

Day 01: 자바 기초

1. 자바 학습환경

- JDK 설치
- IDE(Eclipse) 설치 및 환경 설정

2. 변수와 데이터 타입

- 기본 데이터 타입
- 참조 데이터 타입

3. 연산자

- 산술, 비교, 논리 연산자 등

4. 입출력

- 표준 입력 (`Scanner` 클래스)
- 표준 출력 (`System.out.println`)

5. 제어문

- `if`, `switch` 조건문

6. 반복문

- `for`, `while`, `do-while` 반복문

7. 클래스와 객체

- 클래스 선언
- 객체 생성 및 활용

8. 필드, 메서드

9. `this` 와 `static`

10. 객체를 저장하는 배열

11. 연습문제 및 과제

1. 자바 학습환경

a. JDK (Java Development Kit) 설치

JDK는 자바 어플리케이션을 개발하기 위한 도구와 자바 런타임 환경을 제공합니다.

1. JDK의 필요성

- 자바 컴파일러(`javac`) 사용
- 자바 런타임 환경(JRE) 포함

2. 설치 과정

- 공식 Oracle 웹사이트 혹은 OpenJDK 사이트 접속
- 학습자의 운영체제에 알맞은 JDK 버전 다운로드
- 설치 파일 실행 및 설치 방향 따르기
- 환경 변수 설정 (선택적)
 - `JAVA_HOME` 설정

- `PATH` 변수에 JDK의 bin 디렉토리 추가

3. 설치 확인

- 터미널이나 커맨드 프롬프트에서 `java -version` 및 `javac -version` 명령어를 이용해 설치된 자바 버전 확인

b. Eclipse IDE 다운로드 및 설치 방법


Eclipse의 개념과 특징

Eclipse는 오픈 소스 기반으로 개발되었으며, 다양한 플러그인을 통해 다양한 개발 환경을 지원합니다. 주로 Java 개발에 많이 사용되며, 자바 개발을 위한 기능뿐만 아니라 웹 개발, 모바일 애플리케이션 개발 등에도 활용됩니다. Eclipse는 가볍고 확장성이 높아서 개발자들이 필요한 기능을 추가로 설치하거나 커스터마이징하여 사용할 수 있습니다.

1. Eclipse 다운로드

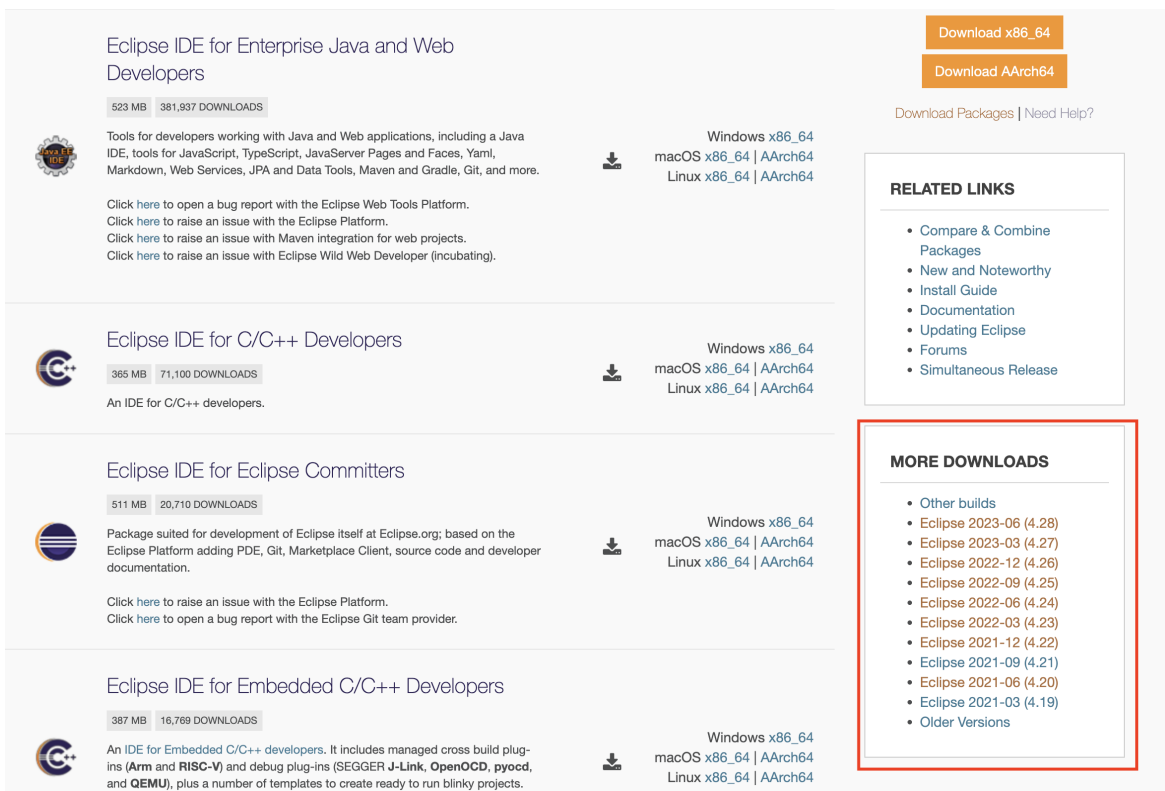
- [Eclipse 공식 웹사이트](#)에 접속합니다.

1. <https://www.eclipse.org/downloads/packages/release/2023-06/r>

Eclipse Packages | The Eclipse Foundation - home to a global community, the Eclipse IDE, Jakarta EE and over 350 open source projects...
The Eclipse Installer 2023-06 R now includes a JRE for macOS, Windows and Linux.
 <https://www.eclipse.org/downloads/packages/release/2023-06/r>

- 해당 웹페이지에서 원하는 Eclipse 패키지를 선택합니다. 다양한 패키지가 제공되며, 자신이 개발할 언어나 환경에 맞는 패키지를 선택할 수 있습니다.

1. 우측의 MORE DOWNLOADS 메뉴에서 이전 버전 선택 가능:



The screenshot shows the Eclipse IDE download page. It lists several IDE packages for different developers: Eclipse IDE for Enterprise Java and Web Developers, Eclipse IDE for C/C++ Developers, Eclipse IDE for Eclipse Committers, and Eclipse IDE for Embedded C/C++ Developers. Each package has a download button and a list of supported operating systems and architectures. On the right side, there is a 'RELATED LINKS' section and a 'MORE DOWNLOADS' section. The 'MORE DOWNLOADS' section is highlighted with a red box and contains a list of previous Eclipse versions with their release dates.

- 선택한 패키지를 클릭하면 다운로드 페이지로 이동합니다. 다운로드 페이지에서 자신의 운영체제에 맞는 다운로드 링크를 클릭합니다.

2. 설치

- 다운로드한 파일을 컴퓨터로 이동시킵니다. 보통 다운로드된 파일은 압축 파일일 것입니다.
- 압축 파일을 C 드라이브 최상위(root) 경로에 압축 해제합니다. 이 때, Eclipse 폴더가 생성되어야 합니다. 경로는 `C:\eclipse` 와 같이 설정할 수 있습니다.
- 압축 해제한 Eclipse 폴더 내부로 들어가면 실행 파일인 `eclipse.exe` 가 있는 것을 확인할 수 있습니다.
- `eclipse.exe` 파일을 더블 클릭하여 Eclipse IDE를 실행합니다.

이렇게 하면 Eclipse IDE가 다운로드되고 설치되어 실행 준비가 완료됩니다. Eclipse는 다양한 개발 작업을 지원하는 강력한 IDE이며, 해당 패키지에 따라 Java, 웹 개발, 데이터베이스 등 다양한 개발 환경을 설정할 수 있습니다.

3. Eclipse 실행 및 기본 설정 확인

a. 실행

- Eclipse를 다운로드하고 압축을 해제한 후, 해당 폴더 내에서 `eclipse.exe` 파일을 더블 클릭하여 Eclipse IDE를 실행합니다.
- 처음 실행할 때는 작업 공간(workspace)을 선택하라는 창이 나타날 수 있습니다. 작업 공간은 프로젝트와 설정 파일 등이 저장되는 폴더로, 본인이 원하는 경로를 선택하면 됩니다.

b. 워크스페이스 설정

- Eclipse를 실행하면 "Welcome" 화면이 표시됩니다. 이 화면에서는 새로운 프로젝트 생성이나 기존 프로젝트 불러오기 등의 작업을 시작할 수 있습니다.

c. Perspective 변경

- Eclipse는 다양한 작업 환경인 Perspective를 제공합니다. 예를 들어 Java 개발을 위한 "Java Perspective"나 웹 개발을 위한 "Web Perspective" 등이 있습니다.
- 화면 상단에 있는 Perspective 전환 아이콘을 클릭하여 필요한 Perspective로 변경할 수 있습니다. 해당 Perspective로 전환하면 해당 작업 환경에 필요한 도구와 뷰가 표시됩니다.

d. 메뉴 및 도구

- Eclipse는 메뉴, 아이콘, 도구 모음 등 다양한 UI 요소를 제공하여 코드 작성, 디버깅, 빌드 등의 작업을 수행할 수 있도록 합니다.
- 상단 메뉴에서 원하는 작업을 수행할 수 있는 메뉴 항목을 선택합니다. 예를 들어 "File" 메뉴에서는 프로젝트를 생성하거나 파일을 열고 저장할 수 있습니다.
- 아이콘과 도구 모음은 작업을 보다 편리하게 수행할 수 있도록 도와줍니다. 예를 들어 코드 편집기, 디버깅 도구, Git 관리 등의 기능을 아이콘으로 사용할 수 있습니다.

4. 첫 번째 자바 프로젝트 생성

- `File > New > Java Project` 선택하여 새 프로젝트 생성
- 생성한 프로젝트 내에 새로운 클래스 파일 추가 및 "Hello, World!" 출력 프로그램 작성 및 실행

Eclipse는 자바 어플리케이션 개발을 위한 통합 개발 환경(IDE)입니다. Eclipse 말고도 많이 사용되는 통합 개발 환경으로는 IntelliJ가 있습니다. IntelliJ에는 좀 더 많은 기능이 있어서 편리한 면도 있지만 그만큼 초보자가 다루기엔 쉽지 않다는 것이 단점입니다. 일단 JAVA 학습에는 좀 더 사용하기 쉬운 Eclipse를 사용 하고 차후에 더 많은 기능을 가진 IDE로 바뀌어서 사용 하길 바랍니다.

C. 기본적인 "Hello, World!" 출력 프로그램 구현

이러한 과정을 통해 자바 개발 환경을 구축하고, 첫 번째 자바 프로그램을 작성하고 실행 해 보겠습니다.

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

실행 방법:

- 위의 소스코드 HelloWorld라는 이름의 클래스로 저장.
- 소스코드에서 마우스 오른쪽 클릭(우클릭) > Run As > Java Application 실행.
- 이클립스 하단 탭 Console 탭이 열리면서 결과 출력.

2. 변수와 데이터 타입

a. 기본 데이터 타입 (Primitive Data Types)

목적: 자바에서 가장 기본적으로 제공되는 데이터 타입들을 이해하고, 이들을 활용해 변수를 선언하고 사용하는 방법을 배우는 것입니다.

1. 자바 기본 데이터 타입

자바의 기본 데이터 타입은 총 8가지입니다. 그중 문자형, 실수형, 논리형을 제외하면 나머지 4가지는 모두 정수형입니다. 정수형은 크기 순서대로 byte, short, int, long 입니다. 엄밀히 문자형도 정수형의 일종이라 할수 있습니다.

유형	타입	byte(bit)	크기	용도
정수형	byte	1(8)	-128 ~ 127	작은 정수
정수형	short	2(16)	-32,768 ~ 32,767	중간 크기의 정수
정수형	int	4(32)	-2 ³¹ ~ 2 ³¹ -1	정수의 기본 형
정수형	long	8(64)	-2 ⁶³ ~ 2 ⁶³ -1	매우 큰 정수
실수형	float	4(32)	±3.40282347E+38F (7 소수점)	단정밀도 실수
실수형	double	8(64)	±1.79769313486231570E+308	실수의 기본형
문자형	char	2(16)	0 ~ 65,535 (유니코드)	단일 문자
논리형	boolean	1bit	true 또는 false	참/거짓 논리값(JVM에 따라 bit크기가 다름)

참고: boolean의 byte 크기와 bit 크기는 JVM(Java Virtual Machine) 구현에 따라 다를 수 있습니다. 따라서 구체적인 크기는 JVM의 문서나 스펙을 참조해야 합니다.

2. 변수 선언과 초기화

- 변수의 선언 방법

자바에서 변수를 선언할 때는 데이터 타입을 먼저 명시하고, 그 다음에 변수 이름을 작성합니다.

- 문법:

```
데이터형 변수명;
```

- 예시:

```
int age;
double salary;
char initial;
boolean isActive;
```

3. 변수에 값 할당 방법

변수에 값을 할당할 때는 = 연산자를 사용합니다. 변수가 선언되면 그 변수에 값을 할당할 수 있습니다.

- 예시:

```
int age = 30;
double salary = 50000.50;
char initial = 'A';
boolean isActive = true;
```

또한 변수를 먼저 선언하고 나중에 값을 할당할 수도 있습니다.

- 예시:

```
int age;
age = 30;

double salary;
salary = 50000.50;
```

4. 변수 초기화 (변수 값 초기화)

변수를 선언할 때 값을 바로 할당하면 그 변수는 초기화된 것입니다.

- 예시:

```
int count = 0; // 변수 'count'를 선언하고 0으로 초기화
```

변수를 선언만 하고 값을 할당하지 않으면 그 변수는 초기화되지 않은 상태입니다. 클래스의 멤버 필드 변수로 선언된 경우 기본값으로 자동 초기화되지만, 메서드 내의 로컬 변수는 반드시 사용하기 전에 초기화해야 합니다. 초기화되지 않은 로컬 변수를 사용하려고 하면 컴파일 오류가 발생합니다.

참고: 클래스의 멤버 변수는 자동으로 기본값으로 초기화되는데, 예를 들어 `int` 타입의 멤버 변수는 0으로, `boolean` 타입의 멤버 변수는 `false` 로 초기화됩니다.

b. 참조 데이터 타입 (Reference Data Types)

목적: 자바에서 객체의 메모리 주소를 저장하는 변수의 타입을 이해하며, 기본 데이터 타입이 아닌 데이터를 저장하고 관리하는 방법을 배우는 것입니다.

1. 객체와 참조 변수의 관계

- 메모리상의 객체 위치를 가리키는 참조 변수의 개념

자바는 객체 지향 프로그래밍 언어로, 데이터와 메서드를 함께 묶어 객체로 표현합니다. 이때, 객체의 실제 데이터는 메모리의 특정 영역에 저장되며, 이 영역을 가리키는 변수를 참조 변수라고 합니다.

- 객체와 참조 변수의 관계

항목	설명
객체(Object)	실제 메모리에 할당된 인스턴스. 객체는 속성(필드)과 행동(메서드)을 가진다.
참조 변수(Reference Variable)	메모리에 할당된 객체의 주소값을 저장하는 변수. 해당 변수를 통해 객체에 접근할 수 있다.
메모리 구조	- 스택(Stack): 참조 변수가 저장되는 영역. 변수의 스코프가 끝나면 자동으로 메모리 해제됨. - 힙(Heap): 실제 객체가 저장되는 영역. JVM의 Garbage Collector에 의해 더 이상 참조되지 않는 객체는 해제됨.

- 예제

```
Person person = new Person(); // 'person'이 참조 변수, 'new Person()'이 객체 생성
```

위 예제에서 `new Person()` 을 통해 `Person` 객체가 메모리의 힙 영역에 생성됩니다. 그리고 `person` 이라는 참조 변수가 이 객체의 메모리 주소를 가리키게 됩니다. 따라서 `person` 변수를 통해 해당 객체의 메서드나 필드에 접근할 수 있습니다.

자바에서 객체를 할당하고 사용하는 방식은 이러한 참조 변수를 통한 간접 참조 방식입니다.

2. 주요 참조 데이터 타입

참조 데이터 타입은 메모리의 주소값을 참조하여 값을 사용합니다. 이는 기본 데이터 타입과는 달리, 실제 데이터가 저장된 메모리 위치의 주소값을 변수에 저장한다는 의미입니다. 이러한 참조 데이터 타입에는 여러 클래스와 인터페이스가 있으며, 그 중 여기서는 `String`, `Arrays`, `Object` 에 대해 간략히 알아보겠습니다.

참조 타입	설명
String	- 문자열을 표현하는 클래스 - 불변(immutable)한 객체로, 한번 생성된 문자열은 변경되지 않습니다. - <code>+</code> , <code>concat()</code> 등의 연산을 통해 문자열을 합치거나 수정하는 경우, 새로운 <code>String</code> 객체가 생성됩니다.
Arrays	- 배열을 다루기 위한 유틸리티 클래스 - 배열의 복사, 정렬, 검색 등의 연산을 지원합니다. - 기본 데이터 타입 및 참조 타입 모두에 사용 가능합니다.
Object	- 자바에서 모든 클래스의 최상위 클래스 - <code>Object</code> 클래스에 정의된 메서드는 모든 자바 객체에 공통적으로 사용할 수 있습니다. - <code>equals()</code> , <code>hashCode()</code> , <code>toString()</code> 등의 메서드가 이 클래스에 정의되어 있습니다.

3. 배열

배열은 동일한 타입의 여러 변수를 하나의 묶음으로 다루는 자료구조입니다. 자바에서 배열은 기본 데이터 타입 뿐만 아니라 객체 참조를 위한 참조 타입으로도 사용할 수 있습니다.

항목	설명
배열 선언	- 배열의 데이터 타입과 배열 이름을 함께 선언합니다. - 예: <code>int[] numbers;</code> or <code>String[] names;</code>
배열 생성	- <code>new</code> 키워드를 사용하여 배열의 크기를 지정하고 메모리를 할당합니다. - 예: <code>numbers = new int[5];</code> (5개의 int 값 저장 가능)
배열 초기화	- 배열을 선언과 동시에 값을 초기화할 수 있습니다. - 예: <code>int[] numbers = {1, 2, 3, 4, 5};</code>
	- 배열의 특정 위치에 값을 할당하려면 인덱스를 사용합니다. (인덱스는 0 부터 시작) - 예: <code>names[0] = "Alice";</code>
다차원 배열	- 배열의 배열로, 주로 행렬이나 테이블 데이터를 표현하는 데 사용됩니다. - 선언 예: <code>int[][] matrix;</code>
	- 생성 예: <code>matrix = new int[3][4];</code> (3x4 행렬)
	- 초기화 예: <code>int[][] matrix = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};</code>

배열은 연속된 메모리 공간에 동일한 타입의 데이터를 순차적으로 저장하며, 인덱스를 통해 빠르게 데이터에 접근할 수 있습니다. 다차원 배열은 실제로는 1차원 배열의 배열로 구성되며, 각 1차원 배열은 다른 길이를 가질 수 있습니다.

4. Boxing과 Unboxing

자바는 기본 데이터 타입의 값을 객체로 변환하거나, 객체에서 기본 데이터 타입의 값을 얻어내기 위해 `Boxing`과 `Unboxing`이라는 개념을 제공합니다. 이를 가능하게 하는 것은 기본 데이터 타입에 대응하는 'wrapper 클래스'입니다.

항목	설명
Boxing	- 기본 데이터 타입의 값을 해당하는 wrapper 클래스의 객체로 변환하는 것. - 자동(Autoboxing) 혹은 수동으로 할 수 있음.
Unboxing	- wrapper 클래스의 객체에서 기본 데이터 타입의 값을 얻어내는 것. - 자동(Auto-unboxing) 혹은 수동으로 할 수 있음.
Wrapper 클래스	- 기본 데이터 타입을 객체로 다룰 수 있게 하는 클래스. - 각 기본 데이터 타입에 대응하는 wrapper 클래스가 있음.

- 주요 Wrapper 클래스

기본 데이터 타입	Wrapper 클래스
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

- 예시

Boxing: 기본 타입의 값을 wrapper 객체로 변환

```
int i = 10;
Integer intObject = i; // 자동 boxing
Integer manualIntObject = Integer.valueOf(i); // 수동 boxing
```

Unboxing: wrapper 객체에서 기본 타입의 값을 추출

```
Integer intObject = 10;
int j = intObject; // 자동 unboxing
int manualInt = intObject.intValue(); // 수동 unboxing
```

Boxing과 Unboxing은 제네릭, 컬렉션, 메서드 오버로딩 등 다양한 곳에서 활용되므로, 기본 데이터 타입과 wrapper 클래스 사이의 관계를 이해하는 것이 중요합니다.

실습

1. 각 기본 데이터 타입에 따른 변수 선언과 값 할당

```
public class DataTypePractice {
    public static void main(String[] args) {
        byte byteValue = 10;
        short shortValue = 20;
        int intValue = 100;
        long longValue = 1000L;
        float floatValue = 10.5f;
        double doubleValue = 10.5;
        char charValue = 'A';
        boolean booleanValue = true;

        System.out.println("byteValue: " + byteValue);
        System.out.println("shortValue: " + shortValue);
        // ... (나머지도 출력)
    }
}
```

2. String을 활용한 문자열 연산 실습

```
public class StringPractice {
    public static void main(String[] args) {
        String str1 = "Hello";
        String str2 = "World";

        String combined = str1 + " " + str2;
        int length = combined.length();

        System.out.println("Combined string: " + combined);
    }
}
```

```

        System.out.println("Length of combined string: " + length);
    }
}

```

3. 1차원 및 2차원 배열 생성 및 사용 실습

```

public class ArrayPractice {
    public static void main(String[] args) {
        int[] singleDimension = {1, 2, 3, 4, 5};
        int[][] multiDimension = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };

        System.out.println("First element of single dimension: " + singleDimension[0]);
        System.out.println("Element at [1][2] of multi dimension: " + multiDimension[1][2]);
    }
}

```

4. Boxing과 Unboxing 실습

```

public class BoxingUnboxingPractice {
    public static void main(String[] args) {
        // Boxing
        int intValue = 10;
        Integer boxedInt = intValue;

        // Unboxing
        Integer anotherBoxedInt = 20;
        int unboxedInt = anotherBoxedInt;

        System.out.println("Boxed integer: " + boxedInt);
        System.out.println("Unboxed integer: " + unboxedInt);
    }
}

```

위의 실습 예제를 통해 각 주제별로 Java의 기본 데이터 타입 및 관련 연산을 경험해 볼 수 있습니다.


3. 연산자

연산자의 개요



연산자는 특정 연산을 수행하기 위한 기호나 키워드입니다. 자바에서는 다양한 연산자를 제공하며, 이를 이용해 값을 계산하거나 조건을 판단하고, 논리 연산을 수행할 수 있습니다.

학습 내용

1. 산술 연산자

-  : 덧셈
-  : 뺄셈
-  : 곱셈
-  : 나눗셈 (몫)
-  : 나머지 연산

2. 비교 연산자

-  : 같다
-  : 다르다

- `<`: 작다
- `<=`: 작거나 같다
- `>`: 크다
- `>=`: 크거나 같다

3. 논리 연산자

- `&&`: AND (논리곱)
- `||`: OR (논리합)
- `!`: NOT (논리 부정)

4. 삼항 연산자

- 조건 `?` 참일때 `:` 거짓일때;

5. 단항연산, 증감연산, 대입연산, 복합대입연산, 괄호 등

실습

1. 간단한 산술 연산 수행하기

목표: 사용자로부터 두 개의 숫자를 입력 받아, 덧셈, 뺄셈, 곱셈, 나눗셈의 결과를 출력하는 프로그램을 작성한다.

```
import java.util.Scanner;

public class ArithmeticOperations {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.println("첫 번째 숫자를 입력하세요:");
        double num1 = sc.nextDouble();

        System.out.println("두 번째 숫자를 입력하세요:");
        double num2 = sc.nextDouble();

        System.out.println("덧셈 결과: " + (num1 + num2));
        System.out.println("뺄셈 결과: " + (num1 - num2));
        System.out.println("곱셈 결과: " + (num1 * num2));
        System.out.println("나눗셈 결과: " + (num1 / num2));
    }
}
```

2. 두 값을 비교하여 결과 확인하기

목표: 사용자로부터 두 개의 숫자를 입력 받아, 두 숫자가 같은지, 다른지, 어떤 숫자가 큰지를 판별하는 프로그램을 작성한다.

```
import java.util.Scanner;

public class ComparisonOperations {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.println("첫 번째 숫자를 입력하세요:");
        double num1 = sc.nextDouble();

        System.out.println("두 번째 숫자를 입력하세요:");
        double num2 = sc.nextDouble();

        if (num1 == num2) {
            System.out.println("두 숫자는 같습니다.");
        } else if (num1 > num2) {
            System.out.println(num1 + "은(는) " + num2 + "보다 큼니다.");
        } else {
            System.out.println(num1 + "은(는) " + num2 + "보다 작습니다.");
        }
    }
}
```

```
}  
}
```

3. 논리 연산을 통한 조건의 참/거짓 판별하기

목표: 사용자로부터 나이와 학년을 입력 받아, 해당 사용자가 성인이면서 대학생일 경우 참인지 거짓인지를 판별하는 프로그램을 작성한다.

```
import java.util.Scanner;  
  
public class LogicalOperations {  
    public static void main(String[] args) {  
        Scanner sc = new Scanner(System.in);  
  
        System.out.println("나이를 입력하세요:");  
        int age = sc.nextInt();  
  
        System.out.println("학년을 입력하세요 (1~4):");  
        int grade = sc.nextInt();  
  
        if (age >= 20 && (grade >= 1 && grade <= 4)) {  
            System.out.println("당신은 성인이면서 대학생입니다.");  
        } else {  
            System.out.println("당신은 성인이면서 대학생이 아닙니다.");  
        }  
    }  
}
```

각 실습을 진행한 후에, 예상된 출력 결과와 실제 출력 결과를 비교하여 코드의 동작을 이해하고, 필요한 경우 수정 및 개선을 진행하세요.

4. 입출력 (I/O)

프로그램은 사용자와의 상호작용을 위해 데이터를 입력받고, 결과를 출력하는 작업을 자주 수행합니다. Java에서는 다양한 입출력 방법을 제공하며, 이 중에서 기본적인 표준 입력 및 출력 방법에 대해 알아봅니다.

1. 표준 입력 (`Scanner` 클래스)

`Scanner` 클래스는 사용자로부터 데이터를 입력받는데 사용되는 클래스입니다. `Scanner` 클래스를 사용하기 위해서는 `java.util` 패키지를 import 해야 합니다.

```
import java.util.Scanner;  
  
public class InputExample {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
  
        System.out.print("문자열을 입력하세요: ");  
        String inputString = scanner.nextLine();  
  
        System.out.print("정수를 입력하세요: ");  
        int inputInt = scanner.nextInt();  
  
        System.out.print("실수를 입력하세요: ");  
        double inputDouble = scanner.nextDouble();  
  
        System.out.println("입력받은 문자열: " + inputString);  
        System.out.println("입력받은 정수: " + inputInt);  
        System.out.println("입력받은 실수: " + inputDouble);  
    }  
}
```

2. 표준 출력 (`System.out.println`)

Java에서 가장 흔히 사용되는 출력 메서드는 `System.out.println()` 입니다. 이 메서드를 사용하여 콘솔에 데이터를 출력할 수 있습니다.

```
public class OutputExample {
    public static void main(String[] args) {
        int number = 10;
        String message = "Hello, World!";

        System.out.println("정수 출력: " + number);
        System.out.println("문자열 출력: " + message);
        System.out.printf("정수 출력 (printf): %d\\n", number);
        System.out.printf("문자열 출력 (printf): %s\\n", message);
    }
}
```

`System.out.printf()` 메서드를 사용하면 서식을 지정하여 출력할 수 있습니다. `%d` 는 정수, `%s` 는 문자열, `%f` 는 실수를 나타내는 서식 지정자입니다.

5. 제어문 (Control Statements)

프로그램에서 특정 조건에 따라 다른 코드 블록을 실행하게 하는 것은 중요한 개념입니다. 제어문을 사용하면 프로그램의 흐름을 제어할 수 있습니다. 여기서는 가장 기본적인 제어문인 `if` 와 `switch` 에 대해 알아보겠습니다.

1. `if` 조건문

`if` 문은 주어진 조건이 참인 경우 해당 코드 블록을 실행합니다. `else if` 와 `else` 를 이용해 추가적인 조건 및 그 외의 경우에 실행될 코드 블록을 지정할 수 있습니다.

```
int number = 10;

if (number > 5) {
    System.out.println("Number is greater than 5.");
} else if (number == 5) {
    System.out.println("Number is equal to 5.");
} else {
    System.out.println("Number is less than 5.");
}
```

2. `switch` 조건문

`switch` 문은 변수의 값에 따라 여러 코드 블록 중 하나를 실행합니다. `case` 키워드를 사용하여 변수의 가능한 값과 해당 값에 따라 실행될 코드 블록을 지정하며, `default` 키워드를 사용하여 변수의 값이 어떤 `case` 에도 해당되지 않는 경우에 실행될 코드 블록을 지정할 수 있습니다.

```
int dayOfWeek = 3;
String dayName;

switch (dayOfWeek) {
    case 1:
        dayName = "Sunday";
        break;
    case 2:
        dayName = "Monday";
        break;
    case 3:
        dayName = "Tuesday";
        break;
    case 4:
        dayName = "Wednesday";
        break;
    case 5:
        dayName = "Thursday";
        break;
    case 6:
```

```

        dayName = "Friday";
        break;
    case 7:
        dayName = "Saturday";
        break;
    default:
        dayName = "Invalid day";
        break;
}

System.out.println("Today is " + dayName);

```

실습

다음은 성적을 입력받아 학점을 출력하는 프로그램을 구현 하세요. 사용자에게 성적을 입력 받은 후, 주어진 요구 조건에 따라 학점과 기호(+, -)를 결정합니다.

- 입력된 성적이 0~100점 사이가 아니면 Error 출력
- 60점 미만은 F학점
- 성적이 60~100점 사이면 A, B, C, D 학점으로 분류
- 성적의 1의 자기가 7이상이면 학점 뒤에 +가 붙고, 3이하면 -가 붙도록

6. 반복문

반복문은 프로그램에서 특정 작업을 반복적으로 수행하고자 할 때 사용하는 구조입니다. 자바에는 `for`, `while`, `do-while` 세 가지 주요 반복문이 있습니다.

1. `for` 반복문

- 주로 시작과 종료 조건이 명확할 때 사용합니다.
- 구조:

```

for(초기화; 조건식; 증감식) {
    // 반복 실행될 코드
}

```

- 예제:

```

for(int i = 0; i < 10; i++) {
    System.out.println("i의 값: " + i);
}

```

2. `while` 반복문

- 조건이 참인 동안 반복을 수행합니다.
- 시작과 종료 조건이 불명확할 때 유용합니다.
- 구조:

```

while(조건식) {
    // 반복 실행될 코드
}

```

- 예제:

```
int count = 0;
while(count < 5) {
    System.out.println("count의 값: " + count);
    count++;
}
```

3. **do-while** 반복문

- **while** 반복문과 유사하지만, 조건 검사 전에 최소한 한 번은 코드가 실행됩니다.
- 구조:

```
do {
    // 반복 실행될 코드
} while(조건식);
```

- 예제:

```
int number;
Scanner scanner = new Scanner(System.in);
do {
    System.out.print("10 이상의 수를 입력하세요: ");
    number = scanner.nextInt();
} while(number < 10);
```

실습:

1. 1부터 10까지의 합을 구하는 프로그램을 **for** 을 사용하여 구현하세요.
2. 사용자로부터 숫자를 입력받아 해당 숫자의 구구단을 출력하는 프로그램을 for 문을 상용하여 작성하세요.
3. 시작단, 종료단을 입력 받아서 시작단~종료단까지의 구구단이 모두 출력 되도록 하세요. 단, 시작단이 종료단 보다 큰 수가 들어와도 정상 출력 되도록 하세요.

7. 클래스와 객체

7.1 클래스 선언

클래스는 객체 지향 프로그래밍에서 데이터와 이를 처리하는 메서드를 함께 묶는 템플릿입니다. 클래스는 객체의 설계도 또는 틀로 생각할 수 있습니다.

```
public class ClassName {
    // 멤버 변수 (필드)
    dataType variableName;

    // 메서드
    returnType methodName(parameters) {
        // 메서드 본문
    }
}
```

- 예제:

```
public class Dog {
    // 필드
    String breed;
    int age;
    String color;

    // 메서드
```

```
void bark() {
    System.out.println("Woof Woof!");
}
}
```

7.2 객체 생성 및 활용

클래스는 설계도와 같으며, 실제로 사용하기 위해서는 해당 클래스의 객체를 생성해야 합니다.

```
ClassName objectName = new ClassName();
```

- **예제:**

```
Dog myDog = new Dog();
```

생성된 객체를 통해 클래스의 필드와 메서드에 접근할 수 있습니다.

```
myDog.breed = "Golden Retriever";
myDog.age = 5;
myDog.bark();
```

7.3 참조 변수와 인스턴스(객체)

자바는 객체 지향 프로그래밍 언어로, 여러 객체를 사용하여 프로그램을 구현합니다. 이때, 이 객체를 사용하기 위해서는 참조 변수와 인스턴스(또는 객체)의 개념을 잘 이해해야 합니다.

1. 인스턴스(객체)

- **정의:** 클래스에 정의된 속성과 메서드를 가진 실체입니다. 이것은 "실제로 메모리 상에 생성된 것"을 의미합니다.
- **생성 방법:** `new` 키워드를 사용하여 생성합니다.

```
new ClassName();
```

2. 참조 변수

- **정의:** 객체의 메모리 주소를 가리키는 변수입니다. 이 변수를 통해 객체에 접근하고 해당 객체의 메서드를 호출하거나 속성을 사용할 수 있습니다.
- **선언 방법:**

```
ClassName variableName;
```

- **할당 방법:** 객체 생성과 동시에 참조 변수에 할당할 수 있습니다.

```
ClassName variableName = new ClassName();
```

이해를 돕는 예시:

클래스와 객체의 관계를 상상하는 좋은 방법은 '붕어빵 틀과 붕어빵'에 비유하는 것입니다.

- **클래스(Class):** 붕어빵 틀입니다. 붕어빵의 모양과 크기, 패턴을 정의합니다. 그러나 실제로 먹을 수 있는 붕어빵은 아닙니다.
- **객체(Instance):** 붕어빵 틀로 만든 실제 붕어빵입니다. 실제로 먹을 수 있고, 각각의 붕어빵은 자신만의 팔(데이터)을 가지고 있습니다.

이때 **참조 변수**는 만들어진 붕어빵을 손에 들고 있는 것과 같습니다. 손을 통해 붕어빵의 팔을 확인하거나 붕어빵을 먹는 것처럼, 참조 변수를 통해 객체의 데이터를 확인하거나 객체의 메서드를 호출할 수 있습니다.

참조 변수와 기본 데이터 타입 변수의 차이:

- **기본 데이터 타입 변수:** 실제 값을 직접 저장합니다.

```
int number = 10;
```

- **참조 변수:** 메모리 상의 객체 위치(주소)를 저장합니다.

```
ClassName obj = new ClassName();
```

결론적으로, 객체는 메모리 상에 데이터와 메서드를 포함한 블록으로 실제로 존재하며, 참조 변수는 이 객체를 가리키는 포인터 또는 링크와 같은 역할을 합니다.

실습:

1. `Person` 이라는 이름의 클래스를 생성하고, `name`, `age`, `address` 라는 필드를 정의하세요.
2. `Person` 클래스에 `introduce` 라는 메서드를 추가하여, 해당 객체의 `name`, `age`, `address` 정보를 출력하는 코드를 작성하세요.
3. `Person` 클래스의 객체를 생성하고, 위에서 정의한 필드와 메서드를 활용하여 정보를 저장하고 출력하는 코드를 작성하세요.

8. 필드와 메서드

8. 필드와 메서드

클래스 내부의 구성 요소는 크게 필드와 메서드로 나눌 수 있습니다.

8.1 필드 (Field)

필드는 클래스나 인터페이스의 멤버 변수를 의미하며, 객체의 상태 또는 속성을 정의하는 데 사용됩니다.

- **기본 구조:**

```
class ClassName {  
    dataType fieldName;  
}
```

- **예제:**

```
class Car {  
    String brand;  
    int year;  
    boolean isElectric;  
}
```

- **특징:**

- 객체마다 독립적인 값을 갖게 됩니다.
- 직접 접근을 제한하고, 메서드를 통해 값을 가져오거나 설정하는 것이 좋습니다(getter와 setter).
- `static` 키워드를 사용하면 클래스 변수가 됩니다. 이는 객체마다 독립적이지 않고, 클래스에 속한 하나의 고정된 값이 됩니다.

8.2 메서드 (Method)

메서드는 특정 작업을 수행하는 일련의 코드를 포함하며, 데이터를 처리하거나 계산을 수행하는 데 사용됩니다.

- 기본 구조:

```
returnType methodName(parameterList) {  
    // 메서드 본문  
}
```

- 예제:

```
void printDetails() {  
    System.out.println("This is a method example.");  
}  
  
int add(int a, int b) {  
    return a + b;  
}
```

- 특징:

- 메서드는 파라미터를 받을 수 있으며, 필요한 경우 값을 반환할 수 있습니다.
- `static` 메서드는 클래스에 속하며 객체 생성 없이 호출할 수 있습니다.
- 메서드 오버로딩: 같은 이름의 메서드를 다른 매개 변수 목록으로 여러 번 정의하는 것.
- 메서드 오버라이딩: 상속을 통해 부모 클래스의 메서드를 자식 클래스에서 재정의하는 것.

물론 클래스의 구성 요소는 이것만 있는 것이 아닙니다. 생성자, 초기화 블록, 내부 클래스 등 여러 추가 요소들이 있지만 필드와 메서드는 클래스의 주요 구성 요소로서 중요한 역할을 합니다.

9. 메서드 오버로드

메서드 오버로드 (Method Overloading)

메서드 오버로드는 하나의 클래스 내에서 같은 이름의 메서드를 여러 개 정의하는 것을 의미합니다. 오버로드된 메서드는 매개변수의 타입, 개수, 순서 등이 다르게 정의되어야 합니다. 반환 타입만 다른 경우에는 오버로드로 간주되지 않습니다.

메서드 오버로드의 규칙:

1. 메서드 이름이 같아야 합니다.
2. 메서드의 매개변수 목록이 달라야 합니다. (타입, 개수, 순서 등)
3. 반환 타입은 메서드 오버로딩과 관련이 없습니다. 즉, 반환 타입만 다르고 매개변수가 동일한 두 메서드는 오버로드 될 수 없습니다.

메서드 오버로드의 장점:

1. **명확성:** 함수 이름이 같아서 기능이 유사하거나 관련된 여러 메서드를 동일한 이름으로 정의하면 코드의 가독성이 향상됩니다.
2. **유연성:** 다양한 매개변수를 허용하여 사용자에게 다양한 옵션을 제공할 수 있습니다.

예제:

```
public class Calculator {  
  
    // 정수 덧셈  
    public int add(int a, int b) {  
        return a + b;  
    }  
}
```



```

    }

    // 실수 덧셈
    public double add(double a, double b) {
        return a + b;
    }

    // 세 개의 정수 덧셈
    public int add(int a, int b, int c) {
        return a + b + c;
    }
}

```

위 예제에서 `add` 메서드는 세 번 오버로드되었습니다. 첫 번째는 두 개의 정수를 매개변수로 받고, 두 번째는 두 개의 실수를 매개변수로 받으며, 세 번째는 세 개의 정수를 매개변수로 받습니다.

이렇게 메서드 오버로드를 통해 여러 형태의 매개변수를 받을 수 있게 되어, 같은 기능을 하는 메서드들을 동일한 이름으로 제공할 수 있게 됩니다. 이는 코드의 간결성과 가독성에 기여합니다.

10. 생성자와 생성자 오버로드

생성자 (Constructor)

정의와 특징:

- 생성자는 클래스의 객체가 생성될 때 자동으로 호출되는 특별한 메서드입니다.
- 생성자의 이름은 클래스 이름과 동일해야 합니다.
- 생성자는 값을 반환하지 않습니다. 즉, 반환 타입을 가질 수 없으며, `void`도 사용할 수 없습니다.
- 객체 생성 시 초기화 작업을 수행하기 위해 주로 사용됩니다.

기능:

1. **객체 초기화:** 객체가 생성될 때 멤버 변수나 상태를 초기화하는 데 사용됩니다.
2. **객체 생성 제어:** 특정 조건 하에서만 객체 생성을 허용하거나, 객체 생성과 관련된 로직을 수행할 수 있습니다.

생성자 오버로드 (Constructor Overloading)

생성자 오버로드는 하나의 클래스 내에서 여러 개의 생성자를 정의하는 것을 의미합니다. 각각의 생성자는 다른 타입 또는 개수의 매개변수를 가집니다.

생성자 오버로드의 장점:

1. **유연한 객체 생성:** 동일한 클래스의 객체를 다양한 방식으로 초기화하여 생성할 수 있습니다.
2. **코드의 간결성:** 생성자를 통해 필요한 초기화 작업을 모두 수행할 수 있으므로, 객체 생성 후 별도의 초기화 메서드 호출이 불필요합니다.

예제:

```

public class Rectangle {
    private int width;
    private int height;

    // 기본 생성자
    public Rectangle() {
        this.width = 0;
        this.height = 0;
    }

    // 너비와 높이를 매개변수로 받는 생성자
    public Rectangle(int width, int height) {

```

```

        this.width = width;
        this.height = height;
    }

    // 정사각형을 만들기 위한 생성자
    public Rectangle(int side) {
        this.width = side;
        this.height = side;
    }
}

```

위 예제에서 `Rectangle` 클래스는 세 가지 생성자를 가지며, 이를 통해 다양한 방식으로 객체를 초기화할 수 있습니다.

생성자와 생성자 오버로드를 통해 클래스의 객체를 유연하게 생성하고 초기화하는 과정을 쉽고 효율적으로 관리할 수 있습니다.

11. `this` 와 `static`

`this` 키워드

정의와 특징:

- `this` 는 현재 객체의 참조를 나타내는 키워드입니다.
- `this` 는 주로 클래스 내부에서 사용되며, 해당 클래스의 현재 인스턴스를 참조할 때 사용합니다.

주요 사용 경우:

1. **필드와 지역 변수 구분:** 메서드의 매개변수 또는 지역 변수와 클래스의 필드(멤버 변수)가 이름이 같을 때 이들을 구분하기 위해 사용됩니다.

```

public class Person {
    private String name;

    public Person(String name) {
        this.name = name; // 여기서 'this.name'은 클래스의 필드를 의미하며 'name'은 매개변수를 의미합니다.
    }
}

```

1. **다른 생성자 호출:** 하나의 생성자에서 같은 클래스의 다른 생성자를 호출할 때 사용됩니다.

```

public class Rectangle {
    private int width;
    private int height;

    public Rectangle() {
        this(0, 0); // 동일 클래스의 다른 생성자 호출
    }

    public Rectangle(int width, int height) {
        this.width = width;
        this.height = height;
    }
}

```

1. **현재 인스턴스 반환:** 메서드에서 현재 인스턴스를 반환할 때 사용될 수 있습니다.

`static` 키워드

정의와 특징:

- `static` 은 클래스 변수나 클래스 메서드를 정의할 때 사용되는 키워드입니다.
- `static` 으로 선언된 변수 또는 메서드는 객체가 아닌 클래스에 속하며, 모든 객체가 공유합니다.
- `static` 멤버는 클래스가 로드될 때 메모리에 할당되고, 프로그램 종료 시까지 메모리에서 해제되지 않습니다.

주요 사용 경우:

1. **클래스 변수:** 모든 객체가 공통으로 사용하는 변수를 선언할 때 사용합니다. 해당 변수는 모든 객체에 의해 공유됩니다.

```
public class Student {
    private static int totalStudents = 0; // 모든 Student 객체가 공유하는 변수
    // ...
}
```

1. **클래스 메서드:** 객체를 생성하지 않고도 호출할 수 있는 메서드입니다. 이 메서드 내에서는 `non-static` 멤버에 직접 접근할 수 없습니다.

```
public class MathUtil {
    public static int add(int a, int b) {
        return a + b;
    }
}
```

이렇게 선언된 메서드는 `MathUtil.add(1, 2)` 와 같이 호출할 수 있습니다.

1. **초기화 블록:** `static` 초기화 블록은 클래스가 로드될 때 한 번만 실행됩니다. 주로 `static` 변수의 복잡한 초기화에 사용됩니다.

```
public class SomeClass {
    static {
        // 초기화 코드
    }
}
```

`this` 와 `static` 은 자바에서 각기 다른 목적으로 사용되며, 이들의 특징과 사용법을 이해하는 것은 객체 지향 프로그래밍의 핵심 개념 중 하나입니다.

12. 객체를 저장 하는 배열

객체를 저장하는 배열

자바에서 배열은 기본 데이터 타입 뿐만 아니라 객체 참조 값도 저장할 수 있습니다. 객체를 저장하는 배열은 실제로 객체를 직접 저장하는 것이 아니라, 객체에 대한 참조(주소)를 저장합니다.

생성과 초기화:

객체를 저장하는 배열을 생성하는 방법은 일반적인 배열 생성 방법과 동일합니다.

```
Student[] students = new Student[5]; // 5명의 학생 객체를 저장할 수 있는 배열 생성
```

위 코드에서 `students` 배열은 `Student` 타입의 객체 참조를 5개 저장할 수 있는 공간을 할당합니다. 초기에는 모든 원소가 `null` 로 설정됩니다.

객체 할당:

배열의 각 요소에 객체를 할당하기 위해서는 먼저 객체를 생성하고 해당 참조를 배열에 저장해야 합니다.

```
students[0] = new Student("John");
students[1] = new Student("Jane");
```

주의 사항:

1. **초기화되지 않은 참조 사용:** 객체 배열은 생성될 때 모든 요소가 `null`로 초기화됩니다. 초기화되지 않은 배열 요소를 사용하려고 하면 `NullPointerException`이 발생할 수 있습니다.

```
Student[] students = new Student[5];
System.out.println(students[0].getName()); // NullPointerException 발생
```

1. **배열의 크기는 고정:** 자바에서 배열의 크기는 고정되어 있으므로, 일단 생성되면 크기를 변경할 수 없습니다. 더 많은 객체를 저장하려면 새 배열을 생성하고 필요한 내용을 복사해야 합니다. 동적으로 크기를 조절하려면 `ArrayList`와 같은 컬렉션 프레임워크를 사용해야 합니다.
2. **객체 참조의 공유:** 배열에 저장된 객체 참조는 다른 변수와 공유될 수 있습니다. 따라서 한 변수를 통해 객체를 수정하면, 해당 객체를 참조하는 다른 변수에서도 그 변경이 반영됩니다.

```
Student john = new Student("John");
students[0] = john;

john.setName("Johnny");
System.out.println(students[0].getName()); // "Johnny" 출력
```

1. **메모리 누수:** 배열에서 객체 참조를 제거하려면 해당 요소를 `null`로 설정해야 합니다. 그렇지 않으면, 그 객체는 가비지 컬렉터에 의해 수거되지 않아 메모리 누수가 발생할 수 있습니다.

```
students[0] = null; // 이를 통해 Student 객체 참조 제거
```

객체를 저장하는 배열을 사용할 때 위의 주의 사항들을 염두에 두고 코드를 작성하면 대부분의 문제를 피할 수 있습니다.

연습문제

높다 낮다 게임

높다 낮다 게임은 컴퓨터가 무작위로 선택한 숫자를 사용자가 추측하는 게임입니다. 사용자는 숫자를 입력하면 컴퓨터는 사용자의 추측이 정답보다 높은지 낮은지 알려줍니다. 사용자는 컴퓨터의 힌트를 바탕으로 숫자를 조정해 다시 추측합니다. 이를 반복하여 사용자가 정답을 맞출 때까지 계속됩니다.

실행 시나리오

1. 컴퓨터는 1부터 100 사이의 랜덤한 숫자를 선택합니다.
2. 사용자는 컴퓨터가 선택한 숫자를 맞추려고 시도합니다.
3. 사용자가 숫자를 입력하면, 컴퓨터는 다음 중 하나의 힌트를 제공합니다.
 - 입력한 숫자가 정답보다 낮다면 "높습니다!"
 - 입력한 숫자가 정답보다 높다면 "낮습니다!"
 - 입력한 숫자가 정답과 같다면 "정답입니다!"를 출력하고 게임을 종료합니다.
4. 사용자가 정답을 맞출 때까지 2-3 단계를 반복합니다.

요구 사항

- 컴퓨터가 선택하는 숫자는 1부터 100 사이의 랜덤한 숫자여야 합니다.
- 사용자는 숫자 외의 값을 입력하면 "잘못된 입력입니다"라는 메시지를 받아야 합니다.
- 게임이 종료된 후에는 사용자에게 다시 게임을 할지 물어보고, 사용자의 응답에 따라 게임을 다시 시작하거나 종료합니다.

힌트

```
import java.util.Random;
import java.util.Scanner;

public class HighLowGame {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Random random = new Random();

        while (true) {
            // 1. 컴퓨터는 1부터 100 사이의 랜덤한 숫자를 선택합니다.
            int targetNumber = random.nextInt(100) + 1;
            System.out.println("숫자를 맞춰보세요! (1~100)");

            while (true) {
                // 2. 사용자가 숫자를 입력
                System.out.print("입력: ");
                String userInput = scanner.nextLine();

                // 3. 정답 판별 및 힌트 제공
            }

            // 게임 종료 후 다시 시작할 것인지 물어봄
            System.out.println("다시 게임을 하시겠습니까? (yes/no)");
            String restartResponse = scanner.nextLine();

            // 입력 값이 yes라면 게임 종료
        }

        scanner.close();
        System.out.println("Good bye!");
    }
}
```

과제

주소록 프로그램 구현

주소록은 개인이나 회사에서 연락처 정보를 관리하는 중요한 도구입니다. 이 연습 문제에서는 객체 지향 프로그래밍의 핵심 개념을 활용하여 간단한 주소록 프로그램을 구현합니다.

실행 시나리오:

1. 사용자는 주소록 프로그램을 실행하면 다음과 같은 메인 메뉴를 볼 수 있습니다.

===== MENU =====

(1)연락처 추가 (2) 목록 출력 (3) 검색 (4) 수정 (5) 삭제 (6) 종료

선택:

2. 사용자는 메뉴에서 원하는 기능을 선택하여 실행할 수 있습니다.

요구사항:

1. **Person** 클래스를 생성하세요.

- `name`: 연락처의 이름
- `age`: 연락처의 나이
- `address`: 연락처의 주소

```
class Person {
    private String name;
    private int age;
    private String address;

    public Person(String name, int age, String address) {
        this.name = name;
        this.age = age;
        this.address = address;
    }

    // Getters and Setters 추가

    @Override
    public String toString() {
        return "Name: " + name + ", Age: " + age + ", Address: " + address;
    }
}
```

2. `AddressBook` 클래스를 생성하세요. 이 클래스는:

- `Person` 객체들의 `ArrayList`를 포함해야 합니다.
- 연락처 추가, 출력, 검색, 수정, 삭제 기능을 메서드로 구현하세요.

```
import java.util.ArrayList;
import java.util.Scanner;

class AddressBook {
    private ArrayList<Person> persons;

    public AddressBook() {
        this.persons = new ArrayList<>();
    }

    public void addPerson() {
        // 기능 구현
    }

    public void displayPersons() {
        // 기능 구현
    }

    public void searchPerson() {
        // 기능 구현
    }

    public void editPerson() {
        // 기능 구현
    }

    public void deletePerson() {
        // 기능 구현
    }
}
```

3. 메인 클래스 (`Main`)에서 메인 메뉴를 구현하세요. 메뉴는 사용자의 입력을 받아 해당 기능을 실행하도록 해야 합니다.

```
public class Main {
    public static void main(String[] args) {
        AddressBook addressBook = new AddressBook();
        Scanner sc = new Scanner(System.in);

        while (true) {
            System.out.println("===== MENU =====");
            System.out.println("(1)연락처 추가 (2) 목록 출력 (3) 검색 (4) 수정 (5) 삭제 (6) 종료");
        }
    }
}
```

```

        System.out.print("선택: ");
        int choice = sc.nextInt();
        sc.nextLine(); // consume newline

        switch (choice) {
            case 1:
                addressBook.addPerson();
                break;
            case 2:
                addressBook.displayPersons();
                break;
            case 3:
                addressBook.searchPerson();
                break;
            case 4:
                addressBook.editPerson();
                break;
            case 5:
                addressBook.deletePerson();
                break;
            case 6:
                System.out.println("Exiting...");
                sc.close();
                return;
            default:
                System.out.println("Invalid choice! Please choose again.");
        }
    }
}

```

Tip: 주소록 프로그램은 간단한 프로젝트임에도 불구하고 객체 지향 프로그래밍의 주요 개념 (클래스, 객체, 상속, 다형성 등) 을 실제로 구현하고 이해하는 데 도움이 됩니다. 프로그램의 기능을 확장하거나 다른 기능을 추가하며 자신만의 주소록 프로그램을 만들어 보세요!

머리 좀 식히기

https://youtube.com/shorts/W8ZJu-njGAc?si=aDt9gITzD_UTrOnp