추상화, 인터페이스 연습문제

🎶 연습문제 – 추상 클래스: 미디어 플레이어 시스템

📝 문제 설명

당신은 다양한 미디어 파일을 재생하는 플레이어 프로그램을 개발하고 있습니다.
모든 미디어 파일은 공통적으로 파일명을 가지고 있으며, 재생 방식은 파일 형식에 따라 다릅니다.
`MediaFile`이라는 추상 클래스를 기반으로 각 미디어 파일 클래스를 설계하고 기능을 구현해보세요.

💡 요구사항

1. `MediaFile` 추상 클래스
    - 멤버 변수: `filename` (String)
    - 일반 메서드: `getInfo()` → `"파일명: [파일명]"` 출력
    - 추상 메서드: `play()` → 각 미디어에 따라 다르게 재정의되어야 함
2. 다음 클래스를 `MediaFile` 클래스를 상속하여 만든다:
    - `AudioFile` → `play()` → `"오디오 파일을 재생합니다."`
    - `VideoFile` → `play()` → `"비디오 파일을 재생합니다."`
    - `ImageFile` → `play()` → `"이미지 파일을 표시합니다."`
3. `MediaFile` 배열에 다양한 미디어 파일을 저장하고,
   반복문을 통해 각 파일의 정보를 출력하고 `play()` 메서드를 실행한다.
4. (선택 과제) `VideoFile` 클래스에 `showSubtitles()` 메서드를 추가하고,
   다운캐스팅을 통해 해당 기능을 호출하시오.
5. main 클래스를 제외한 모든 클래스는 default 클래스 이다.

🧪 출력 예시

- 파일명: music.mp3
  오디오 파일을 재생합니다.
- 파일명: movie.mp4
  비디오 파일을 재생합니다.
- 파일명: picture.jpg
  이미지 파일을 표시합니다.
- 자막: 영화 자막을 표시합니다.

☑️ 학습 목표

- 추상 클래스와 추상 메서드 정의
- 메서드 오버라이딩의 강제성 이해
- 다형성 + 배열 활용
- 다운캐스팅을 통한 고유 기능 호출

🏫 연습문제 – 추상 클래스: 학교 수업 관리 시스템

📝 문제 설명

당신은 학교 수업 정보를 관리하는 프로그램을 개발하고 있습니다.
모든 수업은 공통적인 정보를 가지고 있지만, 수업 방식(온라인/오프라인/혼합)은 서로 다릅니다.
추상 클래스와 상속, 오버라이딩, 다운캐스팅을 활용하여 다양한 수업을 표현하고,
각 수업이 시작될 때 어떤 방식으로 진행되는지를 출력해보세요.

💡 요구사항

1. 추상 클래스 `Course`를 만든다.
    - 멤버 변수: `courseName`(수업명), `teacherName`(교사명)
    - 일반 메서드: `showInfo()` → `"수업명: OO / 교사: OO"` 출력
    - 추상 메서드: `startClass()` → 수업 시작 방식 출력 (자식 클래스에서 구현)
2. 다음 클래스를 `Course` 클래스를 상속받아 만든다:
    - `OnlineCourse`
        - `startClass()` → `"Zoom 링크를 통해 온라인 수업을 시작합니다."` 출력
    - `OfflineCourse`
        - `startClass()` → `"지정된 강의실에서 대면 수업을 시작합니다."` 출력
    - `HybridCourse`
        - `startClass()` → `"온라인과 오프라인 수업을 병행합니다."` 출력
        - 고유 메서드: `switchMode(String mode)`
            - `"현재 수업 모드를 [mode]로 변경합니다."` 출력
3. `Course` 타입의 배열을 만들어 다양한 수업을 저장하고,
   `for`문으로 모든 수업의 정보를 출력하고, `startClass()`를 실행한다.
4. `HybridCourse`는 **다운캐스팅**을 통해 고유 메서드 `switchMode()`를 실행한다.
5. main 클래스를 제외한 모든 클래스는 default 클래스로 작성한다.

🧪 출력 예시

- 수업명: 자바 프로그래밍 / 교사: 김교수
  Zoom 링크를 통해 온라인 수업을 시작합니다.
- 수업명: 자료구조 / 교사: 이교수
  지정된 강의실에서 대면 수업을 시작합니다.
- 수업명: AI 프로젝트 / 교사: 박교수
  온라인과 오프라인 수업을 병행합니다.
- 현재 수업 모드를 실시간 화상으로 변경합니다.

☑ 학습 목표

- 추상 클래스와 추상 메서드 구현
- 오버라이딩을 통한 다형성 이해
- 객체 배열을 통한 수업 관리
- 다운캐스팅을 통한 고유 기능 접근

🚗 연습문제 – 추상화 심화: 스마트 교통수단 제어 시스템

📝 문제 설명

당신은 스마트 도시 교통 제어 시스템을 개발하고 있습니다.
이 도시에 존재하는 모든 교통수단은 **운행 시작, 운행 정지, 경로 안내 기능**을 공통적으로 갖고 있지만,
세부 동작은 수단별로 다릅니다.
또한 특정 교통수단만 가능한 **고유 기능**도 존재합니다.

이 시스템을 **추상 클래스 기반으로 설계**하고,
공통 메서드는 추상 클래스에서 구현,
고유 동작은 서브클래스에서 직접 구현하세요.

💡 요구사항

1. 추상 클래스 `Transport`
    - 멤버 변수: `vehicleId`, `status`
    - 생성자 포함
    - 일반 메서드:
        - `stop()` → `"교통수단 [vehicleId]가 정지합니다."` 출력
    - 추상 메서드:
        - `start()` → 운행 시작 메시지 출력 (교통수단에 따라 다름)
        - `navigate(String destination)` → 목적지 안내
2. 다음 클래스를 `Transport` 클래스를 상속하여 만든다:
    - `Bus`
        - `start()` → `"버스 [vehicleId]가 출발합니다."`
        - `navigate()` → `"버스가 [destination]으로 이동합니다."`
    - `Train`
        - `start()` → `"기차 [vehicleId]가 플랫폼을 떠납니다."`
        - `navigate()` → `"기차가 [destination]행 노선으로 진입합니다."`
        - 고유 메서드: `announceStations()`
          → `"기차가 다음 역들을 안내합니다..."` 출력
    - `Drone`
        - `start()` → `"드론 [vehicleId]가 이륙합니다."`
        - `navigate()` → `"드론이 항공 경로를 따라 [destination]으로 이동합니다."`
        - 고유 메서드: `captureSurroundings()`
          → `"드론이 주변 상황을 촬영합니다."` 출력
3. `Transport` 배열을 만들어 여러 교통수단을 저장하고,
   모든 수단에 대해 `start()`, `navigate("서울역")`, `stop()` 호출
4. `Train`과 `Drone`은 **다운캐스팅**을 통해 고유 기능도 호출할 것

🧪 출력 예시

- 버스 BUS-001가 출발합니다.
  버스가 서울역으로 이동합니다.
  교통수단 BUS-001가 정지합니다.

- 기차 TR-202가 플랫폼을 떠납니다.
  기차가 서울역행 노선으로 진입합니다.
  교통수단 TR-202가 정지합니다.
  기차가 다음 역들을 안내합니다...

- 드론 DRN-A7가 이륙합니다.
  드론이 항공 경로를 따라 서울역으로 이동합니다.
  교통수단 DRN-A7가 정지합니다.
  드론이 주변 상황을 촬영합니다.

☑ 학습 목표

- 추상 클래스에서 **일반 메서드와 추상 메서드의 구분** 이해
- 하위 클래스에서 **강제 구현의 의미와 목적** 체험
- **다형성 + 배열** 구조 이해
- **고유 기능 분리**와 **다운캐스팅 활용**
- **추상화를 통한 코드 설계의 이점** 체험 (유지보수, 확장성)

🔥 확장 과제 (보너스)

- 교통수단의 상태(status)를 "대기", "운행 중", "정지"로 관리하고, 상태 출력 메서드 구현
- `TransportManager` 클래스를 만들어 배열 제어 및 통계 출력
- 추상 클래스 대신 인터페이스를 설계하면 어떤 차이가 있을지 서술

🐯 연습문제 – 인터페이스 기초: 청소 로봇 제어 시스템

📝 문제 설명

당신은 다양한 청소 로봇을 제어하는 프로그램을 개발하고 있습니다.
청소 로봇마다 청소 방식은 다르지만, 공통적으로 **청소 시작**과 **청소 종료** 기능을 반드시 가져야 합니다.
이러한 공통 기능을 **인터페이스**로 정의하고,
여러 종류의 로봇 클래스에서 이를 구현하여 제어 프로그램을 완성해보세요.

💡 요구사항

1. 인터페이스 `Cleanable`을 정의한다.
    - 추상 메서드:
        - `startCleaning()`
        - `stopCleaning()`
2. 다음 클래스를 `Cleanable` 인터페이스를 구현하여 작성한다:
    - `VacuumRobot`
        - `startCleaning()` → `"진공 청소를 시작합니다."`
        - `stopCleaning()` → `"진공 청소를 종료합니다."`
    - `MopRobot`
        - `startCleaning()` → `"물걸레 청소를 시작합니다."`
        - `stopCleaning()` → `"물걸레 청소를 종료합니다."`
3. `Cleanable` 타입의 배열을 생성하여,
   두 개의 로봇 객체(`VacuumRobot`, `MopRobot`)를 저장하고,
   반복문을 통해 모든 로봇의 청소 시작과 종료를 실행한다.

📏 출력 예시

- 진공 청소를 시작합니다.
  진공 청소를 종료합니다.
- 물걸레 청소를 시작합니다.
  물걸레 청소를 종료합니다.

☑️ 학습 목표

- 인터페이스의 정의와 구현 방법 이해
- 클래스 간 공통 기능 설계 방법 습득
- 다형성과 배열을 통한 인터페이스 활용 실습
- 코드 확장성과 유지보수성을 인터페이스를 통해 체험

💳 연습문제 – 인터페이스 중간: 결제 시스템 구현

📝 문제 설명

온라인 쇼핑몰에서 다양한 결제 수단을 지원하는 시스템을 개발하려고 합니다.
모든 결제 수단은 `pay(int amount)` 기능을 반드시 가져야 하며,
결제 방식에 따라 내부 동작은 다릅니다.
이러한 구조를 인터페이스 기반으로 설계하고,
다양한 결제 수단을 구현해 다형성을 체험해보세요.

💡 요구사항

1. `Payable` 인터페이스를 정의한다.
    - 추상 메서드:
        - `pay(int amount)` → 결제 금액을 전달받아 처리
2. 다음 클래스를 `Payable` 인터페이스를 구현하여 작성한다:
    - `CreditCard`
      → `"신용카드로 [amount]원 결제합니다."`
    - `Cash`
      → `"현금으로 [amount]원 결제합니다."`
    - `MobilePay`
      → `"모바일 결제로 [amount]원 결제합니다."`
3. `Payable` 타입의 배열에 다양한 결제 수단을 저장하고,
   반복문을 통해 각각의 결제 방식으로 1만 원을 결제하시오.
4. (선택 과제) `MobilePay` 클래스에 고유 기능 `useBiometrics()` 를 추가하고,
   다운캐스팅을 이용하여 해당 기능을 호출하시오.

🖍️ 출력 예시

- 신용카드로 10000원 결제합니다.
- 현금으로 10000원 결제합니다.
- 모바일 결제로 10000원 결제합니다.
- 생체 인증으로 모바일 결제를 시작합니다.

☑️ 학습 목표

- 인터페이스 정의 및 구현
- 메서드 오버라이딩을 통한 역할 분리
- 다형성과 배열을 통한 제어
- 다운캐스팅을 통한 고유 기능 호출
- 인터페이스 기반 설계의 유연성과 확장성 체험

🤯 연습문제 – 최상위: 자율 로봇 임무 시스템

📝 문제 설명

당신은 다양한 임무를 수행하는 자율 로봇 시스템을 개발하고 있습니다.
모든 로봇은 공통적으로 **이름(name)**, **배터리 상태(batteryLevel)** 를 가지고 있으며,
로봇의 **작동(operate)** 방식은 로봇 종류마다 다릅니다.

또한, 어떤 로봇은 **비행**, 어떤 로봇은 **수색**, 어떤 로봇은 **구조 임무**를 수행합니다.
이러한 기능은 각각 독립적인 행동이므로 **인터페이스로 분리**합니다.

💡 요구사항

1. 추상 클래스 `Robot` 정의
    - 멤버 변수: `name`, `batteryLevel` (int)
    - 생성자, getter 포함
    - 일반 메서드: `showStatus()` → `"로봇명: [name], 배터리: [batteryLevel]%"`
    - 추상 메서드: `operate()` → 작동 메시지를 출력
2. 인터페이스 정의
    - `Flyable` → `fly()`
    - `Searchable` → `search()`
    - `Rescuable` → `rescue()`
3. 다음 클래스를 작성한다:
    - `DroneBot`
        - 상속: `Robot`
        - 구현: `Flyable`, `Searchable`
        - `operate()` → `"드론봇이 공중에서 임무를 시작합니다."`
    - `RescueBot`
        - 상속: `Robot`
        - 구현: `Rescuable`
        - `operate()` → `"구조봇이 현장에 투입됩니다."`
    - `HybridBot`
        - 상속: `Robot`
        - 구현: `Flyable`, `Searchable`, `Rescuable`
        - `operate()` → `"하이브리드봇이 다기능 모드로 작동합니다."`
4. `Robot[]` 배열에 다양한 로봇 객체를 저장하고,
   반복문으로 `showStatus()` 와 `operate()` 실행
5. `instanceof` 와 다운캐스팅을 통해 해당 인터페이스의 메서드를 조건부로 호출하시오.

🖋 출력 예시

- 로봇명: 드론-01, 배터리: 80%
  드론봇이 공중에서 임무를 시작합니다.
  드론이 비행합니다.
  드론이 수색 작업을 수행합니다.

- 로봇명: 구조-09, 배터리: 65%
  구조봇이 현장에 투입됩니다.
  구조 임무를 수행합니다.

- 로봇명: 하이브리드-X, 배터리: 95%
  하이브리드봇이 다기능 모드로 작동합니다.
  드론이 비행합니다.
  드론이 수색 작업을 수행합니다.
  구조 임무를 수행합니다.

☑ 학습 목표

- 추상 클래스와 인터페이스를 함께 활용한 설계 경험
- 역할에 따른 기능 분리와 조합
- 다형성과 조건 분기 로직 구현
- 유지보수성과 확장성이 높은 객체지향 설계 체험