Power Java

제8장 클래스와 객체 I

이번 장에서 학습할 내용

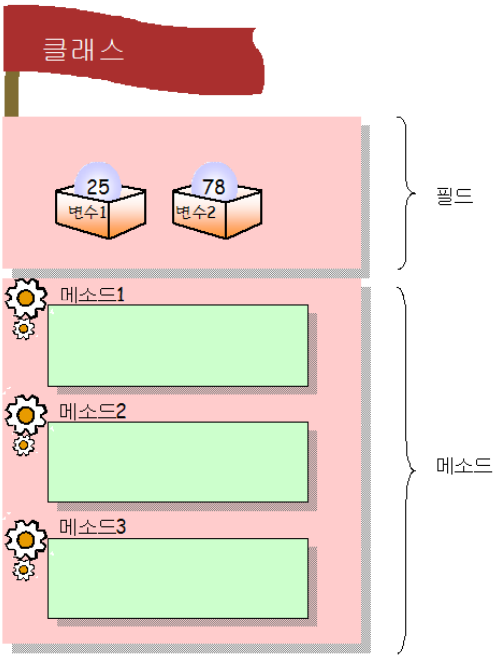
- 클래스와 객체
- 객체의 일생
- 메소드
- 필드
- UML

QUIZ

1. 객체는 _____속성_____과 _____동작_____을 가지고 있다.
2. 자동차가 객체라면 클래스는 _설계도_이다.

먼저 앞장에서 학습한 클래스와 객체의 개념을 복습해봅시다.

클래스의 구성



- 클래스(class)는 객체의 설계도라 할 수 있다.
- 클래스는 필드와 메소드로 이루어진다.
- 필드(field)는 객체의 속성을 나타낸다.
- 메소드(method)는 객체의 동작을 나타낸다.

클래스 정의의 예

```
class Car {

    // 필드 정의
    public int speed; // 속도
    public int mileage; // 주행거리
    public String color; // 색상

    // 메소드 정의
    public void speedUp() { // 속도 증가 메소드
            speed += 10;
    }

    public void speedDown() { // 속도 감소 메소드
            speed -= 10;
    }

    public String toString() { // 객체의 상태를 문자열로 반환하는 메소드
            return "속도: " + speed + " 주행거리: " + mileage + " 색상: " + color;
    }
}
```

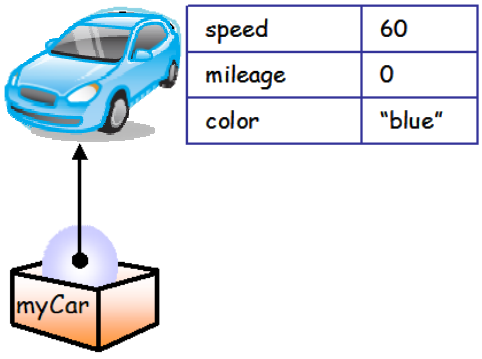
필드 정의!

메소드 정의!

테스트 클래스

```
public class CarTest {
    public static void main(String[] args) {
            Car myCar = new Car();        // 첫번째 객체 생성
            Car yourCar = new Car(); // 두번째 객체 생성

            myCar.speed = 60;  // 객체의 필드 변경
            myCar.mileage = 0; // 객체의 필드 변경
            myCar.color = "blue";          // 객체의 필드 변경

            myCar.speedUp();               // 객체의 메소드 호출
            System.out.println(myCar);

            yourCar.mileage = 10;          // 객체의 필드 변경
            yourCar.speed = 120;     // 객체의 필드 변경
            yourCar.color = "white";  // 객체의 필드 변경

            yourCar.speedDown();           // 객체의 메소드 호출
            System.out.println(yourCar);
    }
}
```

속도: 70 주행거리: 0 색상: blue
속도: 110 주행거리: 10 색상: white



객체의 일생

Car c = new Car();

c.speedUp();

c = null;

객체의 생성

```
Car     myCar;              // ① 참조 변수를 선언
myCar = new  Car();         // ② 객체를 생성하고 ③ 참조값을 myCar에 저장
```

① 참조 변수 선언

Car 타입의 객체를 참조할 수 있는 변수 myCar를 선언한다.

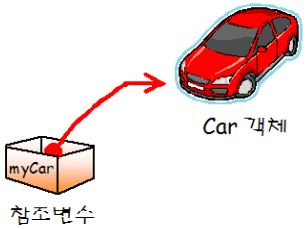
② 객체 생성

new 연산자를 이용하여 객체를 생성하고 객체 참조값을 반환한다.

Car 객체

③ 참조 변수와 객체의 연결

생성된 새로운 객체의 참조값을 myCar라는 참조 변수에 대입한다.

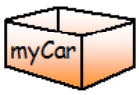


주의

```
Car     myCar;
```

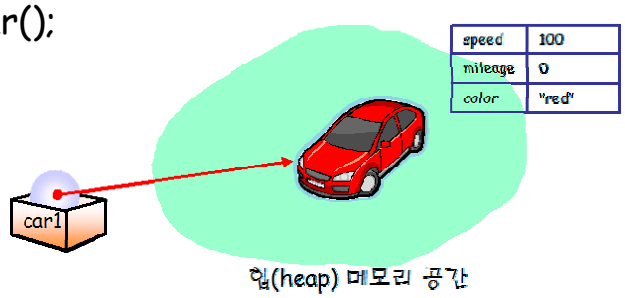
위의 문장으로 객체가
생성되는 것은 아님!!!

객체를 가리키는 참조값을
담을 수 있는 변수만 생성됨.

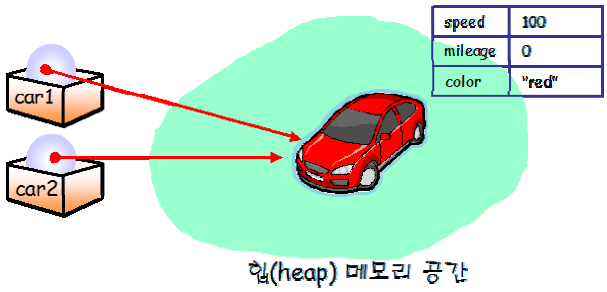


참조 변수와 대입 연산

- Car car1 = new Car();

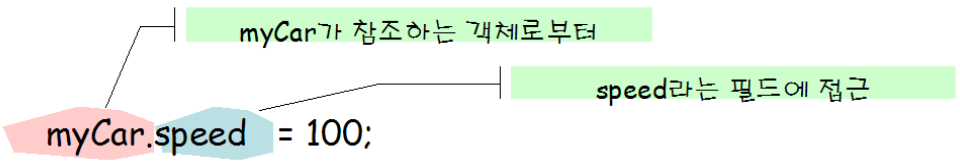


- Car car2 = car1; // 대입 연산의 의미



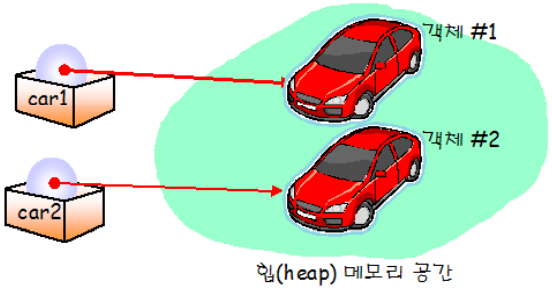
객체의 사용

- 객체를 이용하여 필드와 메소드에 접근할 수 있다.

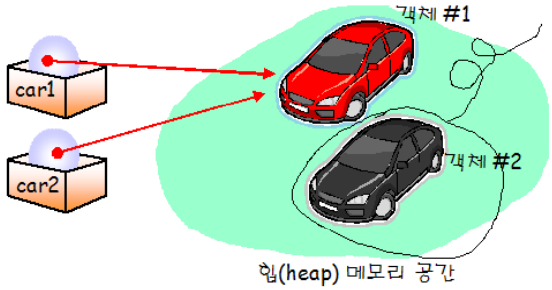


객체의 소멸

```
Car car1 = new Car();    // 첫번째 객체
Car car2 = new Car();    // 두번째 객체
```



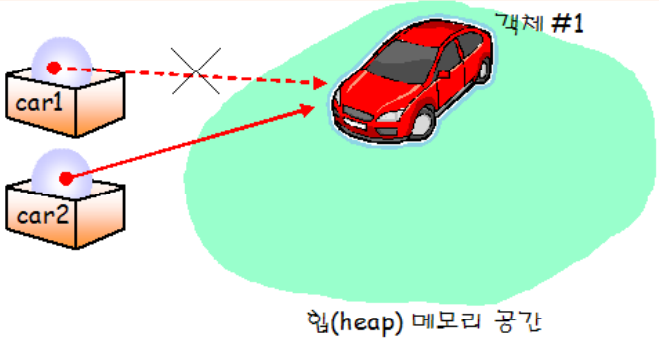

```
car2 = car1;    // car1과 car2는 같은 객체를 가리킨다.
                // car2가 가리켰던 객체는 쓰레기 수집기에 의하여 수거된다.
```



객체는 참조가 없어지면 소멸!!

객체의 소멸

```
car1 = null;    // 객체 1은 아직도 활성화된 참조가 있기 때문에 소멸되지 않는다.
```



메소드

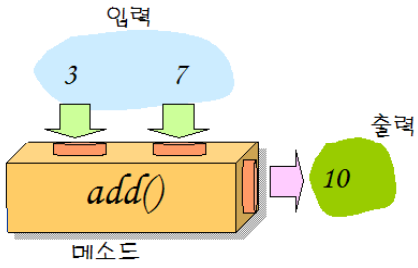


그림 8.6 메소드는 입력을 받아서 출력을 내보내는 작은 기계와 같다.

메소드 선언은 다음과 같은 형식을 가진다.

```
[수식자] 반환형 myMethod ( 매개변수 목록 ) {
        // 문장들
        ...
}
```


매개 변수

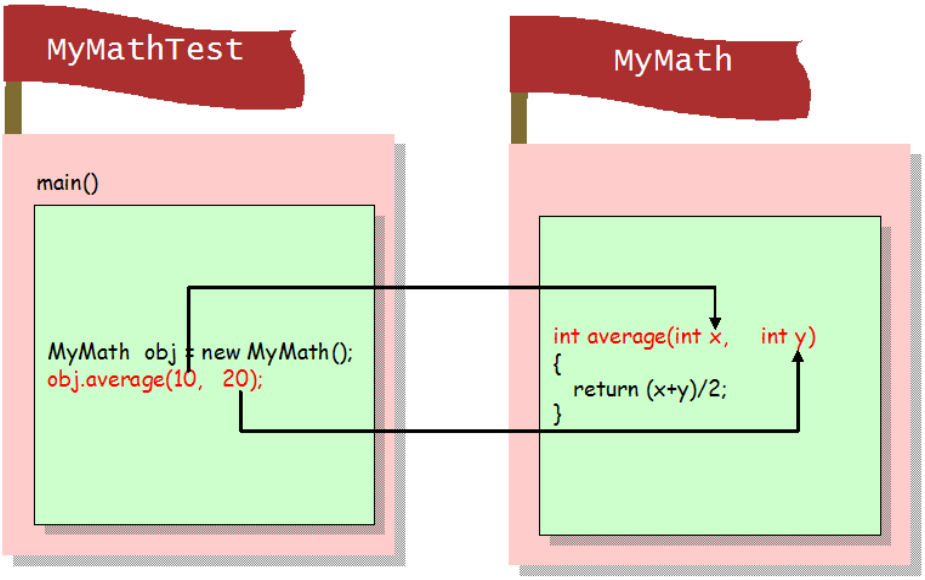


그림 8.7 메소드 호출시 매개 변수의 전달

값에 의한 전달

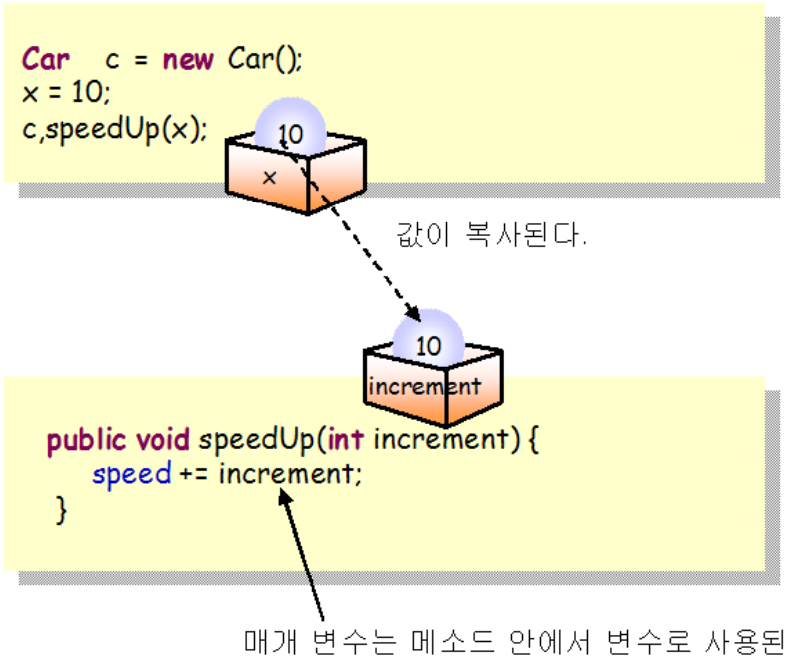


그림 8.8 매개 변수가 기초 타입의 변수일 경우, 값이 복사된다.

매개 변수가 객체인 경우

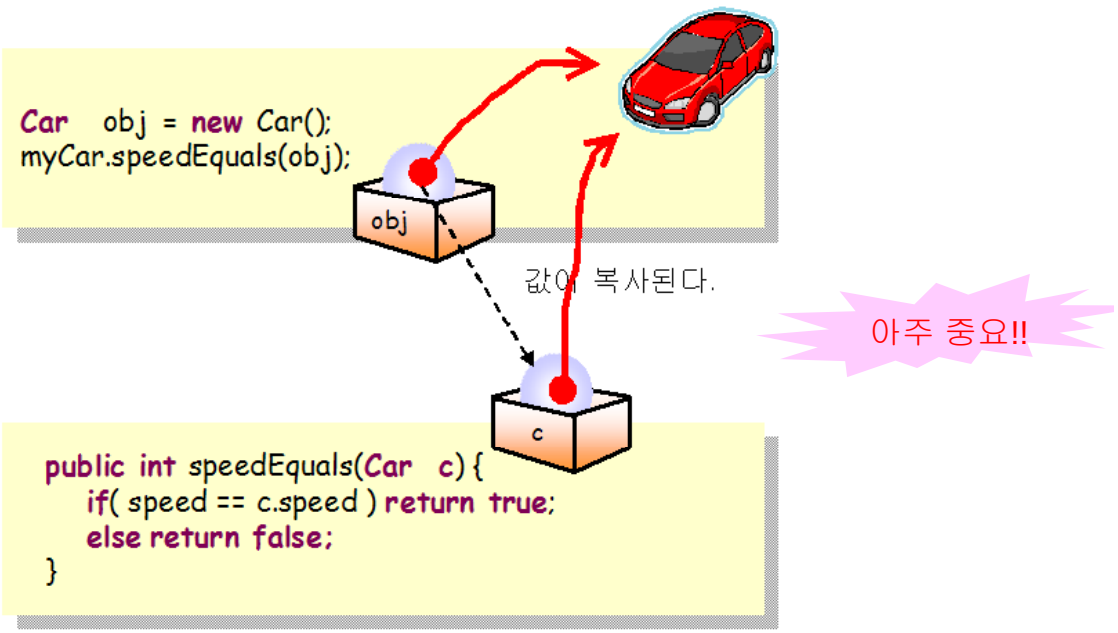
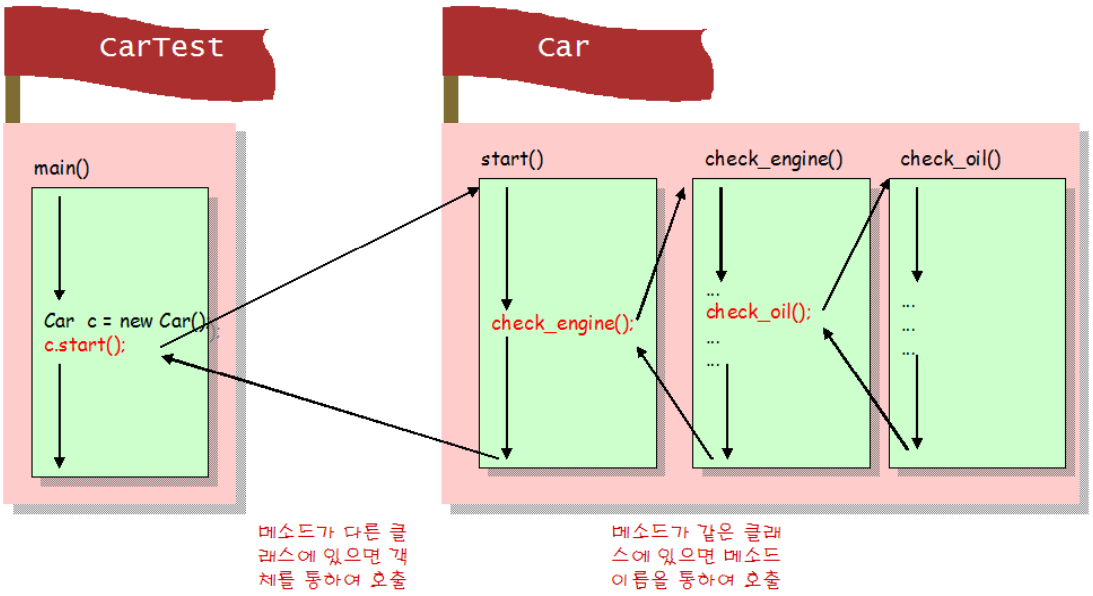


그림 8.9 매개 변수가 참조 타입의 변수일 경우에도 역시 참조값이 복사된다.

메소드 호출



메소드 호출의 예제

```
import java.util.*;
class DiceGame {

    int diceFace;
    int userGuess;

    private void RollDice()
    {
            diceFace = (int)(Math.random() * 6) + 1;
    }
    private int getUserInput(String prompt)
    {
            System.out.println(prompt);
            Scanner s = new Scanner(System.in);
            return s.nextInt();
    }
```

메소드 호출의 예제

```
private void checkUserGuess()
    {
        if( diceFace == userGuess )
            System.out.println("맞았습니다");
        else
            System.out.println("틀렸습니다");
    }
    public void startPlaying()
    {
        int userGuess = getUserInput("예상값을 입력하시오: ");
        RollDice();
        checkUserGuess();
    }
}

public class DiceGameTest {
     public static void main(String[] args) {
        DiceGame game = new DiceGame();
        game.startPlaying();
     }
}
```


결과 화면

```
예상값을 입력하시오:
3
틀렸습니다
```

중복 메소드

- 메소드 오버라이딩(method overriding)

```
// 정수값을 제곱하는 메소드
public int square(int i)
{
	return i*i;
}

// 실수값을 제곱하는 메소드
public double square(double i)
{
	return i*i;
}
```

- 메소드 호출시 매개 변수를 보고 일치하는 메소드가 호출된다.
- 만약 square(3.14)와 같이 호출되면 컴파일러는 매개 변수의 개수, 타입, 순서 등을 봐서 두 번째 메소드를 호출한다.

중복 메소드 예제

```
class Car {

	// 필드 선언
	private int speed; // 속도

	// 중복 메소드: 정수 버전
	public void setSpeed(int s) {
		speed = s;
		System.out.println("정수 버전 호출");
	}

	// 중복 메소드: 실수 버전
	public void setSpeed(double s) {
		speed = (int)s;
		System.out.println("실수 버전 호출");
	}
}
```

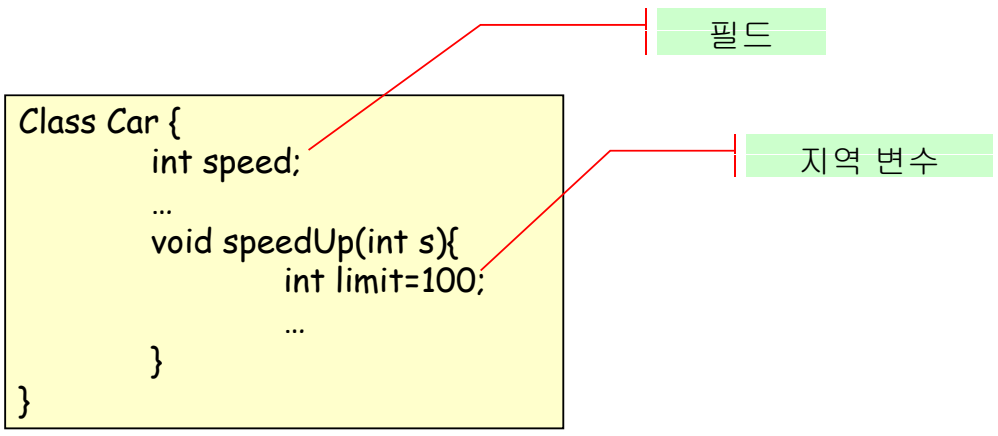
중복 메소드 예제

```
public class CarTest1 {
    public static void main(String[] args) {
        Car myCar = new Car(); // 첫번째 객체 생성

        myCar.setSpeed(100); // 정수 버전 메소드 호출
        myCar.setSpeed(79.2); // 실수 버전 메소드 호출
    }
}
```

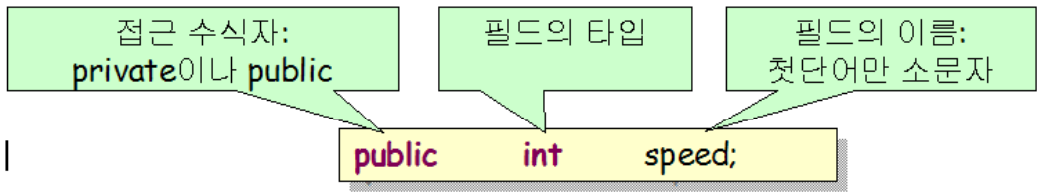
```
정수 버전 호출
실수 버전 호출
```


자바에서의 변수의 종류

- 필드(field) 또는 인스턴스 변수: 클래스 안에서 선언되는 멤버 변수
- 지역 변수(local variable): 메소드나 블록 안에서 선언되는 변수



필드의 선언



필드의 접근 수식자는 어떤 클래스가 필드에 접근할 수 있는지를 표시한다.

- public : 이 필드는 모든 클래스로부터 접근가능하다.
- private : 클래스 내부에서만 접근이 가능하다.

필드의 사용 범위

Date.java

```
public class Date {

        public void printDate() {
                System.out.println(year + "." + month + "." + day);
        }

        public int getDay() {
                return day;
        }

        // 필드 선언
        public int year;
        public String month;
        public int day;
}
```

선언 위치와는 상관없이 어디서나 사용이 가능하다.

설정자와 접근자

- 설정자(mutator)
  - 필드의 값을 설정하는 메소드
  - setXXX() 형식
- 접근자(accessor)
  - 필드의 값을 반환하는 메소드
  - getXXX() 형식

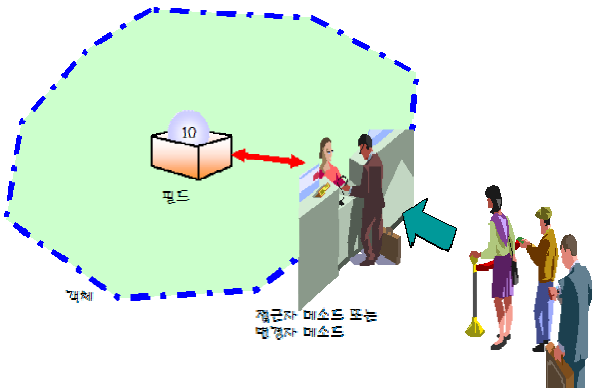


그림 8.12 접근자와 변경자 메소드만을 통하여 필드에 접근한다.

설정자와 접근자의 예

CarTest2.java

```
class Car {

        // 필드 선언
        private int speed; // 속도
        private int mileage; // 주행거리
        private String color; // 색상

        // 접근자 선언
        public int getSpeed() {
                return speed;
        }

        // 설정자 선언
        public void setSpeed(int s) {
                speed = s;
        }
```

설정자와 접근자의 사용

```
public class CarTest2 {
        public static void main(String[] args) {
                // 객체 생성
                Car myCar = new Car();

                // 설정자 메소드 호출
                myCar.setSpeed(100);
                myCar.setMileage(0);
                myCar.setColor("red");

                // 접근자 메소드 호출
                System.out.println("현재 자동차의 속도는 " + myCar.getSpeed());
                System.out.println("현재 자동차의 주행거리는 " + myCar.getMileage());
                System.out.println("현재 자동차의 속도는 " + myCar.getColor());
        }
}
```


설정자와 접근자는 왜 사용하는가?

- 설정자에서 매개 변수를 통하여 잘못된 값이 넘어오는 경우, 이를 사전에 차단할 수 있다.
- 필요할 때마다 필드값을 계산하여 반환할 수 있다.
- 접근자만을 제공하면 자동적으로 읽기만 가능한 필드를 만들 수 있다.

```
public void setSpeed(int s)
{
        if( s < 0 )
                speed = 0;
        else
                speed = s;
}
```

필드의 초기화

- 필드값은 선언과 동시에 초기화 될 수 있다.

```
public class ClassRoom {
        public static int capacity = 60; //60으로 초기화
        private boolean use = false;  // false로 초기화
}
```

- 생성자를 사용하는 방법 -> 다음 장에서 학습

주의

지역 변수는 사용하기 전에 반드시 초기화를 하여야 한다. 자바에서는 지역 변수를 선언하고 초기화하지 않으면 오류가 발생한다.

```
class BugProgram {
   ...
   public int getAverage(int x, int y)
   {
        int sum;
        sum += x;          // 초기화 되지 않은 지역 변수를 사용하면 오류!
        sum += y;
        return sum/2;
   }
}
```

실행결과

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        The local variable sum may not have been initialized
        at BugProgram.getAverage(BugProgram.java:6)
        at Test.main(Test.java:5)
```

변수와 변수의 비교

- "변수1 == 변수2"의 의미

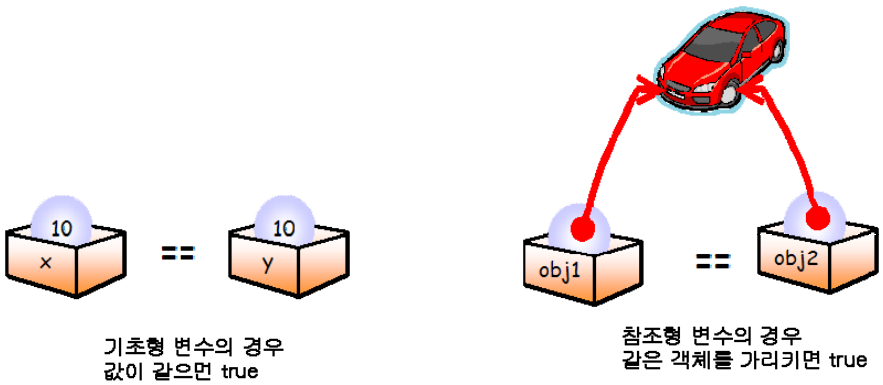


그림 8.14 변수의 비교

- 참조형 변수의 경우, 객체의 내용이 같다는 의미가 아니다.

UML

- UML(Unified Modeling Language)

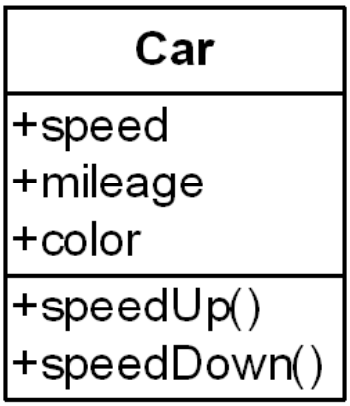


그림 8.15 UML의 예

클래스와 클래스의 관계

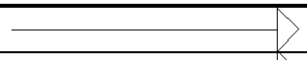
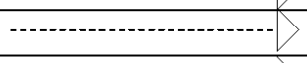
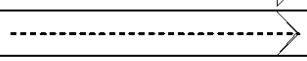
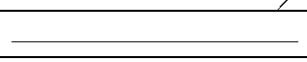
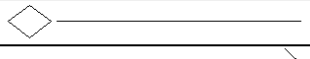
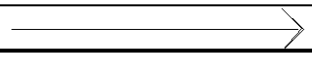
상속(inheritance)	
인터페이스 상속(interface inheritance)	
의존(dependency)	
집합(aggregation)	
연관(association)	
유향 연관(direct association)	

그림 8.16 UML에서 사용되는 화살표의 종류

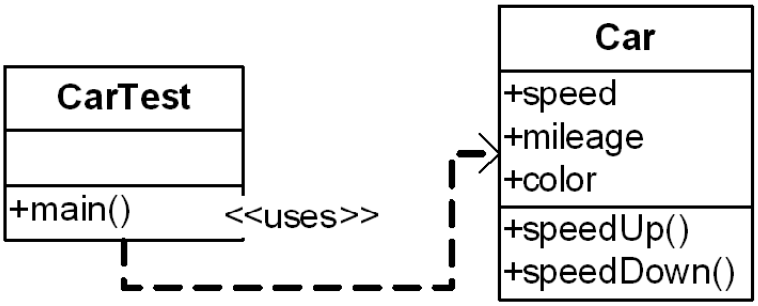
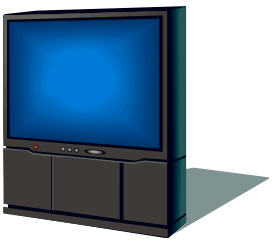


그림 8.17 Car 예제의 UML

중간점검

1. TV를 나타내는 클래스를 정의하고 UML의 클래스 다이어그램으로 표현하여 보라.

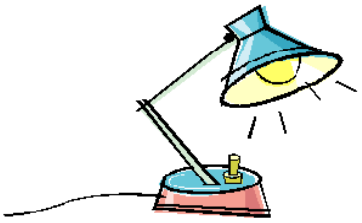


Television
-isOn -volume -channel
+setChannel() +getChannel() +setVolume() +getVolume() +turnOn() +turnOff() +toString()

예제

집에서 사용하는 데스크 램프를 클래스로 작성하여 보면 다음과 같다.

DeskLamp
-isOn : bool
+turnOn() +turnOff()



예제

```
class DeskLamp {
    // 인스턴스 변수 정의
    private boolean isOn; // 켜짐이나 꺼짐과 같은 램프의 상태

    // 메소드 정의
    public void turnOn() // 램프를 켠다.
    {
        isOn = true;
    }

    public void turnOff() // 램프를 끈다.
    {
        isOn = false;
    }

    public String  toString() {
        return "현재 상태는 " + (isOn == true ? "켜짐" : "꺼짐");
    }
}
```

예제

```
public class DeskLampTest {
    public static void main(String[] args) {
        // 역시 객체를 생성하려면 new 예약어를 사용한다.
        DeskLamp myLamp = new DeskLamp();

        // 객체의 메소드를 호출하려면 도트 연산자인 .을 사용한다.
        myLamp.turnOn();
        System.out.println(myLamp);
        myLamp.turnOff();
        System.out.println(myLamp);

    }
}
```

```
현재 상태는 켜짐
현재 상태는 꺼짐
```


Q & A