskType = parts;  
 String description = parts[1];  
 boolean completed = Boolean.parseBoolean(parts[2]);  
  
 Task task;  
 if ("D".equals(taskType) && parts.length >= 4) { // DeadlineTask  
 String deadline = parts[3];  
 task = new DeadlineTask(description, deadline);  
 } else { // 일반 Task  
 task = new Task(description);  
 }  
 task.setCompleted(completed); // 완료 상태 설정  
 tasks.add(task);  
 }  
 }  
 System.out.println("'" + filename + "' 파일에서 할 일 목록을 불러왔습니다.");  
 } catch (IOException e) {  
 System.err.println("파일 로딩 중 오류 발생: " + e.getMessage());  
 }  
}

* **readLine():** 파일에서 한 줄을 읽어 문자열로 반환합니다. 파일의 끝에 도달하면 null을 반환합니다.32
* **line.split(","):** 쉼표(,)를 기준으로 문자열을 분리하여 문자열 배열(String)로 만듭니다.
* **Boolean.parseBoolean():** 문자열("true" 또는 "false")을 boolean 값으로 변환합니다.
* 파일을 읽어 객체를 복원할 때도 타입 구분자(parts)를 확인하여 적절한 클래스(Task 또는 DeadlineTask)의 객체를 생성합니다.

### **D. 프로젝트 6단계 (선택): 저장/불러오기 기능 연동**

1. **메뉴 추가:** "저장", "불러오기" 메뉴 옵션을 추가합니다.
2. **메소드 호출:** 각 메뉴 선택 시 위에서 만든 saveTasksToFile("todolist.txt") 또는 loadTasksFromFile("todolist.txt") 메소드를 호출하도록 구현합니다.
3. **자동 로딩:** main 메소드 시작 부분에서 loadTasksFromFile("todolist.txt")를 호출하여 앱이 시작될 때 자동으로 저장된 데이터를 불러오도록 합니다.

## **IX. 8단계: 최종 코드 및 다음 단계**

지금까지 Java의 핵심 개념들을 배우면서 콘솔 기반 할 일 목록 관리 애플리케이션을 단계별로 완성했습니다.

### **A. 최종 코드 예시**

다음은 ToDoApp.java, Task.java, DeadlineTask.java의 완성된 코드 예시입니다. (간결성을 위해 I/O 관련 코드는 별도 메소드로 분리했다고 가정)

**Task.java**

Java

import java.util.Objects;  
  
public class Task {  
 private String description;  
 private boolean completed;  
  
 public Task(String description) {  
 this.setDescription(description); // Use setter for validation  
 this.completed = false;  
 }  
  
 // Getters  
 public String getDescription() {  
 return description;  
 }  
  
 public boolean isCompleted() {  
 return completed;  
 }  
  
 // Setters  
 public void setDescription(String description) {  
 if (description!= null &&!description.trim().isEmpty()) {  
 this.description = description;  
 } else {  
 // In a real app, might throw an exception or handle differently  
 System.out.println("Warning: Task description cannot be empty. Using default.");  
 this.description = "Untitled Task";  
 }  
 }  
  
 public void setCompleted(boolean completed) {  
 this.completed = completed;  
 }  
  
 @Override  
 public String toString() {  
 return "할 일: " + description + " (완료: " + (completed? "Y" : "N") + ")";  
 }  
  
 // equals() and hashCode() might be needed for certain operations like remove(Object)  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null |  
| getClass()!= o.getClass()) return false;  
 Task task = (Task) o;  
 return completed == task.completed && Objects.equals(description, task.description);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.hash(description, completed);  
 }  
}

**DeadlineTask.java**

Java

import java.util.Objects;  
  
public class DeadlineTask extends Task {  
 private String deadline;  
  
 public DeadlineTask(String description, String deadline) {  
 super(description); // Call parent constructor  
 this.deadline = deadline;  
 }  
  
 // Getter and Setter for deadline  
 public String getDeadline() {  
 return deadline;  
 }  
  
 public void setDeadline(String deadline) {  
 this.deadline = deadline;  
 }  
  
 @Override  
 public String toString() {  
 return super.toString() + ", 마감일: " + deadline;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null |  
| getClass()!= o.getClass()) return false;  
 if (!super.equals(o)) return false; // Check parent fields  
 DeadlineTask that = (DeadlineTask) o;  
 return Objects.equals(deadline, that.deadline);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.hash(super.hashCode(), deadline);  
 }  
}

**ToDoApp.java**

Java

import java.io.\*;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Scanner;  
import java.util.stream.Collectors;  
  
public class ToDoApp {  
  
 private static List<Task> tasks = new ArrayList<>();  
 private static final String FILENAME = "todolist.txt"; // 파일 이름 상수  
  
 public static void main(String args) {  
 loadTasksFromFile(FILENAME); // 앱 시작 시 파일 로드  
 Scanner scanner = new Scanner(System.in);  
 boolean isRunning = true;  
  
 System.out.println("=== 콘솔 할 일 목록 관리 앱 ===");  
  
 while (isRunning) {  
 printMenu();  
 int choice = scanner.nextInt();  
 scanner.nextLine(); // Consume newline  
  
 switch (choice) {  
 case 1:  
 addTask(scanner);  
 break;  
 case 2:  
 viewTasks();  
 break;  
 case 3:  
 markTaskComplete(scanner);  
 break;  
 case 4:  
 viewIncompleteTasks();  
 break;  
 case 5: // 저장 기능 추가  
 saveTasksToFile(FILENAME);  
 break;  
 case 6: // 종료 기능 번호 변경  
 System.out.println("앱을 종료합니다.");  
 saveTasksToFile(FILENAME); // 종료 시 자동 저장  
 isRunning = false;  
 break;  
 default:  
 System.out.println("잘못된 입력입니다. 다시 선택해주세요.");  
 break;  
 }  
 }  
 scanner.close();  
 }  
  
 private static void printMenu() {  
 System.out.println("\n1. 할 일 추가");  
 System.out.println("2. 할 일 목록 보기");  
 System.out.println("3. 할 일 완료 처리");  
 System.out.println("4. 미완료 할 일 보기");  
 System.out.println("5. 파일에 저장");  
 System.out.println("6. 종료");  
 System.out.print("메뉴를 선택하세요: ");  
 }  
  
 private static void addTask(Scanner scanner) {  
 System.out.println("할 일 종류를 선택하세요 (1: 일반, 2: 마감일): ");  
 int taskType = scanner.nextInt();  
 scanner.nextLine(); // Consume newline  
  
 System.out.print("할 일 내용을 입력하세요: ");  
 String description = scanner.nextLine();  
  
 if (taskType == 1) {  
 tasks.add(new Task(description));  
 System.out.println("일반 할 일이 추가되었습니다.");  
 } else if (taskType == 2) {  
 System.out.print("마감일을 입력하세요 (YYYY-MM-DD): ");  
 String deadline = scanner.nextLine();  
 tasks.add(new DeadlineTask(description, deadline));  
 System.out.println("마감일 있는 할 일이 추가되었습니다.");  
 } else {  
 System.out.println("잘못된 할 일 종류입니다.");  
 }  
 }  
  
 private static void viewTasks() {  
 System.out.println("\n--- 전체 할 일 목록 ---");  
 if (tasks.isEmpty()) {  
 System.out.println("등록된 할 일이 없습니다.");  
 } else {  
 for (int i = 0; i < tasks.size(); i++) {  
 System.out.println((i + 1) + ". " + tasks.get(i));  
 }  
 }  
 System.out.println("--------------------");  
 }  
  
 private static void markTaskComplete(Scanner scanner) {  
 viewTasks(); // 완료할 목록을 먼저 보여줌  
 if (tasks.isEmpty()) return;  
  
 System.out.print("완료 처리할 할 일 번호를 입력하세요: ");  
 int taskNumberToComplete = scanner.nextInt();  
 scanner.nextLine(); // Consume newline  
  
 if (taskNumberToComplete > 0 && taskNumberToComplete <= tasks.size()) {  
 Task taskToComplete = tasks.get(taskNumberToComplete - 1);  
 if (!taskToComplete.isCompleted()) {  
 taskToComplete.setCompleted(true);  
 System.out.println("'" + taskToComplete.getDescription() + "' 할 일을 완료 처리했습니다.");  
 } else {  
 System.out.println("이미 완료된 할 일입니다.");  
 }  
 } else {  
 System.out.println("잘못된 번호입니다.");  
 }  
 }  
  
 private static void viewIncompleteTasks() {  
 List<Task> incompleteTasks = tasks.stream()  
 .filter(task ->!task.isCompleted())  
 .collect(Collectors.toList());  
  
 System.out.println("\n--- 미완료 할 일 목록 ---");  
 if (incompleteTasks.isEmpty()) {  
 System.out.println("모든 할 일을 완료했습니다!");  
 } else {  
 for (int i = 0; i < incompleteTasks.size(); i++) {  
 // 원본 리스트에서의 인덱스가 아닌, 미완료 목록에서의 순번 표시  
 System.out.println((i + 1) + ". " + incompleteTasks.get(i));  
 }  
 }  
 System.out.println("-----------------------");  
 }  
  
 // --- File I/O Methods ---  
 private static void saveTasksToFile(String filename) {  
 try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {  
 for (Task task : tasks) {  
 String taskType = (task instanceof DeadlineTask)? "D" : "T";  
 String completedStatus = task.isCompleted()? "true" : "false";  
 String line = taskType + "," + task.getDescription() + "," + completedStatus;  
  
 if (task instanceof DeadlineTask) {  
 line += "," + ((DeadlineTask) task).getDeadline();  
 }  
 writer.write(line);  
 writer.newLine();  
 }  
 System.out.println("할 일 목록을 '" + filename + "' 파일에 저장했습니다.");  
 } catch (IOException e) {  
 System.err.println("파일 저장 중 오류 발생: " + e.getMessage());  
 }  
 }  
  
 private static void loadTasksFromFile(String filename) {  
 File file = new File(filename);  
 if (!file.exists()) {  
 // System.out.println("저장된 할 일 파일이 없습니다. 새 목록을 시작합니다.");  
 return; // 파일 없으면 조용히 리턴  
 }  
  
 try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {  
 String line;  
 tasks.clear(); // 기존 목록 초기화  
 while ((line = reader.readLine())!= null) {  
 // 빈 줄이나 잘못된 형식 방지  
 if (line.trim().isEmpty()) continue;  
 String parts = line.split(",", -1); // limit -1 to keep trailing empty strings if any  
  
 if (parts.length >= 3) {  
 String taskType = parts;  
 String description = parts[1];  
 boolean completed = Boolean.parseBoolean(parts[2]);  
  
 Task task;  
 if ("D".equals(taskType) && parts.length >= 4) {  
 String deadline = parts[3];  
 task = new DeadlineTask(description, deadline);  
 } else if ("T".equals(taskType)) {  
 task = new Task(description);  
 } else {  
 System.err.println("경고: 알 수 없는 할 일 타입 '" + taskType + "' - 건너뜁니다.");  
 continue; // Skip invalid line  
 }  
 task.setCompleted(completed);  
 tasks.add(task);  
 } else {  
 System.err.println("경고: 잘못된 형식의 라인 '" + line + "' - 건너뜁니다.");  
 }  
 }  
 System.out.println("'" + filename + "' 파일에서 할 일 목록을 불러왔습니다.");  
 } catch (IOException e) {  
 System.err.println("파일 로딩 중 오류 발생: " + e.getMessage());  
 } catch (Exception e) { // Catch potential parsing errors etc.  
 System.err.println("파일 로딩 중 예기치 않은 오류 발생: " + e.getMessage());  
 tasks.clear(); // Clear list if loading fails to avoid partial data  
 }  
 }  
}

### **B. 학습 내용 요약 및 중요성 강조**

이 가이드를 통해 스프링 백엔드 개발에 필수적인 Java 기초 역량을 다졌습니다.

* **개발 환경 설정:** JDK와 IDE를 설치하고 프로젝트를 생성하는 방법을 익혔습니다. 이는 모든 Java 개발의 시작입니다.5
* **기본 문법:** 변수, 데이터 타입, 제어문, 연산자 등 프로그래밍의 기본 요소를 Java 방식으로 사용하는 법을 배웠습니다.15
* **객체 지향 프로그래밍 (OOP):** 클래스, 객체, 캡슐화, 상속, 다형성의 개념을 Task와 DeadlineTask를 통해 실습했습니다. 이는 스프링의 핵심 설계 원리인 IoC/DI와 AOP를 이해하는 기반이 됩니다.16
* **컬렉션 프레임워크:** ArrayList를 사용하여 여러 객체를 관리하고, List, Set, Map의 차이점을 이해했습니다. 백엔드에서는 DB에서 가져온 많은 데이터를 효과적으로 다루기 위해 컬션 지식이 필수적입니다.19
* **스트림 API:** filter, map, collect를 사용하여 데이터를 함수형 스타일로 처리하는 방법을 익혔습니다. 복잡한 비즈니스 로직 내에서 데이터를 효율적으로 가공하는 데 유용합니다.24
* **(선택) 기본 I/O:** 파일에 데이터를 저장하고 읽는 기본적인 방법을 배웠습니다. 실제 웹 애플리케이션에서는 더 복잡한 데이터 저장 방식(DB)을 사용하지만, 데이터 영속성의 기본 개념을 이해하는 데 도움이 됩니다.32

### **C. 다음 단계를 향하여**

탄탄한 Java 기초를 다졌으니, 이제 스프링 생태계에 입문할 준비가 되었습니다. 다음 단계에서는 스프링 프레임워크의 핵심 원리인 \*\*IoC(제어의 역전)\*\*와 **DI(의존성 주입)** 개념을 배우고, \*\*스프링 부트(Spring Boot)\*\*를 사용하여 빠르고 간편하게 스프링 프로젝트를 시작하는 방법을 익히게 될 것입니다.

지금까지 학습한 Java 지식은 스프링의 다양한 기능을 이해하고 활용하는 데 든든한 발판이 될 것입니다. 꾸준한 학습과 실습을 통해 목표하시는 스프링 백엔드 개발자로 성장하시기를 응원합니다!

#### 참고 자료

1. 자바(java) 문법 정리 - DATA & AI, 4월 12, 2025에 액세스, <https://ingu627.github.io/java/java_gr1/>
2. JDK Installation Guide for macOS, Linux, and Windows - Noel Kamphoa, 4월 12, 2025에 액세스, <https://nkamphoa.com/install-and-configure-the-java-development-kit-jdk-on-macos-linux-and-windows/>
3. 초보자를 위한 Java 코딩: 올바른 방법으로 Java 배우기 - CodeGym, 4월 12, 2025에 액세스, <https://codegym.cc/ko/groups/posts/ko.169.chobojaleul-wihan-java-koding-olbaleun-bangbeob-eulo-java-baeugi>
4. tinajeong/univRankAnalyzer: :baby\_chick: 자바초보를 위한 프로젝트 (doc 폴더를 참고하세요) - GitHub, 4월 12, 2025에 액세스, <https://github.com/tinajeong/univRankAnalyzer>
5. How to Install JDK (on Windows, Mac OS, Ubuntu) and Get Started ..., 4월 12, 2025에 액세스, <https://www3.ntu.edu.sg/home/ehchua/programming/howto/JDK_HowTo.html>
6. [Java] 이클립스에서 자바 프로젝트 생성하기 - 혼공학습단 - 한빛미디어, 4월 12, 2025에 액세스, <https://hongong.hanbit.co.kr/java-%EC%9D%B4%ED%81%B4%EB%A6%BD%EC%8A%A4%EC%97%90%EC%84%9C-%EC%9E%90%EB%B0%94-%ED%94%84%EB%A1%9C%EC%A0%9D%ED%8A%B8-%EC%83%9D%EC%84%B1%ED%95%98%EA%B8%B0/>
7. [JAVA] JDK 설치 - 환경설정&주의사항 총정리(Windows/Linux/Mac ..., 4월 12, 2025에 액세스, <https://m.blog.naver.com/itso-/222107858644>
8. [부트캠프] 개발자를 위한 IntelliJ로 JAVA 프로젝트 만들기, 4월 12, 2025에 액세스, <https://bitconsulting.tistory.com/entry/%EB%B6%80%ED%8A%B8%EC%BA%A0%ED%94%84-%EA%B0%9C%EB%B0%9C%EC%9E%90%EB%A5%BC-%EC%9C%84%ED%95%9C-IntelliJ%EB%A1%9C-JAVA-%ED%94%84%EB%A1%9C%EC%A0%9D%ED%8A%B8-%EB%A7%8C%EB%93%A4%EA%B8%B0>
9. Intellij 다운 및 프로젝트 처음 시작하기 - 쏘니의 개발블로그, 4월 12, 2025에 액세스, <https://juntcom.tistory.com/230>
10. [JAVA] IntelliJ IDEA Community로 JAVA 프로젝트 생성하기 - 나는야 개발자 김토끼A, 4월 12, 2025에 액세스, <https://pink-rabbit.tistory.com/19>
11. [JAVA] 이클립스 프로젝트 생성 및 실행 방법 - 버물리의 IT공부, 4월 12, 2025에 액세스, <https://javacpro.tistory.com/4>
12. [Java/Eclipse] 프로젝트 만드는 방법 (Hello World!) - 기억의 궁전, 4월 12, 2025에 액세스, <https://parkjye.tistory.com/73>
13. IntelliJ IDEA에서 Java 프로젝트 설정하기 - 블로그 - 주경야근, 4월 12, 2025에 액세스, <https://burn.eone.one/blog/2024/235/intellij-java-new-project/>
14. [Java] 자바 입문 강의 텍스트 정리 - 대두코기, 4월 12, 2025에 액세스, <https://hoohaha.tistory.com/85>
15. 입력과 출력 - 생활코딩, 4월 12, 2025에 액세스, <https://opentutorials.org/course/1223/5575>
16. 11. Java 자바 - Getter와 Setter 메소드 - Kephi Javatory - 티스토리, 4월 12, 2025에 액세스, <https://kephilab.tistory.com/54>
17. Java 자바의 getter, setter - 용이의 개발블로그, 4월 12, 2025에 액세스, <https://youngdroidstudy.tistory.com/entry/Java-%EC%9E%90%EB%B0%94%EC%9D%98-getter-setter>
18. [Java] 자료구조 List, Set, Map의 차이 - 포포의 개발공부방, 4월 12, 2025에 액세스, <https://popo015.tistory.com/102>
19. [간단정리] List, Set, Map 특징 및 차이점(+ 구현체 ) — 넌 잘하고 있어, 4월 12, 2025에 액세스, <https://hahahoho5915.tistory.com/35>
20. [Java] 자바 컬렉션 프레임워크(List, Set, Map) - 슬기로운 개발생활 - Tistory.com, 4월 12, 2025에 액세스, <https://dev-coco.tistory.com/32>
21. 자바 ArrayList 구조 & 사용법 정리 - Inpa Dev ‍ - 티스토리, 4월 12, 2025에 액세스, <https://inpa.tistory.com/entry/JAVA-%E2%98%95-ArrayList-%EA%B5%AC%EC%A1%B0-%EC%82%AC%EC%9A%A9%EB%B2%95>
22. [Java] List, Arraylist 사용법 및 활용 - We Can it - 티스토리, 4월 12, 2025에 액세스, <https://wecanit.tistory.com/27>
23. 자바) ArrayList 사용법 & 예제 - velog, 4월 12, 2025에 액세스, <https://velog.io/@jybin96/%EC%9E%90%EB%B0%94-ArrayList-%EC%82%AC%EC%9A%A9%EB%B2%95-%EC%98%88%EC%A0%9C>
24. JAVA stream API 예제로 이해하기 - 뭐요 - 티스토리, 4월 12, 2025에 액세스, <https://dmansp.tistory.com/63>
25. [JAVA] Stream()의 filter() 메서드를 활용한 요소 필터링 - velog, 4월 12, 2025에 액세스, <https://velog.io/@jungmyeong96/JAVA-Stream%EC%9D%98-filter-%EB%A9%94%EC%84%9C%EB%93%9C%EB%A5%BC-%ED%99%9C%EC%9A%A9%ED%95%9C-%EC%9A%94%EC%86%8C-%ED%95%84%ED%84%B0%EB%A7%81>
26. [JAVA] Java Stream 활용법: Filter, Map, FlatMap 예제와 함께 - 오늘도 열심히 - 티스토리, 4월 12, 2025에 액세스, <https://kylo8.tistory.com/entry/JAVA-Java-Stream-%ED%99%9C%EC%9A%A9%EB%B2%95-Filter-Map-FlatMap-%EC%98%88%EC%A0%9C%EC%99%80-%ED%95%A8%EA%BB%98>
27. 자바 part.4 문제풀이 - 자바의 다형성(polymorphism), 4월 12, 2025에 액세스, <https://071217.tistory.com/200>
28. [Java] 상속과 다형성 실전 활용 예제 - 람스의 개발일지, 4월 12, 2025에 액세스, <https://space-engineers-developer.tistory.com/20>
29. [Java] 객체 지향의 4대 개념 추상화/상속/다형성/캡슐화 - 보통의 개발자 - 티스토리, 4월 12, 2025에 액세스, <https://bnzn2426.tistory.com/119>
30. [Spring] Spring IoC와 DI란? - 느리더라도 꾸준하게 - 티스토리, 4월 12, 2025에 액세스, <https://steady-coding.tistory.com/600>
31. [토비의 스프링] 1장 정리 [IoC와 DI] - Devjun's Devlog - 티스토리, 4월 12, 2025에 액세스, <https://devjun.tistory.com/358>
32. [Java] 파일 입출력(BufferedWriter, BufferedReader, Files) | CodeNexus, 4월 12, 2025에 액세스, <https://umanking.github.io/2021/06/25/java-file-system/>
33. [Java] 빠른 입출력을 위한 BufferedReader, BufferedWriter, StringTokenizer, StringBuilder, 4월 12, 2025에 액세스, <https://rlakuku-program.tistory.com/33>
34. [Java] BufferedReader / BufferedWriter를 활용한 빠른 입출력 - Taking an extra step, 4월 12, 2025에 액세스, <https://livebyfaith117.tistory.com/108>
35. Spring AOP 활용 - velog, 4월 12, 2025에 액세스, <https://velog.io/@juhyeon1114/Spring-AOP-%ED%99%9C%EC%9A%A9>
36. [Spring] AOP 개념 및 적용하기 - dgjinsu - 티스토리, 4월 12, 2025에 액세스, <https://dgjinsu.tistory.com/31>