GUI 구현

기말 평가용 과제

가상의 물품 거래 작업을 선정하여 해당 물품의 입고, 재고 확인, 판매, 자료 저장, 자료 읽기(로드) 등의 기능을 수행하는 프로그램을 작성합니다.
물품 거래와 상점이름 등은 자동차, 페인트, 가구, 자전거, 식료품, 채소, 과일 등 자신이 임의로 선택하여 프로젝트 스토리를 작성합니다.

대상 물품에 대한 클래스 정의 (물품명, 코드번호, 단가, 입고일자, 재고 수량, 이미지 등의 항목 필수)는 필수 항목과 자신의 선택항목 추가로 구성합니다. (Getter & Setter 메서드 활용 필수)

물품의 입고, 재고 확인, 판매, 자료 저장, 자료 로드(읽기) 등에 필요한 메서드 정의는 BasicFunction 인터페이스에 정의하고 이를 구현하는 방식을 적용합니다.

교재의 도서관리 프로젝트와 같은 형태의 GUI 구성도 추가합니다.

1. 사이트 구현 시나리오
   선택한 물품 판매 사이트(상점-shop)를 구성하는 이유와 전반적인 물품 및 진행 상황 설명
   첫 화면에 shop 로고 또는 대표 이미지와 상점이름, 개발자 이름 기재
   기본 클래스와 추가 메서드 설명
   자신만의 추가적인 기능, 디자인 구현 부분 설명
   물품 객체는 반드시 LinkedList 자료구조로 관리합니다.

2. 기능정의
   1) 물품 입고 : 물품 내용 입력, 입고 일자는 시스템에서 자동 저장, 코드는 각자 설계
      (물품 이미지를 10개 이상 준비하고 이를 선택하는 파일대화상자 활용)
      물품명, 코드, 단가, 수량, 입고일자, 구매처 등은 필수 항목으로
      해당 항목별 자료형은 자신이 임의로 지정
   2) 재고 확인 : 물품 이름 또는 코드 번호로 물품 상황 확인
   3) 믈품 판매 : 판매 물품의 내용을 입력 받아서 재고량 조정
   4) 자료 저장 : 파일에 물품 정보 저장하기
   5) 자료 로드 : 파일에서 물품 정보 읽어 오기

Main.java

```java
package com.carshop.main;
import com.carshop.service.CarManager;
import com.carshop.ui.MainFrame;
public class Main {
//        역할: 프로그램 시작점
//        해야 할 일:
//        main() 메서드 작성
//        최초 로그인 창 또는 MainFrame 띄우기
//        로그인 성공 후 CarGUI 호출

        public static void main(String[] args) {
        // 프로그램 시작 시 메인 프레임부터 띄움 (로고 + 상점명 + 개발자명)
        javax.swing.SwingUtilities.invokeLater(() -> {
            CarManager manager = new CarManager();
            new MainFrame(manager);  // MainFrame에서 LoginFrame(manager)로 연결됨
        });
    }
}
```

```java
package com.carshop.main;
```
패키지 선언

com.carshop.main이라는 이름의 폴더(패키지)에 이 Main 클래스가 속한다는 뜻이지.

Java 프로젝트에서는 코드가 논리적, 물리적으로 잘 정리되도록 패키지로 구분해.

예를 들어 이 패키지는 프로그램에서 '메인 진입점' 역할을 하는 클래스들을 모아놓는 곳이라고 생각하면 돼.

```java
import com.carshop.service.CarManager;
```
import는 다른 패키지에 있는 클래스를 사용하겠다는 선언이야.

여기서는 com.carshop.service 패키지 안에 있는 CarManager 클래스를 가져오는 거야.

CarManager는 차량 관련 데이터를 관리하고 조작하는 중요한 '비즈니스 로직' 클래스야.

즉, 입고, 판매, 검색, 저장 같은 기능을 담당하지.

```java
import com.carshop.ui.MainFrame;
```
마찬가지로 com.carshop.ui 패키지에 있는 MainFrame 클래스를 불러와.

이 클래스는 GUI, 즉 사용자와 상호작용하는 화면을 담당해.

아마 JFrame을 상속받아서 프로그램의 '메인 윈도우'를 구현하고 있을 거야.

```java
public class Main {
```
Main 클래스 선언이야.

자바 프로그램은 실행할 때 '시작점'이 필요한데, 보통 이 Main 클래스가 그 역할을 해.

public 키워드는 이 클래스가 외부에서 접근 가능하다는 뜻이지.

```java
public static void main(String[] args) {
```
여기서부터 프로그램이 '진짜' 시작되는 곳, 메인 메서드야.

public은 어디서든 호출 가능하다는 뜻이고,

static은 프로그램이 시작될 때 JVM이 '객체 생성 없이' 바로 호출할 수 있도록 해.

void는 리턴값이 없다는 뜻이고,

main은 자바 프로그램 진입점 메서드 이름, 무조건 이렇게 써야 돼.

String[] args는 실행 시 외부에서 입력받는 '문자열 배열' 매개변수인데,

보통 커맨드라인 인자 받는 용도야.

javax.swing.SwingUtilities.*invokeLater*(() -> {

이건 아주 중요한 코드야.

javax.swing.SwingUtilities.invokeLater() 메서드는 GUI 작업을 이벤트 디스패치 스레드(EDT)라는 별도의 스레드에서 실행하도록 예약하는 함수야.

invokeLater()가 호출되면 바로 실행하지 않고, EDT라는 큐에 실행 작업을 등록해두고, EDT가 하나씩 꺼내서 순서대로 실행해.

왜 그러냐면, Swing GUI 컴포넌트들은 오직 이 EDT 스레드에서만 안전하게 조작할 수 있기 때문이야.

만약 메인 스레드나 다른 스레드에서 UI를 건드리면 '깜빡임', '에러', '충돌'이 발생해.

그래서 프로그램이 GUI 작업을 할 때는 항상 이렇게 EDT에 작업을 맡기는 게 '정석'이야.

()->{ ... } 는 람다식인데, Runnable 인터페이스를 간결하게 구현한 거라고 생각하면 돼.

여기서 invokeLater 메서드의 인자로 넘긴 람다 안에 있는 코드가 나중에 EDT에서 실행될 거야.

CarManager manager = **new** CarManager();
여기서 new 키워드가 나오는데, '객체 생성'이라는 뜻이야.
자바 메모리는 크게 스택 영역과 힙 영역으로 나뉘는데,
new로 만든 객체들은 힙 영역에 저장돼.
힙 영역은 '동적 메모리'라 불리며, 프로그램 실행 중 필요할 때마다 생성되고 소멸되지.
여기서 CarManager 객체를 만든다는 건, '차량 관리에 필요한 모든 데이터와 기능을 이 객체가 책임지고 처리할 것'이라는 의미야.
이 manager라는 변수는 참조 변수로, 힙에 만들어진 CarManager 객체의 주소값을 저장해.
그러니까 이 변수로 객체 내부의 모든 변수와 메서드에 접근할 수 있다는 뜻이지.

new MainFrame(manager);
이 줄도 객체를 하나 만들었어. MainFrame이라는 GUI 창 객체야.
마찬가지로 new 키워드로 생성해서 힙에 만들어진다.
MainFrame 생성자의 인자로 manager 객체를 넘겨줬는데,
이는 MainFrame이 '차량 관리' 기능을 위해 CarManager를 사용할 수 있게 하기 위해서야.
보통 이런 구조는 MVC 패턴에서 Controller 역할을 하는 객체를 UI에 전달해 역할 분담을 명확히 하는 것과 같다고 생각하면 좋아.
MainFrame 내부에서는 LoginFrame 창을 띄워서 사용자 로그인 과정을 먼저 진행시킬 거야.
로그인에 성공하면 다시 CarGUI 창으로 넘어가서 차량 관리가 본격적으로 이루어진다.

});

이 중괄호는 람다식의 끝을 의미해.

위에서 invokeLater()에 넘긴 람다 함수가 여기서 끝나고,

invokeLater는 이 람다(=Runnable 객체)를 이벤트 큐에 등록해두게 되는 거야.

}

이 중괄호는 main 메서드가 끝나는 부분.

}

마지막 중괄호는 Main 클래스가 끝나는 부분이야.

데이터 모델부터 파악하기 — Car 클래스

```
package com.carshop.model;

import java.time.LocalDate;

public class Car {
//      역할: 자동차 한 대의 데이터 저장
//      해야 할 일:
//      자동차 정보 멤버변수 선언 (이름, 코드번호, 가격, 입고일자, 재고 수량, 이미지 경로)
//      Getter/Setter 메서드 작성
//      생성자 작성 (모든 필드를 초기화)
//      toString() 재정의 (GUI/콘솔 출력용)
//      저장/로드용 문자열 변환 메서드 (toDataLine(), fromDataLine()) 작성

                private String name;
                private int codeNumber;
                private long price;
                private LocalDate arrivalDate;
                private int stockQuantity;
                private String imagePath;

                public String getName() {
                        return name;
                }

                public void setName(String name) {
                        this.name = name;
                }

                public int getCodeNumber() {
                        return codeNumber;
                }

                public void setCodeNumber(int codeNumber) {
                        this.codeNumber = codeNumber;
                }

                public long getPrice() {
                        return price;
                }

                public void setPrice(long price) {
                        this.price = price;
                }

                public LocalDate getArrivalDate() {
                        return arrivalDate;
                }

                public void setArrivalDate(LocalDate arrivalDate) {
                        this.arrivalDate = arrivalDate;
                }

                public int getStockQuantity() {
                        return stockQuantity;
                }

                public void setStockQuantity(int stockQuantity) {
                        this.stockQuantity = stockQuantity;
                }

                public String getImagePath() {
                        return imagePath;
                }

                public void setImagePath(String imagePath) {
                        this.imagePath = imagePath;
                }

                public Car(String name, int codeNumber, long price, LocalDate arrivalDate, int
```

```
stockQuantity, String imagePath){
                        this.name = name;
                        this.codeNumber = codeNumber;
                        this.price = price;
                        this.arrivalDate = arrivalDate;
                        this.stockQuantity = stockQuantity;
                        this.imagePath = imagePath;


                }

                public String toString() {
                        return String.format("차종 : %s\n 코드번호 : %d\n 가격 : %d\n 입고일자 :%s\n
재고 : %d\n "
                                        + "이미지 경로 :%s\n", this.name, this.codeNumber, this.price,
this.arrivalDate, this.stockQuantity, this.imagePath);
                }

                public String toDataLine() {
                    return String.format("%s|%d|%d|%s|%d|%s",
                        name, codeNumber, price, arrivalDate.toString(), stockQuantity, imagePath);
                }

                public static Car fromDataLine(String line) {
                    String[] parts = line.split("\\|");
                    if (parts.length != 6) {
                        throw new IllegalArgumentException("잘못된 데이터 형식");
                    }

                    String name = parts[0];
                    int codeNumber = Integer.parseInt(parts[1]);
                    long price = Long.parseLong(parts[2]);
                    LocalDate arrivalDate = LocalDate.parse(parts[3]);   // yyyy-MM-dd
                    int stockQuantity = Integer.parseInt(parts[4]);
                    String imagePath = parts[5];

                    return new Car(name, codeNumber, price, arrivalDate, stockQuantity, imagePath);
                }
//                 ◇  생성자와 private 필드, 그리고 getter/setter 관계
//                1. private 필드로 캡슐화(정보 은닉)
//                클래스 내부 필드를 private으로 선언하면
//
//                외부에서 직접 접근 못 하게 막아서 데이터 보호 가능
//
//                2. 생성자로 초기화
//                생성자에서만 필드를 초기화해서,
//
//                객체가 올바른 상태로 태어나도록 보장
//
//                3. getter/setter로 간접 접근 제공
//                외부에서는 직접 필드를 건드리지 않고,
//
//                getter로 값을 읽고, setter로 값을 바꾸는 방법으로만 접근
//
//                이를 통해 필드 값 검증이나 변경 제한 등 로직 삽입 가능
}
```

```
package com.carshop.model;
```
이 줄은 Car 클래스가 com.carshop.model 패키지 안에 속한다는 선언이야.
model 패키지는 데이터 모델, 즉 프로그램이 다루는 '데이터 구조'를 정의하는 클래스를 모아놓는 곳이야.

```
import java.time.LocalDate;
```
LocalDate 클래스를 불러오는 구문이야.
LocalDate는 '날짜'를 표현하는 클래스인데, 연도-월-일만 다룰 때 사용해.
java.time 패키지는 날짜와 시간을 더 안전하고 편리하게 다룰 수 있게 Java 8부터 도입된 API야.

```
public class Car {
```
Car 클래스 선언.
자동차 한 대의 정보를 담는 '데이터 객체(Data Object)' 역할을 해.
public으로 선언해서 다른 패키지에서도 이 클래스를 쓸 수 있게 했어.

멤버 변수 선언 및 역할

```
private String name;
private int codeNumber;
private long price;
private LocalDate arrivalDate;
private int stockQuantity;
private String imagePath;
```
private 접근 제어자로 선언된 필드들이야.
이렇게 private으로 선언하면 외부에서 직접 접근할 수 없고, 오직 클래스 내부에서만 조작 가능해.
이것이 바로 캡슐화(정보 은닉) 기법으로, 데이터 보호에 매우 중요해.
각 변수 설명:
name : 차종 이름을 저장하는 문자열
codeNumber : 차종을 식별하는 고유 코드 번호 (정수형)
price : 차량 가격을 나타내는 긴 정수형 (long)
arrivalDate : 입고일자를 LocalDate 객체로 저장
stockQuantity : 현재 재고 수량 (몇 대가 남아있는지)
imagePath : 차량 이미지가 저장된 파일 경로 (문자열)

Getter / Setter 메서드

```
public String getName() {
        return name;
}

public void setName(String name) {
        this.name = name;
}

public int getCodeNumber() {
        return codeNumber;
}

public void setCodeNumber(int codeNumber) {
        this.codeNumber = codeNumber;
}

public long getPrice() {
        return price;
}

public void setPrice(long price) {
        this.price = price;
}

public LocalDate getArrivalDate() {
        return arrivalDate;
}

public void setArrivalDate(LocalDate arrivalDate) {
        this.arrivalDate = arrivalDate;
}

public int getStockQuantity() {
        return stockQuantity;
}

public void setStockQuantity(int stockQuantity) {
        this.stockQuantity = stockQuantity;
}

public String getImagePath() {
        return imagePath;
}
```

외부에서 private 필드에 직접 접근 못 하니까,

'간접 접근 통로'로 getter와 setter를 제공하는 거야.

getXxx()는 해당 필드의 값을 반환하고,

setXxx()는 필드 값을 바꾸는 역할을 해.

이 메서드들을 통해서만 필드 값을 읽거나 변경할 수 있지.

만약 데이터 무결성을 지키고 싶으면, setter 내부에 검증 로직을 넣을 수도 있어.

생성자

```
public Car(String name, int codeNumber, long price, LocalDate arrivalDate, int stockQuantity, String imagePath){
                        this.name = name;
                        this.codeNumber = codeNumber;
                        this.price = price;
                        this.arrivalDate = arrivalDate;
                        this.stockQuantity = stockQuantity;
                        this.imagePath = imagePath;


                }
```

이 생성자는 Car 객체를 만들 때, 필수 모든 정보를 한 번에 초기화하도록 만든 거야.

this.name = name; 이런 식으로 인자로 받은 값을 멤버 변수에 할당해.

생성자 덕분에 Car 객체는 언제나 올바른 상태로 태어나도록 보장받아.

new Car("Sonata", 1001, 25000000L, LocalDate.now(), 5, "images/sonata.png")처럼 쓸 수 있어.

toString() 메서드 재정의

```
public String toString() {
        return String.format("차종 : %s\n 코드번호 : %d\n 가격 : %d\n 입고일자 :%s\n 재고 : %d\n "
+ "이미지 경로 :%s\n", this.name, this.codeNumber, this.price, this.arrivalDate, this.stockQuantity,
this.imagePath);
}
```

toString()은 자바에서 객체를 문자열로 표현할 때 쓰는 메서드야.

기본 Object 클래스에서 상속받아 재정의한 거지.

GUI나 콘솔 출력 시 이 메서드를 호출하면, Car 객체의 상태를 사람이 읽기 좋은 형식으로 보여줘.

%s, %d는 문자열과 숫자 출력 포맷이고, String.format으로 깔끔하게 출력하도록 처리했어.

저장용 문자열 변환 메서드 (toDataLine)

```
public String toDataLine() {
        return String.format("%s|%d|%d|%s|%d|%s",
        name, codeNumber, price, arrivalDate.toString(), stockQuantity, imagePath);
}
```

이 메서드는 Car 객체의 데이터를 '한 줄 문자열'로 변환해.

각 데이터 필드를 | 문자로 구분해서 연결했어.

이렇게 하면 파일이나 DB에 저장하거나, 네트워크로 전송할 때 편리해.

arrivalDate.toString()은 LocalDate를 yyyy-MM-dd 문자열로 변환해주는 메서드야.

저장된 문자열을 객체로 복원하는 메서드 (fromDataLine)

```
public static Car fromDataLine(String line) {
                        String[] parts = line.split("\\|");
                        if (parts.length != 6) {
                            throw new IllegalArgumentException("잘못된 데이터 형식");
                        }

                        String name = parts[0];
                        int codeNumber = Integer.parseInt(parts[1]);
                        long price = Long.parseLong(parts[2]);
                        LocalDate arrivalDate = LocalDate.parse(parts[3]);  // yyyy-MM-dd
                        int stockQuantity = Integer.parseInt(parts[4]);
                        String imagePath = parts[5];

                        return new Car(name, codeNumber, price, arrivalDate, stockQuantity, imagePath);
                    }
```

fromDataLine은 static 메서드라 클래스 이름으로 호출 가능해.
저장된 문자열 한 줄을 받아서, 다시 Car 객체로 '복원'하는 역할을 해.
line.split("\\|") 으로 | 기준으로 문자열을 나누고,
각 필드를 알맞은 타입으로 변환(parse)해서 새 객체를 만든다.
만약 |로 나눈 결과가 6개가 아니면 데이터 형식 오류라 판단해서 예외를 던짐.
저장과 로드가 쌍으로 맞게 설계된 메서드야.

캡슐화와 객체지향 원칙 설명

```
// ◇ 생성자와 private 필드, 그리고 getter/setter 관계
// 1. private 필드로 캡슐화(정보 은닉)
// 클래스 내부 필드를 private으로 선언하면 외부에서 직접 접근 못 하게 막아서 데이터 보호 가능
//
// 2. 생성자로 초기화
// 생성자에서만 필드를 초기화해서, 객체가 올바른 상태로 태어나도록 보장
//
// 3. getter/setter로 간접 접근 제공
// 외부에서는 직접 필드를 건드리지 않고,
// getter로 값을 읽고, setter로 값을 바꾸는 방법으로만 접근
//
// 이를 통해 필드 값 검증이나 변경 제한 등 로직 삽입 가능
```

이 주석은 객체지향 프로그래밍(OOP)의 핵심 원칙 중 하나인 캡슐화를 설명하고 있어.
private 필드는 외부에서 직접 조작할 수 없도록 막아 객체 내부 상태가 임의로 바뀌는 것을 방지해.
getter와 setter는 필드를 안전하게 읽고 쓸 수 있도록 중간에 '통로' 역할을 해.
생성자는 객체 생성 시 필드 값을 반드시 초기화하도록 해서 '무결성'을 지켜줌.

Car 클래스는 '자동차 한 대' 정보를 구조화한 데이터 모델이야.
private 필드로 내부 정보를 은닉하고, getter/setter로 외부 접근을 제어함.
생성자를 통해 객체가 올바른 상태로 생성되도록 보장함.
toString()은 사람이 읽기 좋은 상태로 객체 내용을 출력하고,
toDataLine()과 fromDataLine()은 파일 저장/로드를 위한 문자열 변환 기능을 제공함.
LocalDate를 써서 입고일자를 명확하게 다루고,
예외처리를 해서 데이터 형식이 틀릴 때 적절히 대응할 수 있게 설계되어 있어.

비즈니스 로직 – CarManager 클래스

```
package com.carshop.service;
import com.carshop.model.Car;
import com.carshop.util.FileHandler;
import java.util.LinkedList;
import java.util.List;
import java.util.ListIterator;
import java.time.LocalDate;

public class CarManager implements BasicFunction{
//        역할: 자동차 리스트 관리 및 기능 구현
//        해야 할 일:
//        LinkedList<Car> 선언 및 초기화
//        addCar()에서 입고일 현재 날짜 자동 지정 후 리스트에 추가
//        sellCar()에서 코드번호로 찾아 재고 수량 1 감소, 재고 0시 삭제 혹은 알림 처리
//        viewStock()에서 전체 리스트 반환
//        saveData()에서 파일에 자동차 정보를 저장 (텍스트 포맷)
//        loadData()에서 파일 읽고 자동차 리스트 복원, 형식 에러 예외 처리
//        권장: 저장/로드 시 Car 클래스의 toDataLine() 및 fromDataLine() 활용
//        List<Car> carList = new LinkedList<>();
         private List<Car> carList = new LinkedList<>();

            @Override
            public void addCar(Car car) {
                car.setArrivalDate(LocalDate.now());  // 입고일 자동 설정
                carList.add(car);
            }

            @Override
            public void sellCar(String code) {
                ListIterator<Car> iterator = carList.listIterator();
                while (iterator.hasNext()) {
                    Car car = iterator.next();
                    if (String.valueOf(car.getCodeNumber()).equals(code)) {
                        if (car.getStockQuantity() > 0) {
                            car.setStockQuantity(car.getStockQuantity() - 1);
                            if (car.getStockQuantity() == 0) {
                                iterator.remove();  // 재고 없으면 리스트에서 제거
                                System.out.println("재고 0으로 삭제됨: " + car.getName());
                            }
                        } else {
                            System.out.println("재고가 없습니다.");
                        }
                        break;
                    }
                }
            }

            @Override
            public List<Car> viewStock() {
                return carList;
            }

            @Override
            public void saveData(String path) {
                FileHandler.saveToTextFile(carList, path);  // 🔄 FileHandler로 위임
            }

            @Override
            public void loadData(String path) {
                carList = FileHandler.loadFromTextFile(path);  // 🔄 불러온 List로 교체
            }
        }
```

```
package com.carshop.service;
```
이 코드는 현재 클래스가 com.carshop.service라는 패키지에 속해 있음을 나타냅니다.
이 패키지는 일반적으로 자동차 관련 비즈니스 로직을 처리하는 서비스 계층을 의미합니다.
JVM이 이 클래스를 로드할 때 이 경로를 기반으로 위치를 찾습니다.

```
import com.carshop.model.Car;
```
외부 클래스인 Car를 사용하기 위해 해당 클래스를 import 합니다.
이 Car 클래스는 com.carshop.model 패키지에 있으며, 자동차의 정보를 담고 있는
데이터 모델 클래스 (DTO 역할)입니다. 이 클래스가 없으면 CarManager에서 Car 타입을 사용할 수 없습니다
.

```
import com.carshop.util.FileHandler;
```
자동차 데이터를 파일로 저장하거나 불러오기 위해 FileHandler 유틸리티 클래스를 import 합니다.
즉, 입출력 기능(저장/로드)을 외부 클래스로 분리하여 책임을 분산시킨 구조입니다.

```
import java.util.LinkedList;
import java.util.List;
import java.util.ListIterator;
import java.time.LocalDate;
```
이 부분은 자바 표준 라이브러리의 여러 클래스를 import 합니다.
LinkedList: 자동차 데이터를 저장할 리스트로 사용할 자료구조
List: 인터페이스로서 LinkedList의 상위 타입
ListIterator: 리스트 내부 요소를 순회하면서 수정(삭제 포함)할 수 있는 반복자
LocalDate: 자동차 입고 날짜를 저장하기 위한 날짜 클래스 (시간 없이 날짜만 있음)

```
public class CarManager implements BasicFunction{
```
CarManager 클래스는 BasicFunction 인터페이스를 구현한 클래스입니다.
즉, addCar, sellCar, viewStock, saveData, loadData 메서드를 반드시 구현해야 합니다.
역할: 자동차 데이터들을 관리하고 비즈니스 로직(입고, 판매, 저장 등)을 담당합니다.

```
  private List<Car> carList = new LinkedList<>();
```
자동차 객체들을 저장할 리스트입니다.
인터페이스 List 타입으로 선언하고, 구현체로 LinkedList를 사용합니다.
이 구조를 통해 자동차를 순차적으로 저장, 검색, 삭제할 수 있습니다.
private 접근 제어자는 외부에서 이 리스트에 직접 접근하지 못하게 막아 정보 은닉(encapsulation)을 실현합니다.
메모리 관점에서 이 리스트는 CarManager 객체가 힙 메모리에 생성될 때 함께 생성됩니다

```
@Override
            public void addCar(Car car) {
```
BasicFunction 인터페이스의 메서드 addCar를 오버라이드한 부분입니다.
이 메서드는 외부에서 CarManager를 통해 호출되어 자동차 한 대를 시스템에 등록하는 역할을 합니다.
Car 객체를 파라미터로 받습니다. 이 객체는 이미 외부에서 생성된 상태일 것입니다.

```
  car.setArrivalDate(LocalDate.now());  // 입고일 자동 설정
```
car 객체의 입고일자 필드를 현재 날짜로 설정합니다.
이 코드는 addCar()가 호출되는 시점의 실시간 날짜를 기준으로 하기 때문에 입고 기록이 정확하게 남습니다.
이 작업은 데이터를 통일적으로 관리하기 위한 내부 정책에 가깝습니다.

```
carList.add(car);
```
방금 날짜를 설정한 Car 객체를 carList에 추가합니다.
이로써 리스트에 자동차 한 대가 입고 완료된 상태가 됩니다

```
 @Override
            public void sellCar(String code) {
```
코드번호를 기준으로 자동차를 검색해 재고를 감소시키는 메서드입니다.
String code는 외부에서 사용자가 입력하는 값이며, 내부적으로는 int 타입의 getCodeNumber()와 비교해야 하므로 변환이 필요합니다.

```
  ListIterator<Car> iterator = carList.listIterator();
```
ListIterator는 Iterator보다 기능이 많고, 순회 중 요소 삭제, 수정이 가능합니다.
이 반복자를 통해 리스트를 하나씩 순회하며 조건에 맞는 자동차를 찾습니다.

```
while (iterator.hasNext()) {
```
반복자의 다음 요소가 존재하는 동안 반복합니다.

```
Car car = iterator.next();
```
리스트에서 다음 자동차 객체를 가져옵니다.
이때 가져온 car는 힙 메모리에 존재하는 실제 Car 객체를 참조합니다.

```
if (String.valueOf(car.getCodeNumber()).equals(code)) {
```
Car 객체의 codeNumber 필드를 문자열로 변환한 뒤, 외부에서 받은 code와 비교합니다.
codeNumber는 int, code는 String이므로 형 변환이 필요합니다.

```
if (car.getStockQuantity() > 0) {
```
재고 수량이 1 이상일 경우에만 판매가 가능하다는 조건입니다.

```
car.setStockQuantity(car.getStockQuantity() - 1);
```
재고 수량을 1 줄입니다. 실제 Car 객체의 상태를 바꾸는 작업입니다.
이 작업은 자동차가 한 대 팔렸음을 의미합니다.

```
if (car.getStockQuantity() == 0) {
            iterator.remove();  // 재고 없으면 리스트에서 제거
            System.out.println("재고 0으로 삭제됨: " + car.getName());
 }
```
재고가 0이 되면 이 자동차 객체를 리스트에서 제거합니다.
iterator.remove()는 ListIterator를 사용해야만 가능한 삭제 방식입니다.
콘솔에 로그를 출력하여 어떤 차종이 삭제됐는지 알 수 있게 합니다.

```
} else {
            System.out.println("재고가 없습니다.");
}
```
재고 수량이 이미 0일 경우, 콘솔에 메시지를 출력하고 아무 작업도 하지 않습니다.

```
break;
```
코드번호는 유일하므로, 해당 자동차를 찾은 후에는 더 이상 반복할 필요 없이 종료합니다.

```
    @Override
    public List<Car> viewStock() {
            return carList;
    }
```
현재 등록되어 있는 자동차 리스트를 외부에 반환합니다.
직접적으로 리스트를 넘기므로, 읽기 전용으로 사용하는 것이 좋습니다.

```
@Override
        public void saveData(String path) {
            FileHandler.saveToTextFile(carList, path);  // 🔄 FileHandler로 위임
        }
```

자동차 리스트를 텍스트 파일로 저장합니다.
이 작업은 FileHandler 클래스의 saveToTextFile() 메서드에 위임합니다.
이때 리스트 내부의 각 Car 객체는 toDataLine()을 통해 텍스트로 변환될 것입니다

```
@Override
        public void loadData(String path) {
            carList = FileHandler.loadFromTextFile(path);  // 🔄 불러온 List로 교체
        }
```

저장된 텍스트 파일을 읽어 자동차 리스트를 복원합니다.
파일에서 읽어들인 자동차 정보들을 다시 Car 객체로 만든 뒤, 이 리스트로 교체합니다.
기존의 carList는 새로 읽어온 리스트로 완전히 대체됩니다.

BasicFunction 클래스

```
package com.carshop.service;
import java.util.List;
import com.carshop.model.Car;

public interface BasicFunction {
//        역할: 프로그램이 꼭 구현해야 할 기능 목록 선언
//        해야 할 일:
//        입고(addCar(Car car))
//        판매(sellCar(String code))
//        재고확인(viewStock())
//        저장(saveData(String path))
//        불러오기(loadData(String path))
//        주의: 메서드 선언만, 구현은 하지 않음

        void addCar(Car car);
        void sellCar(String code);
        List<Car> viewStock();
        void saveData(String path);
        void loadData(String path);
}
```

```
package com.carshop.service;
```

이 클래스(인터페이스)는 com.carshop.service 패키지에 속해 있어.
이 패키지는 비즈니스 로직, 즉 자동차 관리와 관련된 서비스 기능을 담당하는 위치야.
이후 이 인터페이스를 구현한 클래스(CarManager)도 이 패키지 안에 들어 있어야 구조가 깔끔해.

```
import java.util.List;
```

List는 Java의 대표적인 인터페이스 기반의 컬렉션 타입.
이 코드는 viewStock() 메서드가 List<Car>를 반환하기 때문에 필요해.

```
import com.carshop.model.Car;
```

자동차 객체(Car)를 다루기 위해 모델 패키지에서 Car 클래스를 import 함.
즉, 이 인터페이스는 자동차(Car)의 입고, 판매, 조회 등 다양한 조작을 수행해야 하므로 반드시 이 클래스가 필요함.

```
public interface BasicFunction {
```

interface: 인터페이스는 기능의 뼈대만 정의하고, 실제 구현은 하지 않아.
public: 다른 패키지에서도 이 인터페이스를 구현할 수 있게 공개(public)되어 있음.
의미: 이 인터페이스는 "자동차 거래 시스템에서 반드시 구현해야 할 핵심 기능"들의 표준 계약서 역할을 함.
나중에 CarManager 클래스에서 이 인터페이스를 implements 하면서 모든 메서드를 구현하게 됨.

```
//        역할: 프로그램이 꼭 구현해야 할 기능 목록 선언
//        해야 할 일:
//        입고(addCar(Car car))
//        판매(sellCar(String code))
//        재고확인(viewStock())
//        저장(saveData(String path))
//        불러오기(loadData(String path))
//        주의: 메서드 선언만, 구현은 하지 않음
```

이 주석은 매우 중요함.
이 인터페이스의 목적과 필수 구현 기능을 명확히 문서화함.
유지보수나 협업 시 이 파일만 보면 전체 기능의 윤곽을 알 수 있음.

```
void addCar(Car car);
```
목적: 새로운 자동차를 입고함 (즉, 시스템에 등록함).
Car 객체를 매개변수로 받아서 내부 재고 리스트에 추가하는 기능을 요구.
구현 클래스에서 내부적으로 List<Car> stock 같은 컬렉션에 add(car) 하게 될 것.
호출 위치 예: CarGUI에서 입고 버튼을 누르면 → CarManager.addCar(...) 호출됨.

```
void sellCar(String code);
```
목적: 자동차를 판매함 (즉, 재고에서 제거).
code는 자동차의 고유 식별자 (예: 차량 코드)임.
실제 구현에서는 List<Car>에서 해당 코드와 일치하는 객체를 찾아 remove() 해야 함.
호출 위치 예: CarGUI에서 판매 버튼을 누르면 CarManager.sellCar("K3-002") 호출.

```
List<Car> viewStock();
```
목적: 현재 재고로 남아 있는 모든 자동차를 리스트 형태로 반환.
반환 타입이 List<Car> 이므로, GUI 등에서 테이블로 재고를 표시할 수 있음.
예: CarTableModel 또는 MainFrame에서 이 메서드를 통해 테이블 데이터 구성.

```
void saveData(String path);
```
목적: 현재 재고 정보를 지정된 파일 경로(path)에 저장함.
내부 구현에서는 FileHandler.saveToTextFile(List<Car>, path) 같은 유틸을 호출하게 됨.
경로 예: c:/car/data.txt

```
void loadData(String path);
```

목적: 지정된 파일 경로로부터 자동차 데이터를 불러와서 시스템 재고로 세팅.
역시 FileHandler.loadFromTextFile(path) 같은 유틸을 호출하고, 그 결과를 재고 리스트에 대입함.

```
}
```
인터페이스 종료.

메모리 구조 및 호출 흐름 요약

요소	위치	설명
interface 정의	메서드 영역 (Method Area)	실행 시 인터페이스 구조 정보만 저장
실제 구현 클래스	예: CarManager implements BasicFunction	
메서드 호출	CarManager 인스턴스를 통해 호출됨	

실제 사용 흐름 (예시)
Main.java → CarManager manager = new CarManager();
CarManager는 implements BasicFunction 으로 모든 기능 구현
MainFrame, CarGUI 등에서 manager.addCar(...), manager.viewStock() 등 호출
이 호출들은 결국 인터페이스로부터 요구된 메서드들을 기반으로 동작

요약 정리

항목	내용
클래스 타입	interface (인터페이스)
이름	BasicFunction
책임	자동차 관리 시스템이 구현해야 할 필수 기능 목록 제시
구현 위치	CarManager 클래스가 이 인터페이스를 implements 함
기능 종류	addCar, sellCar, viewStock, saveData, loadData
메모리	Method Area에 정의 정보만 존재
실제 사용	CarManager를 통해 호출되어 기능이 실행됨

CarGUI 클래스

```
package com.carshop.ui;
import com.carshop.model.Car;
import com.carshop.service.CarManager;
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.io.File;
import java.util.List;
public class CarGUI extends JFrame {
//        역할: 실제 자동차 재고 관리 UI
//        해야 할 일:
//        자동차 목록 표시 (JTable 또는 JList 추천)
//        자동차 선택 시 상세정보 + 이미지 미리보기 영역 구현
//        입고, 판매, 저장, 불러오기 버튼과 이벤트 연결 (CarManager 호출)
//        이미지 파일 대화상자 통해 입고 시 이미지 선택 기능 구현
//        UI 업데이트 및 예외 처리
            private CarManager manager;
            private JTable table;
            private DefaultTableModel tableModel;
            private JLabel imageLabel;
            private JTextArea detailArea;

            public CarGUI(CarManager manager) {
                this.manager = manager;
                setTitle("자동차 재고 관리 시스템");
                setSize(800, 600);
                setLocationRelativeTo(null);
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                setLayout(new BorderLayout());

                // 테이블 모델 설정
                String[] columnNames = {"이름", "코드번호", "가격", "입고일자", "재고수량"};
                tableModel = new DefaultTableModel(columnNames, 0);
                table = new JTable(tableModel);
                table.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
                JScrollPane tableScroll = new JScrollPane(table);

                // 오른쪽 상세보기
                JPanel rightPanel = new JPanel(new BorderLayout());
                imageLabel = new JLabel("이미지 미리보기", SwingConstants.CENTER);
                imageLabel.setPreferredSize(new Dimension(200, 150));
                detailArea = new JTextArea(6, 20);
                detailArea.setEditable(false);
                rightPanel.add(imageLabel, BorderLayout.NORTH);
                rightPanel.add(new JScrollPane(detailArea), BorderLayout.CENTER);

                // 하단 버튼 패널
                JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));
                JButton addBtn = new JButton("입고");
                JButton sellBtn = new JButton("판매");
                JButton saveBtn = new JButton("저장");
                JButton loadBtn = new JButton("불러오기");

                buttonPanel.add(addBtn);
                buttonPanel.add(sellBtn);
                buttonPanel.add(saveBtn);
                buttonPanel.add(loadBtn);

                // 컴포넌트 배치
                add(tableScroll, BorderLayout.CENTER);
                add(rightPanel, BorderLayout.EAST);
                add(buttonPanel, BorderLayout.SOUTH);

                // 이벤트 연결
                addBtn.addActionListener(e -> openCarInputDialog());
                sellBtn.addActionListener(e -> sellSelectedCar());
                saveBtn.addActionListener(e -> saveData());
                loadBtn.addActionListener(e -> loadData());

                table.getSelectionModel().addListSelectionListener(e -> showSelectedCarDetail());
```

```
        refreshTable(); // 초기 데이터 표시
        setVisible(true);
    }

    private void refreshTable() {
        tableModel.setRowCount(0);
        for (Car car : manager.viewStock()) {
            tableModel.addRow(new Object[]{
                    car.getName(), car.getCodeNumber(), car.getPrice(),
                    car.getArrivalDate(), car.getStockQuantity()
            });
        }
    }

    private void showSelectedCarDetail() {
        int row = table.getSelectedRow();
        if (row == -1) return;
        Car car = manager.viewStock().get(row);
        detailArea.setText(car.toString());

        // 이미지 표시
        File imgFile = new File(car.getImagePath());
        if (imgFile.exists()) {
            ImageIcon icon = new ImageIcon(car.getImagePath());
            Image scaled = icon.getImage().getScaledInstance(200, 150, Image.SCALE_SMOOTH);
            imageLabel.setIcon(new ImageIcon(scaled));
            imageLabel.setText("");
        } else {
            imageLabel.setIcon(null);
            imageLabel.setText("이미지 없음");
        }
    }

    private void openCarInputDialog() {
        CarInputDialog dialog = new CarInputDialog(this, manager);
        dialog.setVisible(true);
        refreshTable();
    }

    private void sellSelectedCar() {
        int row = table.getSelectedRow();
        if (row == -1) {
            JOptionPane.showMessageDialog(this, "판매할 차를 선택하세요.");
            return;
        }
        String code = String.valueOf(table.getValueAt(row, 1));
        manager.sellCar(code);
        refreshTable();
    }

    private void saveData() {
        JFileChooser chooser = new JFileChooser();
        if (chooser.showSaveDialog(this) == JFileChooser.APPROVE_OPTION) {
            manager.saveData(chooser.getSelectedFile().getPath());
        }
    }

    private void loadData() {
        JFileChooser chooser = new JFileChooser();
        if (chooser.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {
            manager.loadData(chooser.getSelectedFile().getPath());
            refreshTable();
        }
    }
}
```

1. 패키지 선언 및 import 구문

```
package com.carshop.ui;
```

이 클래스는 com.carshop.ui 패키지에 포함되어 있음. 즉, UI 계층(View)을 담당함.

```
import com.carshop.model.Car;
import com.carshop.service.CarManager;
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.io.File;
import java.util.List;
```

Car: 자동차 정보를 나타내는 Model 객체
CarManager: 자동차 재고를 관리하는 서비스 계층 클래스
javax.swing.*: 버튼, 레이블, 프레임 등 UI 컴포넌트
javax.swing.table.DefaultTableModel: 테이블 데이터 모델
java.awt.*: 레이아웃, 이벤트 처리 등 AWT 컴포넌트
java.io.File: 이미지 경로 확인용
java.util.List: 재고 목록

2. 클래스 정의 및 필드

```
public class CarGUI extends JFrame {
```

CarGUI 클래스는 JFrame을 상속받음 → 하나의 메인 창을 생성할 수 있는 GUI 컴포넌트

```
    private CarManager manager;
    private JTable table;
    private DefaultTableModel tableModel;
    private JLabel imageLabel;
    private JTextArea detailArea;
```

CarManager manager: Controller 역할, 내부적으로 List<Car>를 관리
JTable table: 재고 목록을 표시하는 UI 컴포넌트
DefaultTableModel: JTable에 표시할 데이터 모델
JLabel imageLabel: 이미지 미리보기용
JTextArea detailArea: 자동차 상세정보 출력

3. 생성자: UI 생성 및 이벤트 연결

```
public CarGUI(CarManager manager) {
```

CarGUI 인스턴스를 생성할 때 외부에서 CarManager 인스턴스를 주입함 (의존성 주입)

```
  this.manager = manager;
```

주입받은 CarManager 저장

```
    setTitle("자동차 재고 관리 시스템");
    setSize(800, 600);
    setLocationRelativeTo(null);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLayout(new BorderLayout());
```

창 타이틀 설정
크기 800x600
화면 가운데 배치
종료 버튼 클릭 시 프로그램 종료
BorderLayout: 중앙, 동쪽, 남쪽 등 위치에 따라 컴포넌트 배치

4. 테이블 설정

```
// 테이블 모델 설정
                String[] columnNames = {"이름", "코드번호", "가격", "입고일자", "재고수량"};
                tableModel = new DefaultTableModel(columnNames, 0);
                table = new JTable(tableModel);
                table.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
                JScrollPane tableScroll = new JScrollPane(table);
```

열 제목 정의 후 테이블 모델 생성
JTable에 모델 연결
한 줄만 선택 가능하도록 설정
스크롤 가능하도록 JScrollPane으로 감쌈

5. 오른쪽 패널 구성 (상세정보, 이미지)

```
// 오른쪽 상세보기
                JPanel rightPanel = new JPanel(new BorderLayout());
                imageLabel = new JLabel("이미지 미리보기", SwingConstants.CENTER);
                imageLabel.setPreferredSize(new Dimension(200, 150));
                detailArea = new JTextArea(6, 20);
                detailArea.setEditable(false);
                rightPanel.add(imageLabel, BorderLayout.NORTH);
                rightPanel.add(new JScrollPane(detailArea), BorderLayout.CENTER);
```

오른쪽에 이미지 + 상세정보 표시
JLabel에 이미지 미리보기 텍스트 기본 표시
detailArea: 자동차 정보 문자열 출력 (Car.toString())
BorderLayout을 이용해 위: 이미지, 중앙: 텍스트

6. 하단 버튼 구성

```
// 하단 버튼 패널
                JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));
                JButton addBtn = new JButton("입고");
                JButton sellBtn = new JButton("판매");
                JButton saveBtn = new JButton("저장");
                JButton loadBtn = new JButton("불러오기");

                buttonPanel.add(addBtn);
                buttonPanel.add(sellBtn);
                buttonPanel.add(saveBtn);
                buttonPanel.add(loadBtn);
```

FlowLayout을 사용해 하단에 버튼 4개 정렬
각각 입고, 판매, 저장, 불러오기 역할

7. 전체 GUI 배치

```
// 컴포넌트 배치
                add(tableScroll, BorderLayout.CENTER);
                add(rightPanel, BorderLayout.EAST);
                add(buttonPanel, BorderLayout.SOUTH);
```

중앙에 재고 테이블
오른쪽에 상세정보 + 이미지
아래쪽에 버튼 4개

8. 버튼 및 이벤트 연결

```
// 이벤트 연결
                addBtn.addActionListener(e -> openCarInputDialog());
                sellBtn.addActionListener(e -> sellSelectedCar());
                saveBtn.addActionListener(e -> saveData());
                loadBtn.addActionListener(e -> loadData());

                table.getSelectionModel().addListSelectionListener(e -> showSelectedCarDetail());
```

버튼마다 람다식으로 이벤트 연결
테이블에서 항목 선택 시 상세정보 표시

9. 초기 테이블 새로고침 후 화면 표시

```
refreshTable(); // 초기 데이터 표
setVisible(true);
```

창을 보이도록 설정 (setVisible)
manager.viewStock()을 기반으로 테이블 초기화

내부 메서드 분석
refreshTable()

```
private void refreshTable() {
                tableModel.setRowCount(0);
                for (Car car : manager.viewStock()) {
                    tableModel.addRow(new Object[]{
                            car.getName(), car.getCodeNumber(), car.getPrice(),
                            car.getArrivalDate(), car.getStockQuantity()
                    });
                }
            }
```

테이블 데이터 초기화 후 manager.viewStock()의 리스트로 다시 채움

showSelectedCarDetail()

```
private void showSelectedCarDetail() {
                int row = table.getSelectedRow();
                if (row == -1) return;
                Car car = manager.viewStock().get(row);
                detailArea.setText(car.toString());
```

선택된 row에서 Car 객체 꺼냄 → Car.toString()으로 텍스트 영역에 출력

```
// 이미지 표시
                File imgFile = new File(car.getImagePath());
                if (imgFile.exists()) {
                    ImageIcon icon = new ImageIcon(car.getImagePath());
                    Image scaled = icon.getImage().getScaledInstance(200, 150, Image.SCALE_SMOOTH);
                    imageLabel.setIcon(new ImageIcon(scaled));
                    imageLabel.setText("");
                } else {
                    imageLabel.setIcon(null);
                    imageLabel.setText("이미지 없음");
                }
            }
```

이미지가 존재하면 미리보기 출력, 없으면 "이미지 없음"

openCarInputDialog()

```
private void openCarInputDialog() {
                CarInputDialog dialog = new CarInputDialog(this, manager);
                dialog.setVisible(true);
                refreshTable();
        }
```

새 입고 창을 열고 닫힌 후 테이블을 새로고침

sellSelectedCar()

```
private void sellSelectedCar() {
                int row = table.getSelectedRow();
                if (row == -1) {
                    JOptionPane.showMessageDialog(this, "판매할 차를 선택하세요.");
                    return;
                }
                String code = String.valueOf(table.getValueAt(row, 1));
                manager.sellCar(code);
                refreshTable();
        }
```

선택된 차량의 코드번호를 이용해 manager.sellCar() 호출

saveData() & loadData()

```
 private void saveData() {
                JFileChooser chooser = new JFileChooser();
                if (chooser.showSaveDialog(this) == JFileChooser.APPROVE_OPTION) {
                    manager.saveData(chooser.getSelectedFile().getPath());
                }
        }

        private void loadData() {
            JFileChooser chooser = new JFileChooser();
            if (chooser.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {
                manager.loadData(chooser.getSelectedFile().getPath());
                refreshTable();
            }
        }
```

파일 대화상자를 열고 선택된 경로로 저장/불러오기 실행
불러오기 후 refreshTable()로 UI 동기화

CarInputDialog 클래스

```
package com.carshop.ui;

import com.carshop.model.Car;
import com.carshop.service.CarManager;

import javax.swing.*;
import javax.swing.filechooser.FileNameExtensionFilter;
import javax.swing.text.NumberFormatter;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.io.File;
import java.text.NumberFormat;
import java.time.LocalDate;

public class CarInputDialog extends JDialog {
//      역할: 자동차 정보 입력 창
//      해야 할 일:
//      자동차 정보 입력 필드(이름, 코드번호, 가격, 재고 수, 이미지 선택 등)
//      이미지 파일 선택 버튼과 JFileChooser 연결
//      등록 버튼 클릭 시 Car 객체 생성 후 CarManager의 addCar() 호출
//      입력값 검증 (예: 빈칸, 숫자 형식 등)
            private JTextField nameField;
            private JFormattedTextField codeField, priceField, stockField;
            private JTextField imageField;
            private JButton imageButton, registerButton;
            private CarManager manager;

            public CarInputDialog(JFrame parent, CarManager manager) {
                super(parent, "자동차 입고", true);
                this.manager = manager;

                setLayout(new GridLayout(6, 2, 10, 10));
                setSize(400, 300);
                setLocationRelativeTo(parent);

                // 숫자 포맷터 설정 (음수 입력 불가)
                NumberFormat intFormat = NumberFormat.getIntegerInstance();
                intFormat.setGroupingUsed(false);
                NumberFormatter numberFormatter = new NumberFormatter(intFormat);
                numberFormatter.setValueClass(Integer.class);
                numberFormatter.setAllowsInvalid(false);
                numberFormatter.setMinimum(0);

                NumberFormat longFormat = NumberFormat.getIntegerInstance();
                longFormat.setGroupingUsed(false);
                NumberFormatter longFormatter = new NumberFormatter(longFormat);
                longFormatter.setValueClass(Long.class);
                longFormatter.setAllowsInvalid(false);
                longFormatter.setMinimum(0L);

                // 입력 필드 초기화
                nameField = new JTextField();
                codeField = new JFormattedTextField(numberFormatter);
                priceField = new JFormattedTextField(longFormatter);
                stockField = new JFormattedTextField(numberFormatter);
                imageField = new JTextField();
                imageField.setEditable(false);

                imageButton = new JButton("이미지 선택");
                registerButton = new JButton("등록");

                // 레이아웃
                add(new JLabel("차 이름:"));
                add(nameField);

                add(new JLabel("코드번호:"));
                add(codeField);

                add(new JLabel("가격:"));
                add(priceField);
```

```
        add(new JLabel("재고 수량:"));
        add(stockField);

        add(new JLabel("이미지 경로:"));
        add(imageField);

        add(imageButton);
        add(registerButton);

        // 이미지 선택 버튼 액션 리스너
        imageButton.addActionListener(this::chooseImage);

        // 등록 버튼 액션 리스너
        registerButton.addActionListener(this::registerCar);
    }

    private void chooseImage(ActionEvent e) {
        JFileChooser chooser = new JFileChooser();
        chooser.setDialogTitle("이미지 선택");
        chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);

        // 이미지 확장자 필터 추가
        FileNameExtensionFilter filter = new FileNameExtensionFilter(
                "이미지 파일 (JPG, PNG)", "jpg", "jpeg", "png");
        chooser.setFileFilter(filter);

        if (chooser.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {
            File selected = chooser.getSelectedFile();
            imageField.setText(selected.getAbsolutePath());
        }
    }

    private void registerCar(ActionEvent e) {
        try {
            String name = nameField.getText().trim();
            int code = ((Number) codeField.getValue()).intValue();
            long price = ((Number) priceField.getValue()).longValue();
            int stock = ((Number) stockField.getValue()).intValue();
            String imagePath = imageField.getText().trim();

            if (name.isEmpty() || imagePath.isEmpty()) {
                throw new IllegalArgumentException("빈 칸 없이 입력해주세요.");
            }

            Car car = new Car(name, code, price, LocalDate.now(), stock, imagePath);
            manager.addCar(car);

            JOptionPane.showMessageDialog(this, "자동차 입고 완료!");
            dispose();
        } catch (NullPointerException ex) {
            JOptionPane.showMessageDialog(this, "모든 숫자 필드를 올바르게 입력해주세요.");
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(this, "입력 오류: " + ex.getMessage());
        }
    }
}
```

```
package com.carshop.ui;
```

역할: CarInputDialog 클래스는 com.carshop.ui 패키지에 포함되어 있음을 나타냄.
com.carshop.ui는 사용자 인터페이스(UI) 요소들을 담는 패키지이므로, 이 클래스도 사용자와 상호작용하는 UI 구성 요소로 설계된 것임.

```
import com.carshop.model.Car;
import com.carshop.service.CarManager;
```

외부 패키지에서 두 클래스를 가져옴.
Car: 자동차 한 대를 표현하는 모델 클래스 (도메인 객체)
CarManager: 자동차 목록 관리, 재고 추가/삭제 등 핵심 비즈니스 로직을 담당하는 클래스
여기서 CarInputDialog는 CarManager를 이용해 UI → 비즈니스 로직으로 데이터 전달하는 역할을 함.

```
import javax.swing.*;
import javax.swing.filechooser.FileNameExtensionFilter;
import javax.swing.text.NumberFormatter;
```

javax.swing.*: Java의 GUI 컴포넌트들 (버튼, 텍스트 필드, 다이얼로그 등)
FileNameExtensionFilter: JFileChooser에서 확장자 필터링할 때 사용 (예: 이미지 파일만 선택)
NumberFormatter: 숫자 전용 필드를 다룰 때 유효성 및 포맷을 지정하기 위해 사용

```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.io.File;
import java.text.NumberFormat;
import java.time.LocalDate;
```

AWT 및 유틸 패키지 불러오기
주요 목적:
GridLayout, ActionEvent 등 UI 이벤트 및 레이아웃 구성
File: 이미지 경로 저장
NumberFormat: 숫자 입력 필드 포맷 지정
LocalDate: 오늘 날짜 가져와서 Car 객체 생성 시 등록일로 사용

클래스 선언부

```
public class CarInputDialog extends JDialog {
```

CarInputDialog는 JDialog를 상속받은 모달 다이얼로그.
모달이란? → 이 다이얼로그가 떠 있으면 다른 창 조작 불가.
역할: 사용자로부터 자동차 정보를 입력받아 CarManager에 전달해 재고를 추가하는 용도.

클래스 필드

```
private JTextField nameField;
private JFormattedTextField codeField, priceField, stockField;
private JTextField imageField;
private JButton imageButton, registerButton;
private CarManager manager;
```

JTextField nameField: 자동차 이름 입력
JFormattedTextField codeField, priceField, stockField: 각각 코드번호, 가격, 재고 입력용 (숫자만 허용)
JTextField imageField: 이미지 경로 표시 (수정 불가)
imageButton: 이미지 선택 열기
registerButton: 등록 버튼
CarManager manager: 입력된 정보를 넘겨줄 핵심 서비스 객체 (입고 처리 담당)
→ 즉, 이 클래스는 UI 구성 필드 + 비즈니스 로직 연결 필드(CarManager)로 나뉨.

생성자

```
public CarInputDialog(JFrame parent, CarManager manager) {
super(parent, "자동차 입고", true);
this.manager = manager;
```

CarInputDialog 생성자에서 JDialog의 super() 호출
parent: 부모 프레임을 지정 (모달로 띄우기 위함)
"자동차 입고": 타이틀
true: 모달 모드 설정
this.manager = manager: 외부에서 주입한 CarManager를 내부에서 사용하기 위해 저장
→ 여기까지가 "다이얼로그 초기 세팅"에 해당

```
setLayout(new GridLayout(6, 2, 10, 10));
                    setSize(400, 300);
                    setLocationRelativeTo(parent);
```

GridLayout(6, 2, 10, 10):
6행 2열
컴포넌트 간 간격은 가로/세로 10픽셀
setSize: 다이얼로그의 너비 400px, 높이 300px
setLocationRelativeTo(parent): 부모창 중앙에 이 다이얼로그를 위치시킴

숫자 입력 필드용 포맷터 설정

```
NumberFormat intFormat = NumberFormat.getIntegerInstance();
                    intFormat.setGroupingUsed(false);
                    NumberFormatter numberFormatter = new NumberFormatter(intFormat);
                    numberFormatter.setValueClass(Integer.class);
                    numberFormatter.setAllowsInvalid(false);
                    numberFormatter.setMinimum(0);
```

intFormat: 숫자 포맷을 설정. 쉼표(,) 없이 정수만 입력 받음
numberFormatter:
Integer.class: 정수 입력만 가능
setAllowsInvalid(false): 잘못된 입력(예: 문자, 음수) 즉시 거부
setMinimum(0): 0 이상의 값만 허용 (음수 입력 불가)
→ 코드번호와 재고에 사용됨

```
NumberFormat longFormat = NumberFormat.getIntegerInstance();
                    longFormat.setGroupingUsed(false);
                    NumberFormatter longFormatter = new NumberFormatter(longFormat);
                    longFormatter.setValueClass(Long.class);
                    longFormatter.setAllowsInvalid(false);
                    longFormatter.setMinimum(0L);
```

longFormatter: 가격은 long 타입이므로 별도로 설정
원리는 동일하지만, 더 큰 숫자 허용
→ 가격 입력 필드에 사용됨

```
                    imageButton = new JButton("이미지 선택");
                    registerButton = new JButton("등록");
```

imageButton: 클릭 시 파일 탐색기(JFileChooser)가 뜨고 이미지 파일을 선택하게 됨.
registerButton: 모든 값이 올바르게 입력되었는지 확인 후, Car 객체 생성 → CarManager.addCar() 호출

입력 필드 및 버튼 초기화

```
// 입력 필드 초기화
                nameField = new JTextField();
                codeField = new JFormattedTextField(numberFormatter);
                priceField = new JFormattedTextField(longFormatter);
                stockField = new JFormattedTextField(numberFormatter);
                imageField = new JTextField();
                imageField.setEditable(false);
```

메모리 구조 및 역할 설명:
nameField = new JTextField();
자동차 이름을 입력하는 일반 텍스트 필드.
사용자 입력값은 String 타입으로 저장됨.
new 연산자 사용 → 힙 영역에 JTextField 객체 생성됨.
codeField = new JFormattedTextField(numberFormatter);
자동차 코드번호 입력 필드.
numberFormatter를 적용하여 정수만 입력 가능, 실수/문자 불허.
priceField = new JFormattedTextField(longFormatter);
가격 입력 필드. longFormatter 적용 → long 타입 값만 입력 가능
stockField = new JFormattedTextField(numberFormatter);
재고 수량 입력 필드. 역시 정수형 숫자만 입력 가능.
imageField = new JTextField();
이미지 파일의 절대 경로를 표시하는 필드.
사용자가 직접 입력하지 못하도록 .setEditable(false) 설정됨 → 읽기 전용으로 사용

UI 컴포넌트 레이아웃 구성

```
// 레이아웃
                add(new JLabel("차 이름:"));
                add(nameField);

                add(new JLabel("코드번호:"));
                add(codeField);

                add(new JLabel("가격:"));
                add(priceField);

                add(new JLabel("재고 수량:"));
                add(stockField);

                add(new JLabel("이미지 경로:"));
                add(imageField);

                add(imageButton);
                add(registerButton);
```

설명:
setLayout(new GridLayout(6, 2))**에 따라 총 12칸이 있음 → 각 줄에 Label + 입력 필드 구성
실제 배치:

열 1 (add())	열 2 (add())
"차 이름" Label	nameField
"코드번호" Label	codeField
"가격" Label	priceField
"재고 수량" Label	stockField
"이미지 경로" Label	imageField
imageButton	registerButton

순서대로 add() 호출되며 GridLayout에 따라 왼쪽 → 오른쪽 → 다음 줄 순서로 채워짐

버튼 이벤트 리스너 연결

```
// 이미지 선택 버튼 액션 리스너
                imageButton.addActionListener(this::chooseImage);

                // 등록 버튼 액션 리스너
                registerButton.addActionListener(this::registerCar);
```

imageButton: 클릭하면 chooseImage(ActionEvent e) 메서드 호출됨 → JFileChooser 실행
registerButton: 클릭하면 registerCar(ActionEvent e) 메서드 호출 → 입력값 검증 + Car 객체 생성
여기서 this::chooseImage 구문은 Java 8부터 도입된 메서드 참조 방식이며,
내부적으로 ActionListener의 actionPerformed(ActionEvent e)를 구현한 것으로 인식됨.

chooseImage(ActionEvent e) 메서드

```
    private void chooseImage(ActionEvent e) {
```

private: 이 메서드는 CarInputDialog 내부에서만 호출 가능
void: 반환값 없음
chooseImage는 imageButton.addActionListener(this::chooseImage)에서 호출됨
즉, 사용자가 "이미지 선택" 버튼을 클릭하면 이 메서드가 실행됨
ActionEvent e: 이벤트 정보 객체 (어떤 버튼이 눌렸는지 등의 정보 포함)

```
JFileChooser chooser = new JFileChooser();
```

JFileChooser: Swing에서 제공하는 파일 선택 대화상자 컴포넌트
chooser: 힙 메모리에 새로 생성됨
이후 .showOpenDialog()를 호출하면 실제 파일 선택 UI가 뜸

```
    chooser.setDialogTitle("이미지 선택");
```

다이얼로그 창의 타이틀 텍스트를 설정함
→ 사용자에게 "이미지 선택"이라는 제목이 보임

```
chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
```

파일만 선택할 수 있도록 설정 (디렉토리 선택은 불가)
상수 JFileChooser.FILES_ONLY는 정수 값으로 내부적으로 설정됨 (1)

```
FileNameExtensionFilter filter = new FileNameExtensionFilter(
                        "이미지 파일 (JPG, PNG)", "jpg", "jpeg", "png");
```

이미지 확장자 필터 객체 생성
"이미지 파일 (JPG, PNG)"는 사용자에게 표시될 설명
"jpg", "jpeg", "png"만 선택 가능하도록 제한

```
chooser.setFileFilter(filter);
```

위에서 만든 filter를 chooser에 적용
사용자는 JPEG, PNG 파일만 볼 수 있음

```
if (chooser.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {
```

파일 선택 대화상자를 띄우고 결과값을 확인
this: 현재 JDialog를 기준으로 다이얼로그가 뜸
showOpenDialog()는 모달로 실행됨 (사용자가 선택하거나 취소할 때까지 코드 진행 중단)
리턴값이 APPROVE_OPTION이면 = 사용자가 파일 선택을 완료했음을 의미

```
File selected = chooser.getSelectedFile();
```
사용자가 선택한 파일을 가져옴
File 객체는 선택된 이미지의 메타정보를 담고 있음 (경로, 이름, 확장자 등)

```
imageField.setText(selected.getAbsolutePath());
```
imageField (이미지 경로 표시용 JTextField)에 선택된 이미지의 절대경로 입력
imageField는 .setEditable(false)로 되어 있어서 직접 수정 불가, 보기 전용

registerCar(ActionEvent e) 메서드

```
private void registerCar(ActionEvent e) {
```
이 메서드는 "등록" 버튼 클릭 시 실행됨
registerButton.addActionListener(this::registerCar)에서 연결
역할: 사용자 입력값 → 유효성 검사 → Car 객체 생성 → CarManager 등록

```
try {
```
예외 처리 시작
내부에서 NullPointerException, IllegalArgumentException 발생 가능성이 있음

```
String name = nameField.getText().trim();
```
nameField에 입력된 문자열을 가져옴
trim()은 앞뒤 공백 제거
→ 자동차 이름 값

```
int code = ((Number) codeField.getValue()).intValue();
```
codeField: JFormattedTextField → getValue()는 Number 타입 리턴
intValue()로 변환
→ 자동차의 코드번호 (정수)

```
long price = ((Number) priceField.getValue()).longValue();
```
가격 필드에서 값 추출, long으로 캐스팅

```
int stock = ((Number) stockField.getValue()).intValue();
```
재고 수량 필드에서 값 추출, 정수로 변환

```
String imagePath = imageField.getText().trim();
```
이미지 경로 필드에서 절대경로 추출

```
if (name.isEmpty() || imagePath.isEmpty()) {
                    throw new IllegalArgumentException("빈 칸 없이 입력해주세요.");
                }
```
이름 또는 이미지 경로가 비어있다면 예외 발생시킴
→ 필수 입력 검증 로직

```
Car car = new Car(name, code, price, LocalDate.now(), stock, imagePath);
```
Car 객체 생성 → 힙 영역에 객체가 생성됨
LocalDate.now()는 등록일(입고일)을 오늘 날짜로 설정

```
manager.addCar(car);
```
CarManager는 CarInputDialog의 필드로 전달받은 객체
실제 이 메서드는 CarManager 클래스 내부의 addCar(Car) 메서드를 호출하여
→ 자동차를 내부 리스트에 추가함
MVC 관점에서 보면:
CarInputDialog(View)가
CarManager(Controller → Model 연동)에게
Car(Model)를 전달

```
JOptionPane.showMessageDialog(this, "자동차 입고 완료!");
```
팝업 창으로 사용자에게 성공 메시지 표시

```
dispose();
```
현재 다이얼로그를 닫음
메모리 상에서도 다이얼로그는 제거됨

```
                } catch (NullPointerException ex) {
                    JOptionPane.showMessageDialog(this, "모든 숫자 필드를 올바르게 입력해주세요.");
```

숫자 필드 중 입력이 안 되어 getValue()가 null이면 NullPointerException 발생
사용자에게 입력 오류 메시지 표시

```
        } catch (IllegalArgumentException ex) {
                    JOptionPane.showMessageDialog(this, "입력 오류: " + ex.getMessage());
                }
            }
        }
```

사용자가 빈 칸을 입력했거나, 다른 예외 발생 시 메시지를 띄움

LoginFrame 클래스

```
package com.carshop.ui;
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import com.carshop.service.CarManager;

public class LoginFrame extends JFrame {
//      역할: 사용자 로그인 처리
//      해야 할 일:
//      ID, 비밀번호 입력 필드 UI 구성
//      로그인 버튼과 이벤트 리스너 구현
//      로그인 성공 시 CarGUI 화면 호출
//      로그인 실패 시 오류 메시지 출력
        private JTextField idField;
        private JPasswordField passwordField;
        private CarManager manager;  // ◇ 필드 추가

        public LoginFrame(CarManager manager) {
            this.manager = manager;  // ◇ 전달받기

            setTitle("로그인");
            setSize(400, 250);
            setLocationRelativeTo(null);
            setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            setLayout(new BorderLayout(10, 10));

            JLabel titleLabel = new JLabel("CarShop 로그인", SwingConstants.CENTER);
            titleLabel.setFont(new Font("맑은 고딕", Font.BOLD, 20));
            add(titleLabel, BorderLayout.NORTH);

            JPanel inputPanel = new JPanel(new GridLayout(2, 2, 10, 10));
            inputPanel.setBorder(BorderFactory.createEmptyBorder(20, 30, 20, 30));
            inputPanel.add(new JLabel("아이디:"));
            idField = new JTextField();
            inputPanel.add(idField);
            inputPanel.add(new JLabel("비밀번호:"));
            passwordField = new JPasswordField();
            inputPanel.add(passwordField);
            add(inputPanel, BorderLayout.CENTER);

            JButton loginButton = new JButton("로그인");
            loginButton.addActionListener(new LoginActionListener());

            JPanel buttonPanel = new JPanel();
            buttonPanel.add(loginButton);
            add(buttonPanel, BorderLayout.SOUTH);

            setVisible(true);
        }

        private class LoginActionListener implements ActionListener {
            @Override
            public void actionPerformed(ActionEvent e) {
                String inputId = idField.getText();
                String inputPw = new String(passwordField.getPassword());

                if (inputId.equals("admin") && inputPw.equals("1234")) {
                    JOptionPane.showMessageDialog(LoginFrame.this, "로그인 성공!");
                    dispose();
                    new CarGUI(manager);  // ◇ 로그인 성공 시 CarManager 넘김
                } else {
                    JOptionPane.showMessageDialog(LoginFrame.this, "로그인 실패: 아이디 또는
비밀번호가 잘못되었습니다.", "오류", JOptionPane.ERROR_MESSAGE);
                }
            }
        }
    }
```

1. 패키지 및 import

```
package com.carshop.ui;
```

이 클래스는 com.carshop.ui 패키지 소속
즉, UI 구성 요소 중 로그인 창을 담당

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import com.carshop.service.CarManager;
```

javax.swing.*: JFrame, JLabel, JTextField, JButton 등 Swing UI 컴포넌트
java.awt.*: 레이아웃 매니저 (BorderLayout, GridLayout)
java.awt.event.*: 버튼 클릭 이벤트(ActionEvent, ActionListener)
CarManager: 프로그램 전역에서 사용하는 자동차 재고 관리 객체

2. 클래스 정의

```
public class LoginFrame extends JFrame {
```

LoginFrame은 JFrame을 상속 → 하나의 창(window) 역할
역할: 로그인 화면 구성 및 처리

3. 필드 선언

```
private JTextField idField;
```

아이디 입력 필드 (한 줄 문자열 입력)

```
private JPasswordField passwordField;
```

비밀번호 입력 필드
getPassword()로 char 배열을 반환 → 보안상 안전

```
private CarManager manager;  // ◇ 필드 추가
```

자동차 관리 기능을 담당할 CarManager 객체 참조
이 프레임을 통해 로그인에 성공한 후, 다음 화면(CarGUI)에 전달됨

4. 생성자

```
public LoginFrame(CarManager manager) {
                this.manager = manager;  // ◇ 전달받기
```

CarManager 객체를 생성자에서 전달받음 (의존성 주입)
이후 CarGUI로 넘겨서 공유

JFrame 기본 설정

```
                setTitle("로그인");
                setSize(400, 250);
                setLocationRelativeTo(null);
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                setLayout(new BorderLayout(10, 10));
```

setTitle: 창 상단 제목 표시
setSize: 창의 너비 400px, 높이 250px
setLocationRelativeTo(null): 화면 가운데 정렬
EXIT_ON_CLOSE: 창 닫으면 전체 프로그램 종료
BorderLayout(10, 10): 각 영역 간 여백 설정

제목 라벨

```
JLabel titleLabel = new JLabel("CarShop 로그인", SwingConstants.CENTER);
                titleLabel.setFont(new Font("맑은 고딕", Font.BOLD, 20));
                add(titleLabel, BorderLayout.NORTH);
```

상단에 "CarShop 로그인"이라는 큰 제목 라벨
가운데 정렬, 굵은 글씨체

입력 필드 구성 (중앙 영역)

```
JPanel inputPanel = new JPanel(new GridLayout(2, 2, 10, 10));
inputPanel.setBorder(BorderFactory.createEmptyBorder(20, 30, 20, 30));
```

GridLayout(2, 2): 2행 2열로 아이디, 비밀번호 구성

EmptyBorder: 바깥 여백 설정

```
inputPanel.add(new JLabel("아이디:"));
idField = new JTextField();
inputPanel.add(idField);
inputPanel.add(new JLabel("비밀번호:"));
passwordField = new JPasswordField();
inputPanel.add(passwordField);
add(inputPanel, BorderLayout.CENTER);
```

첫 번째 행: "아이디" 라벨 + 입력 필드

두 번째 행: "비밀번호" 라벨 + 비밀번호 필드

로그인 버튼 구성 (하단 영역)

```
JButton loginButton = new JButton("로그인");
loginButton.addActionListener(new LoginActionListener());
```

"로그인" 버튼 생성

이벤트 리스너로 LoginActionListener 내부 클래스 연결

```
JPanel buttonPanel = new JPanel();
buttonPanel.add(loginButton);
add(buttonPanel, BorderLayout.SOUTH);
```

하단 패널에 버튼 배치 → BorderLayout.SOUTH

```
setVisible(true);
```

최종적으로 화면에 보이도록 설정

이 시점에 창이 실제로 생성되고 이벤트 루프 시작

5. 내부 클래스: LoginActionListener

```
private class LoginActionListener implements ActionListener {
```

로그인 버튼을 클릭했을 때 호출되는 이벤트 핸들러 클래스

ActionListener 구현 → actionPerformed() 구현 필요

actionPerformed

```
    @Override
    public void actionPerformed(ActionEvent e) {
```

버튼이 클릭되면 자동 호출됨

ActionEvent는 어떤 컴포넌트가 눌렸는지 정보 포함

```
String inputId = idField.getText();
```

사용자가 입력한 아이디 텍스트 읽어옴

```
String inputPw = new String(passwordField.getPassword());
```

JPasswordField는 char[] 반환 → 보안상 더 안전
여기서는 문자열로 변환해서 비교함 (단순 로그인이므로 허용)

로그인 검증 로직

```
if (inputId.equals("admin") && inputPw.equals("1234")) {
```

고정된 아이디/비밀번호 (admin/1234)와 비교
실제 앱이라면 DB 또는 파일에서 사용자 인증해야 함

```
JOptionPane.showMessageDialog(LoginFrame.this, "로그인 성공!");
```
팝업 메시지로 로그인 성공 알림

```
dispose();
```
현재 로그인 창 닫기 → 메모리 해제

```
new CarGUI(manager);  // ◇ 로그인 성공 시 CarManager 넘김
```
메인 화면인 CarGUI 창 열기
여기서도 CarManager를 넘김으로써 재고 관리 로직은 그대로 공유

```
} else {
                        JOptionPane.showMessageDialog(LoginFrame.this, "로그인 실패: 아이디 또는
비밀번호가 잘못되었습니다.", "오류", JOptionPane.ERROR_MESSAGE);
                    }
                }
            }
        }
```
로그인 실패 시 경고 메시지 출력
"오류" 타이틀, ERROR_MESSAGE 아이콘 사용

클래스 전체 요약

요소	설명
목적	CarGUI 실행 전 사용자 인증 역할
필드	아이디 필드, 비밀번호 필드, CarManager 객체
성공 흐름	아이디/비밀번호 OK → 로그인 성공 메시지 → 창 닫기 → CarGUI 열기
실패 흐름	로그인 실패 → 경고 팝업 표시
CarManager	재고 시스템의 상태 유지를 위해 전달됨 (싱글톤처럼 사용 가능)

MainFrame 클래스

```
package com.carshop.ui;
import javax.swing.*;
import java.awt.*;
import java.io.File;
import com.carshop.service.CarManager;

public class MainFrame extends JFrame {
//        역할: 프로그램 최초 시작 화면 (로고, 상점명, 개발자명)
//        해야 할 일:
//        JFrame 생성 및 기본 설정 (크기, 종료 동작 등)
//        상단에 로고 이미지 표시 (이미지 크기 조절 포함)
//        중앙에 상점 이름 JLabel
//        하단에 개발자명과 [시작하기] 버튼 배치
//        버튼 클릭 시 CarGUI 또는 로그인 창으로 화면 전환 연결
        private CarManager manager;  // ◇ 필드 추가

    public MainFrame(CarManager manager) {
        this.manager = manager;  // ◇ 전달받은 매니저 저장
        setTitle("CarShop 시스템 시작화면");
        setSize(600, 400);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        // ◇ 상단 로고
        JLabel logoLabel = new JLabel();
        logoLabel.setHorizontalAlignment(SwingConstants.CENTER);

        String imagePath =
"C:\\Java_Programming\\workspace\\Inventory_Management_System\\images\\logo.png";
        File imageFile = new File(imagePath);
        if (imageFile.exists()) {
            ImageIcon icon = new ImageIcon(imagePath);
            Image scaledImage = icon.getImage().getScaledInstance(300, 120, Image.SCALE_SMOOTH);
            logoLabel.setIcon(new ImageIcon(scaledImage));
        } else {
            logoLabel.setText("로고 이미지 없음");
            logoLabel.setFont(new Font("맑은 고딕", Font.ITALIC, 16));
        }

        // ◇ 중앙 상호명
        JLabel shopNameLabel = new JLabel("🚗 CarShop 자동차 상점 🚗", SwingConstants.CENTER);
        shopNameLabel.setFont(new Font("맑은 고딕", Font.BOLD, 24));

        // ◇ 하단 버튼
        JPanel bottomPanel = new JPanel(new BorderLayout());

        JLabel devLabel = new JLabel("개발자: 김다훈", SwingConstants.CENTER);
        devLabel.setFont(new Font("맑은 고딕", Font.PLAIN, 16));

        JButton startButton = new JButton("시작하기");
        startButton.setFont(new Font("맑은 고딕", Font.BOLD, 16));
        startButton.addActionListener(e -> {
            dispose();                    // 현재 창 닫고
            new LoginFrame(manager);      // ◇ CarManager 전달
        });

        bottomPanel.add(devLabel, BorderLayout.NORTH);
        bottomPanel.add(startButton, BorderLayout.SOUTH);

        add(logoLabel, BorderLayout.NORTH);
        add(shopNameLabel, BorderLayout.CENTER);
        add(bottomPanel, BorderLayout.SOUTH);

        setVisible(true);
    }
}
```

클래스 선언부

```
package com.carshop.ui;
```

이 클래스는 com.carshop.ui 패키지에 속해 있습니다.
UI 전용 기능을 모은 패키지이며, 화면 관련 클래스(LoginFrame, CarGUI)와 함께 동작합니다.

```
import javax.swing.*;
import java.awt.*;
import java.io.File;
import com.carshop.service.CarManager;
```

javax.swing.*: JFrame, JLabel, JButton 등 Swing UI 요소들을 전부 import합니다.
java.awt.*: 레이아웃(BorderLayout), Font, Image, Component 등의 그래픽 도구들 import.
java.io.File: 로고 이미지 파일의 존재 여부를 확인하기 위한 파일 클래스.
com.carshop.service.CarManager: 핵심 서비스 로직 담당 클래스. 이 MainFrame은 단순한 UI만 있는 게 아니라, CarManager 인스턴스를 받아서 LoginFrame에 전달합니다.

클래스 선언 및 필드

```
public class MainFrame extends JFrame {
```

JFrame을 상속받음으로써, 이 클래스 자체가 하나의 창(Window) 으로 동작합니다.
MainFrame은 프로그램 실행 시 가장 먼저 보여지는 메인 진입 UI입니다.

```
private CarManager manager;  // ◇ 필드 추가
```

CarManager 인스턴스를 필드로 보관합니다.
이 인스턴스는 이후 LoginFrame이나 CarGUI로 전달되며, 실제 비즈니스 로직(재고, 입출고 등)을 실행하는 데 사용됩니다.
즉, UI는 UI고, 실제 동작은 CarManager가 함 → MVC 분리 구조를 따름.

생성자 MainFrame(CarManager manager)

```
public MainFrame(CarManager manager) {
```

생성자 호출 시 CarManager 객체를 전달받습니다.
이 객체는 이미 외부에서 생성된 비즈니스 로직 컨트롤러입니다.

```
this.manager = manager;  // ◇ 전달받은 매니저 저장
```

필드에 저장해 두었다가, 로그인 프레임 등 다른 곳에 넘길 준비.

기본 설정

```
setTitle("CarShop 시스템 시작화면");
```

창의 상단 타이틀 바에 표시될 이름을 설정합니다.

```
setSize(600, 400);
```

프레임의 크기를 가로 600px, 세로 400px로 지정합니다.

```
setLocationRelativeTo(null);
```

프레임을 화면 정중앙에 띄우는 설정입니다.

```
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

사용자가 창을 닫으면 프로그램이 완전히 종료되도록 설정합니다.

```
setLayout(new BorderLayout());
```

프레임의 레이아웃을 BorderLayout으로 지정합니다.
이 레이아웃은 NORTH, CENTER, SOUTH, EAST, WEST 다섯 영역으로 구성됩니다.
우리는 여기서 NORTH (로고), CENTER (상호명), SOUTH (버튼/개발자명) 를 사용합니다.

상단 로고 구성

```
JLabel logoLabel = new JLabel();
        logoLabel.setHorizontalAlignment(SwingConstants.CENTER);
```

이미지를 담을 JLabel 생성.

setHorizontalAlignment(SwingConstants.CENTER)로 중앙 정렬.

```
String imagePath =
"C:\\Java_Programming\\workspace\\Inventory_Management_System\\images\\logo.png";
        File imageFile = new File(imagePath);
```

로고 이미지의 경로를 지정합니다.

File imageFile을 통해 실제 파일이 존재하는지 체크할 수 있음.

```
if (imageFile.exists()) {
```

파일이 존재할 경우에만 이미지 로딩을 시도합니다.

```
ImageIcon icon = new ImageIcon(imagePath);
            Image scaledImage = icon.getImage().getScaledInstance(300, 120, Image.SCALE_SMOOTH);
            logoLabel.setIcon(new ImageIcon(scaledImage));
```

ImageIcon으로 이미지 로딩 → getImage()로 Image 객체로 추출

getScaledInstance()로 300x120 크기로 부드럽게 리사이징 → 다시 ImageIcon에 담아서 JLabel에 설정

이때 리사이징된 이미지는 GC Heap의 이미지 객체 메모리에 존재하며, JLabel은 이를

```
} else {
            logoLabel.setText("로고 이미지 없음");
            logoLabel.setFont(new Font("맑은 고딕", Font.ITALIC, 16));
        }
```

이미지가 없을 경우, 텍스트로 대체 → "로고 이미지 없음" 텍스트를 띄움

중앙 상호명 구성

```
// ◇ 중앙 상호명
        JLabel shopNameLabel = new JLabel("🚗 CarShop 자동차 상점 🚗", SwingConstants.CENTER);
        shopNameLabel.setFont(new Font("맑은 고딕", Font.BOLD, 24));
```

JLabel로 중앙 상호명 텍스트 설정

글씨 크기 24pt, 굵은 글씨체

하단 개발자명 + 버튼 구성

```
JPanel bottomPanel = new JPanel(new BorderLayout());
```

하단에 또 하나의 Panel을 만들고, 역시 BorderLayout 사용

```
JLabel devLabel = new JLabel("개발자: 김다훈", SwingConstants.CENTER);
        devLabel.setFont(new Font("맑은 고딕", Font.PLAIN, 16));
```

개발자명을 표시하는 라벨 (중앙 정렬)

```
JButton startButton = new JButton("시작하기");
        startButton.setFont(new Font("맑은 고딕", Font.BOLD, 16));
```

사용자 클릭을 받을 “시작하기” 버튼

```
startButton.addActionListener(e -> {
                dispose();                          // 현재 창 닫고
                new LoginFrame(manager);        //  ◇  CarManager 전달
            });
```

람다식 리스너로 이벤트 핸들링

dispose() → 현재 MainFrame 창 종료

new LoginFrame(manager) → 로그인 창 생성 + CarManager 전달

이때 호출 흐름:

```
MainFrame → (시작하기 클릭) → LoginFrame → 로그인 성공 시 CarGUI
```

컴포넌트 붙이기

```
  bottomPanel.add(devLabel, BorderLayout.NORTH);
  bottomPanel.add(startButton, BorderLayout.SOUTH);
```

하단 Panel에 위에는 개발자명, 아래는 버튼

```
add(logoLabel, BorderLayout.NORTH);
add(shopNameLabel, BorderLayout.CENTER);
add(bottomPanel, BorderLayout.SOUTH);
```

JFrame 자체에 NORTH, CENTER, SOUTH 순서로 요소를 붙임

창 표시

```
setVisible(true);
```

여기서 비로소 실제 창이 화면에 출력됩니다

이전까지는 컴포넌트들이 내부에서 준비만 되었던 상태

순서	동작 설명
1	생성자로 CarManager가 전달됨
2	UI 구성 (로고 → 상호명 → 하단 버튼)
3	버튼 누르면 dispose() 후 LoginFrame 열림
4	LoginFrame에서 로그인 후 성공 시 CarGUI 생성

FileHandler 클래스

```
package com.carshop.util;
import com.carshop.model.Car;
import java.io.*;
import java.util.ArrayList;
import java.util.List;
public class FileHandler {
//       역할: 파일 저장/불러오기 공통 기능 담당
//       해야 할 일:
//       saveToTextFile(List<Car> list, String path) : 파일 저장 구현
//       loadFromTextFile(String path) : 파일 읽기 후 List<Car> 반환
//       파일 입출력 예외 처리 및 메시지 출력
        public static void saveToTextFile(List<Car> list, String path) {
            try (BufferedWriter writer = new BufferedWriter(new FileWriter(path))) {
                for (Car car : list) {
                    writer.write(car.toDataLine());
                    writer.newLine();
                }
                System.out.println("☑ 파일 저장 성공: " + path);
            } catch (IOException e) {
                System.out.println("❌ 파일 저장 실패: " + e.getMessage());
            }
        }

        /**
         * 역할: 텍스트 파일에서 Car 객체 리스트로 로드
         */
        public static List<Car> loadFromTextFile(String path) {
            List<Car> list = new ArrayList<>();

            try (BufferedReader reader = new BufferedReader(new FileReader(path))) {
                String line;
                while ((line = reader.readLine()) != null) {
                    try {
                        Car car = Car.fromDataLine(line);
                        list.add(car);
                    } catch (Exception e) {
                        System.out.println("⚠ 잘못된 형식의 데이터 무시됨: " + line);
                    }
                }
                System.out.println("☑ 파일 로드 성공: " + path);
            } catch (IOException e) {
                System.out.println("❌ 파일 로드 실패: " + e.getMessage());
            }

            return list;
        }
}
```

```
package com.carshop.util;
```

이 클래스가 속한 패키지는 com.carshop.util입니다.
보통 util은 “유틸리티”, 즉 여러 곳에서 공통으로 사용할 수 있는 도구나 보조 기능을 모아놓은 곳이에요.
그래서 파일 입출력과 같은 반복되는 작업을 여기서 처리하는 거죠.

```
import com.carshop.model.Car;
```

Car 클래스가 필요해서 import 했어요.
여기서 Car는 자동차 한 대의 정보를 담는 데이터 모델 클래스입니다.
파일 저장/로드 시 자동차 정보를 문자열로 변환하거나, 다시 객체로 만드는 데 사용됩니다.

```
import java.io.*;
```

자바 표준 입출력 관련 클래스들(예: FileReader, FileWriter, BufferedReader, BufferedWriter, IOException 등)을 전부 포함시키기 위해 *로 한꺼번에 import 했어요.
이걸 통해 텍스트 파일을 읽고 쓰는 기능을 쉽게 쓸 수 있죠.

```
import java.util.ArrayList;
import java.util.List;
```
리스트를 다루기 위해 List 인터페이스와 ArrayList 구현체를 import 했습니다.
이걸로 파일에서 읽은 자동차들을 저장할 리스트를 만듭니다.

```
public class FileHandler {
```
FileHandler라는 이름의 클래스 선언입니다.
역할은 파일 저장/불러오기 같은 공통 기능을 담당하는 것입니다.
모든 메서드는 static이라서, 객체를 생성하지 않고 바로 사용할 수 있어요.

saveToTextFile 메서드

```
public static void saveToTextFile(List<Car> list, String path) {
```
리스트에 있는 Car 객체들을 텍스트 파일로 저장하는 메서드입니다.
list는 저장할 자동차 목록, path는 저장할 파일의 경로를 의미합니다.

```
try (BufferedWriter writer = new BufferedWriter(new FileWriter(path))) {
```
try-with-resources 구문으로 BufferedWriter 객체를 생성했어요.
BufferedWriter는 버퍼링을 해서 파일에 효율적으로 쓰기 위해 사용하고,
FileWriter는 실제로 파일에 문자를 쓰는 역할입니다.
path에 지정한 위치에 파일을 만들거나 덮어씁니다.
try-with-resources라서 이 블록 끝나면 writer가 자동으로 닫혀서 파일 자원 누수가 없습니다.

```
for (Car car : list) {
    writer.write(car.toDataLine());
    writer.newLine();
}
```
리스트에 있는 모든 Car 객체를 순회하면서 한 대씩 저장합니다.
car.toDataLine()은 Car 객체를 문자열 한 줄로 바꾸는 메서드입니다. (예:
"Sonata|1001|25000|2023-07-25|5|path/to/image.jpg")
이 문자열을 파일에 씁니다.
writer.newLine()으로 한 줄씩 구분해서 저장하니까 파일에서 읽을 때 한 줄이 한 대 자동차 정보를 의미해요.

```
System.out.println("☑ 파일 저장 성공: " + path);
```
저장이 성공했음을 콘솔에 표시합니다.
개발자나 사용자에게 파일 저장이 잘 됐는지 알려주는 용도입니다.

```
} catch (IOException e) {
                    System.out.println("❌ 파일 저장 실패: " + e.getMessage());
}
```
파일 저장 중 에러가 발생하면 여기서 잡아서 에러 메시지를 출력합니다.
예외가 터져도 프로그램이 바로 종료되지 않고, 알림만 주고 안전하게 처리됩니다.

loadFromTextFile 메서드

```
public static List<Car> loadFromTextFile(String path) {
```
지정된 파일 경로에서 자동차 목록을 읽어서 List<Car>를 반환하는 메서드입니다.

```
List<Car> list = new ArrayList<>();
```
읽어온 자동차 데이터를 저장할 빈 리스트를 생성합니다.
이 리스트를 최종적으로 메서드 밖으로 반환할 거예요.

```
try (BufferedReader reader = new BufferedReader(new FileReader(path))) {
```

BufferedReader와 FileReader를 사용해서 텍스트 파일을 읽습니다.
역시 try-with-resources로 자동 자원 관리를 해줘서 안전합니다.

```
String line;
while ((line = reader.readLine()) != null) {
```

파일에서 한 줄씩 읽어옵니다.
readLine()은 더 이상 읽을 라인이 없으면 null을 반환하니까, 그걸 기준으로 반복을 종료합니다.


```
String line;
while ((line = reader.readLine()) != null) {
```

파일에서 한 줄씩 읽어옵니다.
readLine()은 더 이상 읽을 라인이 없으면 null을 반환하니까, 그걸 기준으로 반복을 종료합니다.


```
try {
            Car car = Car.fromDataLine(line);
            list.add(car);
        } catch (Exception e) {
System.out.println("⚠ 잘못된 형식의 데이터 무시됨: " + line);
}
```

각 줄을 Car.fromDataLine()을 이용해 다시 Car 객체로 변환합니다.
만약 데이터 형식이 맞지 않거나 변환에 실패하면 예외가 발생하는데, 이걸 잡아서
해당 줄은 건너뛰고 경고 메시지만 출력합니다.
이렇게 하면 파일에 일부 손상된 데이터가 있어도 전체 로드는 계속 유지됩니다.

```
System.out.println("☑ 파일 로드 성공: " + path);
```

모든 읽기가 정상적으로 끝나면 성공 메시지를 출력합니다.

```
 } catch (IOException e) {
                        System.out.println("❌ 파일 로드 실패: " + e.getMessage());
                    }
```

파일을 읽는 도중 IO 오류가 발생하면 여기서 처리하고 실패 메시지를 출력합니다

```
return list;
```

읽어온 자동차 리스트를 호출한 곳으로 반환합니다.
반환된 리스트는 CarManager 같은 다른 클래스에서 재고 목록으로 사용됩니다.

실제 사용 흐름(전체 맥락)
프로그램이 자동차 목록을 저장할 때는 CarManager 클래스에서 saveData(path)를 호출합니다.
이 메서드는 내부적으로 FileHandler.saveToTextFile(carList, path)를 호출해서 실제 파일 저장 작업을 위임합니다.
반대로, 프로그램 시작이나 불러오기 작업 시 CarManager는 loadData(path)를 호출합니다.
loadData는 FileHandler.loadFromTextFile(path)를 호출해 파일을 읽고, 반환된 리스트로 현재 재고 리스트를 교체합니다.
FileHandler는 파일 입출력 과정에서 오류가 나면 콘솔에 경고 메시지를 띄우고, 프로그램이 중단되지 않도록 안전하게 처리합니다.
이 클래스를 따로 객체화 하지 않고 static 메서드로 바로 호출하기 때문에, 어디서든 쉽게 파일 저장/로드 기능을 재사용할 수 있습니다.

ImageUtil 클래스

```
package com.carshop.util;

import javax.swing.*;
import java.awt.*;
import java.io.File;
public class ImageUtil {
//      역할: 이미지 크기 조절 및 처리
//      해야 할 일:
//      이미지 리사이징 메서드 (예: resizeImage(String path, int width, int height)) 작성
//      CarGUI, MainFrame에서 이미지 표시할 때 활용
        /**
     * 경로로부터 이미지를 읽고 지정된 크기로 리사이징하여 ImageIcon으로 반환
     *
     * @param path 이미지 파일 경로
     * @param width 원하는 너비
     * @param height 원하는 높이
     * @return 리사이징된 ImageIcon 객체, 실패 시 null
     */
    public static ImageIcon resizeImage(String path, int width, int height) {
        File file = new File(path);
        if (!file.exists()) {
            System.out.println("이미지 파일이 존재하지 않습니다: " + path);
            return null;
        }

        ImageIcon icon = new ImageIcon(path);
        Image scaled = icon.getImage().getScaledInstance(width, height, Image.SCALE_SMOOTH);
        return new ImageIcon(scaled);
    }
}
```

1. 패키지 선언

```
package com.carshop.util;
```

이 클래스는 com.carshop.util 패키지에 속합니다.
util은 공통 기능 묶음으로, 이미지 처리 기능도 자주 재사용되기 때문에 이곳에 둡니다.
CarGUI, MainFrame 등이 이 유틸 클래스의 resizeImage()를 불러다가 씁니다.

2. import 문

```
import javax.swing.*;
import java.awt.*;
import java.io.File;
```

javax.swing.*
ImageIcon을 사용하려면 Swing 패키지를 import 해야 해요.
ImageIcon은 이미지 파일을 GUI에서 보여줄 수 있게 해주는 클래스입니다.

java.awt.*
Image 클래스 사용을 위해 필요합니다.
getImage()나 getScaledInstance() 같은 메서드는 Image 타입에서 제공됩니다.

java.io.File
경로(path)가 실제로 존재하는지 확인할 때 사용합니다.
이미지 파일이 없는 경우 오류 방지를 위해 File 객체로 존재 여부를 검사합니다.

클래스 선언

```
public class ImageUtil {
```

public이기 때문에 다른 패키지에서도 자유롭게 사용할 수 있습니다.
이름 그대로 "이미지 관련 유틸리티" 역할을 합니다.
객체를 생성할 필요 없이 사용하는 유틸리티라서, 내부 메서드도 static으로 선언됩니다.

4. 메서드 선언부

```
public static ImageIcon resizeImage(String path, int width, int height) {
```

주요 특징

static: 객체 생성 없이 ImageUtil.resizeImage(...)처럼 바로 호출 가능.

호출 위치:

CarGUI, MainFrame 클래스 내부에서, 자동차 사진을 보여줄 때 다음처럼 호출됨:

```
ImageIcon icon = ImageUtil.resizeImage(car.getImagePath(), 200, 150);
label.setIcon(icon);
```

매개변수

String path: 이미지 파일의 절대 경로나 상대 경로.

int width, int height: 이미지가 리사이징되어야 할 크기.

반환

성공 시: 리사이징된 ImageIcon 객체.

실패 시: null.

5. 파일 존재 확인

```
File file = new File(path);
if (!file.exists()) {
    System.out.println("이미지 파일이 존재하지 않습니다: " + path);
    return null;
}
```

동작 설명

File file = new File(path);

주어진 경로의 파일 객체 생성.

아직 실제로 읽거나 열지는 않았고, 존재 확인만 할 예정.

if (!file.exists()) { ... }

파일이 실제로 존재하는지 검사.

이미지 경로가 틀렸거나 파일이 삭제됐을 경우를 대비한 방어 코드.

return null;

실패했기 때문에 이 메서드를 호출한 쪽에는 null을 넘겨줘서,

호출 쪽에서 "이미지 없음" 처리 등으로 대체할 수 있도록 합니다.

6. 이미지 읽기 및 리사이징

```
ImageIcon icon = new ImageIcon(path);
    Image scaled = icon.getImage().getScaledInstance(width, height, Image.SCALE_SMOOTH);
    return new ImageIcon(scaled);
```

ImageIcon icon = new ImageIcon(path);

ImageIcon 생성자에 파일 경로를 넘기면, 내부적으로 해당 파일의 이미지를 메모리로 로드합니다.

이 icon은 원본 크기를 가지고 있어요.

icon.getImage()

ImageIcon에서 실제 이미지(java.awt.Image) 객체를 꺼냅니다.

Image는 그래픽 작업을 위한 클래스이고, 다양한 변형 작업이 가능합니다.

.getScaledInstance(width, height, Image.SCALE_SMOOTH)

이미지를 지정된 크기(width, height)로 리사이징합니다.

SCALE_SMOOTH는 부드럽게 보정된 고품질 스케일링을 의미해요.

↳ 더 느리지만 가장 보기 좋습니다.

new ImageIcon(scaled)

리사이징된 Image 객체를 다시 ImageIcon으로 감싸서 GUI에 표시 가능하게 만듭니다.

JVM 메모리 구조와 연관된 설명

영역	관련 설명
힙 (Heap)	ImageIcon, Image, File 객체 등은 모두 new로 생성되며, 힙 메모리에 저장됩니다. 즉, 프로그램 전역에서 사용할 수 있는 동적 객체들이 여기에 올라갑니다
메서드 영역	resizeImage()는 static 메서드이므로, 클래스 로딩 시 메서드 영역에 올라갑니다.
스택 메모리	resizeImage(path, width, height) 호출 시 각 지역 변수(file, icon, scaled)는 스택 프레임에 생성됩니다. 메서드 종료 시 스택은 제거되고, return된 ImageIcon만 힙에 남습니다.

DateUtil 클래스

```
package com.carshop.util;

import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
public class DateUtil {
//        역할: 날짜 관련 편의 메서드 제공
//        해야 할 일:
//        오늘 날짜를 "yyyy-MM-dd" 문자열로 리턴하는 메서드 (getTodayDate()) 작성
        /**
     * 오늘 날짜를 "yyyy-MM-dd" 형식의 문자열로 반환
     */
    public static String getTodayDate() {
        LocalDate today = LocalDate.now();
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");
        return today.format(formatter);

    }

    // 필요 시 다른 날짜 관련 유틸 메서드도 추가 가능
}
```

```
package com.carshop.util;
```

이 클래스는 com.carshop.util 패키지 소속이라는 뜻이야.
즉, 프로젝트 내에서 “공통 유틸 기능”을 담는 위치에 이 클래스를 두었다는 의미.
다른 클래스에서 import com.carshop.util.DateUtil;로 불러와 사용할 수 있어.

```
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
```

LocalDate: Java 8 이상에서 사용하는 날짜 클래스. 날짜 정보를 다룰 때 사용 (시간은 없음).
DateTimeFormatter: 날짜를 문자열로 포맷팅하거나, 문자열을 날짜로 변환할 때 사용.
이 두 클래스는 내부적으로 java.time 패키지에 속하며, 스레드-세이프한 현대적인 날짜 처리 방식이야.

```
public class DateUtil {
```

이 클래스는 public이고 정적(static) 메서드만 포함된 유틸리티 클래스야.
객체 생성 없이 사용 가능 (DateUtil.getTodayDate() 식으로 바로 호출 가능).

```
// 역할: 날짜 관련 편의 메서드 제공
// 해야 할 일:
// 오늘 날짜를 "yyyy-MM-dd" 문자열로 리턴하는 메서드 (getTodayDate()) 작성
```

이 주석은 클래스의 의도와 책임(R) 을 설명하고 있어.
향후 유지보수나 협업 시 이 클래스의 역할이 날짜 편의 기능임을 쉽게 이해할 수 있게 함.

```
/**
     * 오늘 날짜를 "yyyy-MM-dd" 형식의 문자열로 반환
     */
    public static String getTodayDate() {
```

public: 외부 클래스 어디에서든 호출 가능.
static: 객체 생성 없이 클래스명으로 바로 호출 가능.
String: 리턴 타입은 포맷된 문자열.
예: "2025-07-27" (오늘 날짜 기준)

```
LocalDate today = LocalDate.now();
```
현재 날짜를 구하는 코드.
LocalDate.now()는 시스템 시간으로부터 오늘 날짜를 구함.
예: 2025-07-27
이 today 객체는 JVM의 스택(Stack) 영역에 로컬 변수로 존재하고, LocalDate 객체 자체는 힙(Heap) 에 존재함.

```
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");
```
날짜를 문자열로 포맷할 수 있는 Formatter 객체 생성.
"yyyy-MM-dd" 포맷은 4자리 연도, 2자리 월, 2자리 일로 구성됨.
예: 2025-07-27
이 formatter 객체도 힙에 생성되며, 포맷 규칙 정보를 담고 있음.

```
return today.format(formatter);
```
today.format(...): LocalDate 객체를 문자열로 포맷함.
예: "2025-07-27"이라는 문자열이 반환됨.
즉, 내부적으로는 StringBuilder 등을 사용해 날짜 객체를 문자열로 변환함.

```
}
```
getTodayDate() 메서드 종료.

```
// 필요 시 다른 날짜 관련 유틸 메서드도 추가 가능
```
향후 여기에 "문자열 → 날짜" 또는 "날짜 차이 계산" 등 추가적인 날짜 관련 유틸 기능을 넣을 수 있음.

JVM 메모리 구조 관점 요약

요소	위치	설명
today	스택(Stack)	메서드 내 로컬 변수 (LocalDate 객체 참조)
formatter	스택(Stack)	포맷터 객체 참조
LocalDate, DateTimeFormatter 인스턴스	힙(Heap)	new 키워드는 없지만 내부적으로 힙에 생성됨
DateUtil 클래스 정보	메서드 영역(Method Area)	클래스 정의 정보와 static 메서드들 포함
반환된 String	힙(Heap)	interned 문자열 풀에 저장되거나 새로 힙에 생성됨

이 메서드는 실제 어디서 호출될까?
예시: 자동차 등록 시 오늘 날짜 자동 입력
String today = DateUtil.getTodayDate();
Car newCar = new Car("K7", "1003", 32000000, today, 2, "c:\\car\\k7.jpg");
자동차 등록 화면에서 사용자가 날짜를 수동으로 선택하지 않고, 오늘 날짜를 자동으로 입력할 때 쓰일 수 있음.

항목	설명
클래스명	DateUtil
목적	날짜 관련 기능을 모아놓은 유틸 클래스
제공 메서드	getTodayDate() - 오늘 날짜를 "yyyy-MM-dd" 형식으로 반환
호출 위치 예시	CarGUI, CarManager에서 자동차 생성 시
메모리 구조	today와 formatter는 스택, 객체는 힙
특징	static 유틸리티 - 객체 생성 불필요