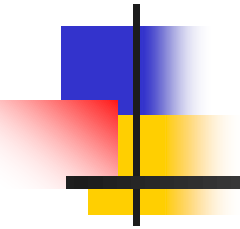


생성자와 접근제어

경복대학교
소프트웨어융합과
배희호 교수

Constructor

- Constructor는 실제로 Object를 생성하는 Method는 아니며, Object를 초기화하는 특수 Method 임
- Object가 생성될 때 Field에게 초기값을 제공하고, 필요한 초기화 절차를 실행하고, Heap에 Object를 저장하는 Method
- Object가 생성되는 순간에 자동 호출
- 구성자라고도 함

기본 객체 생성자 초기화된 객체

생성자의 역할

Constructor

- Constructor는 Class로부터 Object가 생성될 때 Object의 초기화 과정을 기술하는 특수한 Method
- Method와 비슷하지만 Object가 생성될 때마다 무조건 한번 수행된다는 특징을 가지고 있음
- Constructor는 Program에 의해 명시적으로 호출되지 않고 Object를 생성하는 new 연산자에 의해 자동으로 한번만 실행 됨
- 주로 Object 변수에 대한 초기화 작업을 하거나 Memory 할당을 함
- 초기화는 주로 Object 변수를 초기화할 필요가 있을 때 사용
- Constructor 이름은 반드시 Class 이름과 동일하여야 함
- 예)

```
Scanner input = new Scanner(System.in);
```

Constructor

- Constructor 정의

형식

```
public class Car {
    Car() {
        ...
    }
}
```

클래스 이름과 동일한 메소드가 바로 생성자입니다. 여기서 객체의 초기화를 담당합니다.

- Constructor의 이름은 Class의 이름과 반드시 같아야 함
- Method와 비슷한 구조를 가지지만 다른 점
  - 반환값(return value)을 가지지 않음(void를 쓰지 않음)
  - 주로 Field에 초기값을 부여할 때 많이 사용되지만 특별한 초기화 절차를 수행할 수도 있음

Constructor

- Constructor의 형식

```
[public/protected/private] 클래스 이름(형식 매개 변수 리스트) {
    ……
}

클래스 이름(형식 매개 변수 리스트) {
    다른 객체 생성자 호출;
        // 반드시 첫번째 줄에서 이루어져야 함
}
```

```
Class 이름 Object 이름
            = new Class 이름(실 매개 변수 리스트);
```

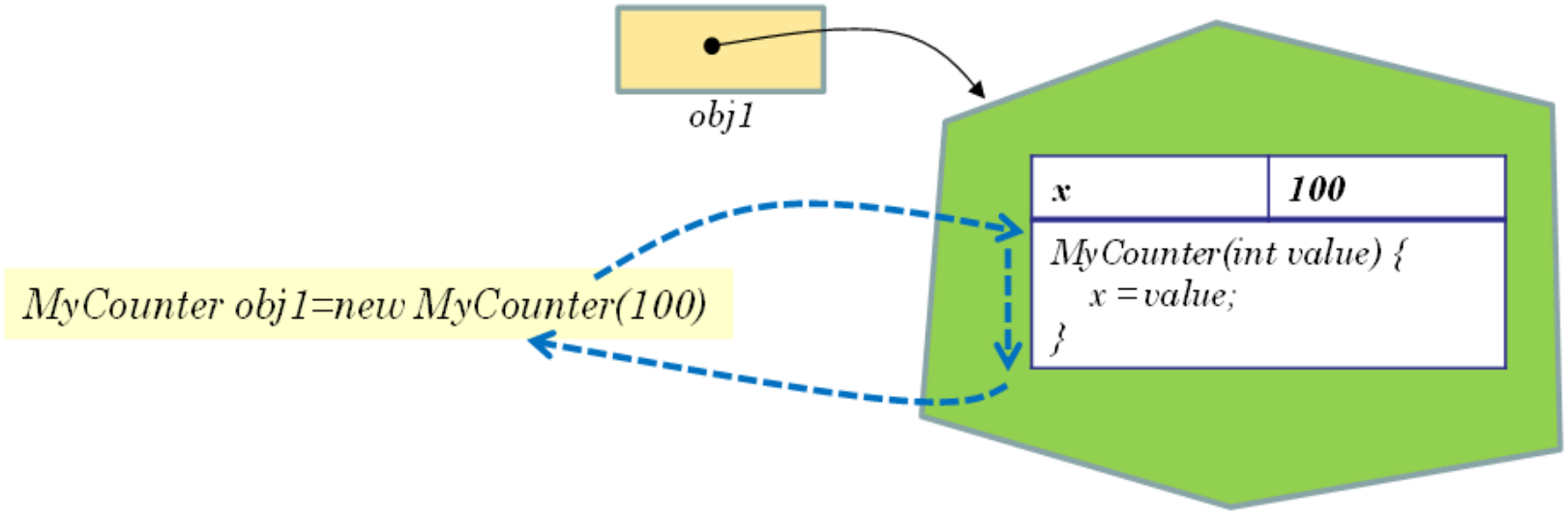
Constructor

- Method 다중 정의에 의해 같은 이름을 갖는 Method가 여러 개 존재할 수 있듯이 Constructor도 여러 개 존재 가능
  - Constructor는 Constructor가 갖는 매개 변수의 개수와 Data Type을 이용하여 서로 구별
- 접근 한정자의 의미는 멤버 변수의 접근 한정자와 같음
  - 반드시 public 이어야 함
  - Constructor가 private로 선언되는 경우는 Class 내부에서만 사용한다는 의미

Constructor

- Constructor의 특징
  - Constructor는 객체 생성시 한 번만 호출
    - JAVA에서 객체 생성은 반드시 new 연산자로 함

```
Circle pizza = new Circle(10, "자바피자");
                                    // 생성자 Circle(int r, String n) 호출
Circle donut = new Circle();        // 생성자 Circle() 호출
```



Constructor

- Constructor의 특징
  - 주로 Object가 가지는 변수에 대한 초기화 작업 또는 Memory(Heap)할당 등의 작업을 수행
  - new 연산자를 이용하여 Object를 생성할 때 자동으로 호출되며, 이 때 Object를 위한 Memory를 할당하고, 다음으로 Object 생성자를 호출

```
Circle pizza = new Circle(10, "자바피자");
                                    // 생성자 Circle(int r, String n) 호출
Circle donut = new Circle();     // 생성자 Circle() 호출
```

  - Method와 비슷한 구조를 갖지만, 다른 점은 Object가 생성될 때 자동으로 호출되며, 반환 값을 갖지 않으며, 메소드와 같이 직접 호출할 수 없음

오류

```
public void Circle() {...}          // 오류 void도 사용 안 됨
```

Constructor

- Default Constructor
  - 만약 Class 작성시에 Constructor를 하나도 만들지 않는 경우에는 JAVA Compiler는 자동적으로 Constructor의 몸체 부분이 비어있어 아무런 작업도 하지 않는 기본(디폴트) 생성자가 만들어짐
  - Constructor가 하나라도 있으면 Compiler는 Default Constructor를 추가하지 않음

```
class Circle {
        public Circle() {          // 기본 생성자

        }
}
```

Constructor

- Default Constructor가 자동 생성되는 경우
  - Class에 Constructor가 하나도 선언되어 있지 않을 때
  - Compiler에 의해 Default Constructor 자동 생성

```
class Car {
        private String color;      // 색상
        private int speed;         // 속도
        private int gear;          // 기어
}
```

Compiler가 디폴트 생성자를 자동으로 생성

```
public class CarTest {
    public static void main(String args[]) {
        Car car = new Car();          // 디폴트 생성자 호출
    }
}
```

Constructor

- Default Constructor가 자동 생성되지 않는 경우
  - Class에 Constructor가 하나라도 선언되어 있는 경우
  - Compiler는 Default Constructor를 자동 생성하지 않음

```
class Car {
    private String color;     // 색상
    private int speed;        // 속도
    private int gear;         // 기어
    public Car(String color, int speed, int gear) {    // 생성자
            this.color = color;
            this.speed = speed;
            this.gear = gear;
    }
}
public class CarTest {
    public static void main(String args[]) {
        Car car = new Car();
    }
}
```

오류

>>> 주의 <<<
생성자가 하나라도 정의되면, 디폴트 생성자는 자동 정의되지 않으므로 디폴트 생성자를 통한 객체 생성은 오류!!!

Constructor Overloading

- Method처럼 Constructor도 중복(Overloading)될 수 있음
- Class에 하나 이상의 생성자를 중복하여 사용할 수 있음
- 여러 개의 생성자를 사용할 때는 생성자의 이름은 같지만 매개 변수의 Data Type과 개수는 달라야 함
- 만일 한 Class에 같은 매개 변수를 가진 생성자를 2개 이상 사용하면 Error 발생
- Object를 생성할 때 Keyword new와 같이 명시한 인자와 동일한 인자의 수와 유형을 가진 생성자를 호출하여 실행함

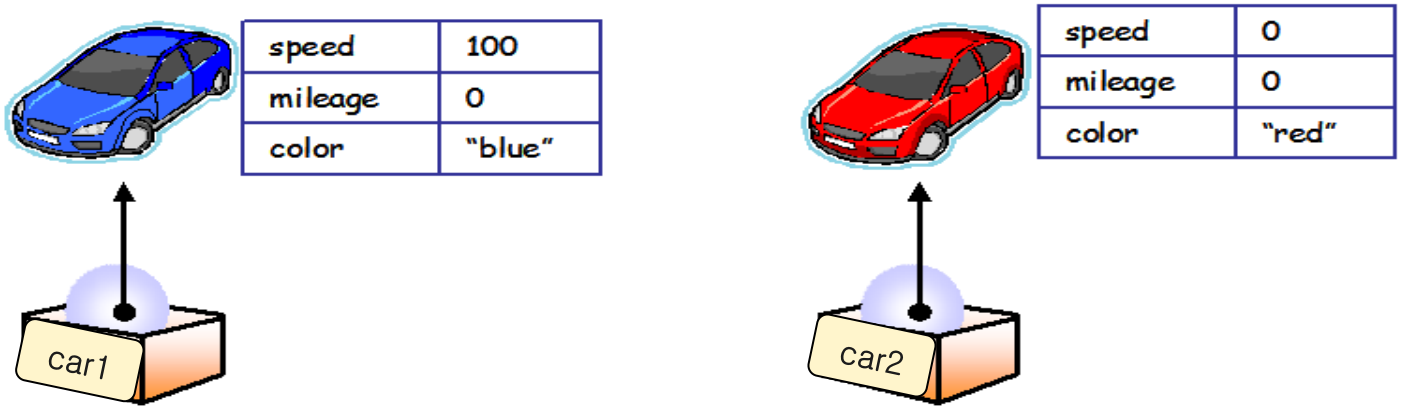
```
public class Circle {
   public Circle() {...}                    // 매개 변수 없는 생성자
   public Circle(int r, String n) {...} // 2개의 매개변수를 가진 생성자
}
```

Constructor Overloading

```
class Car {
    private String color;       // 색상
    private int speed;          // 속도
    private int gear;           // 기어

    public Car(String color, int speed, int gear) {  // 첫 번째 생성자
        this.color = color;
        this.speed = speed;
        this.gear = gear;
    }

    public Car() {        // 두 번째 생성자
        color = "red";
        speed = 0;
        gear = 1;
    }
}
```

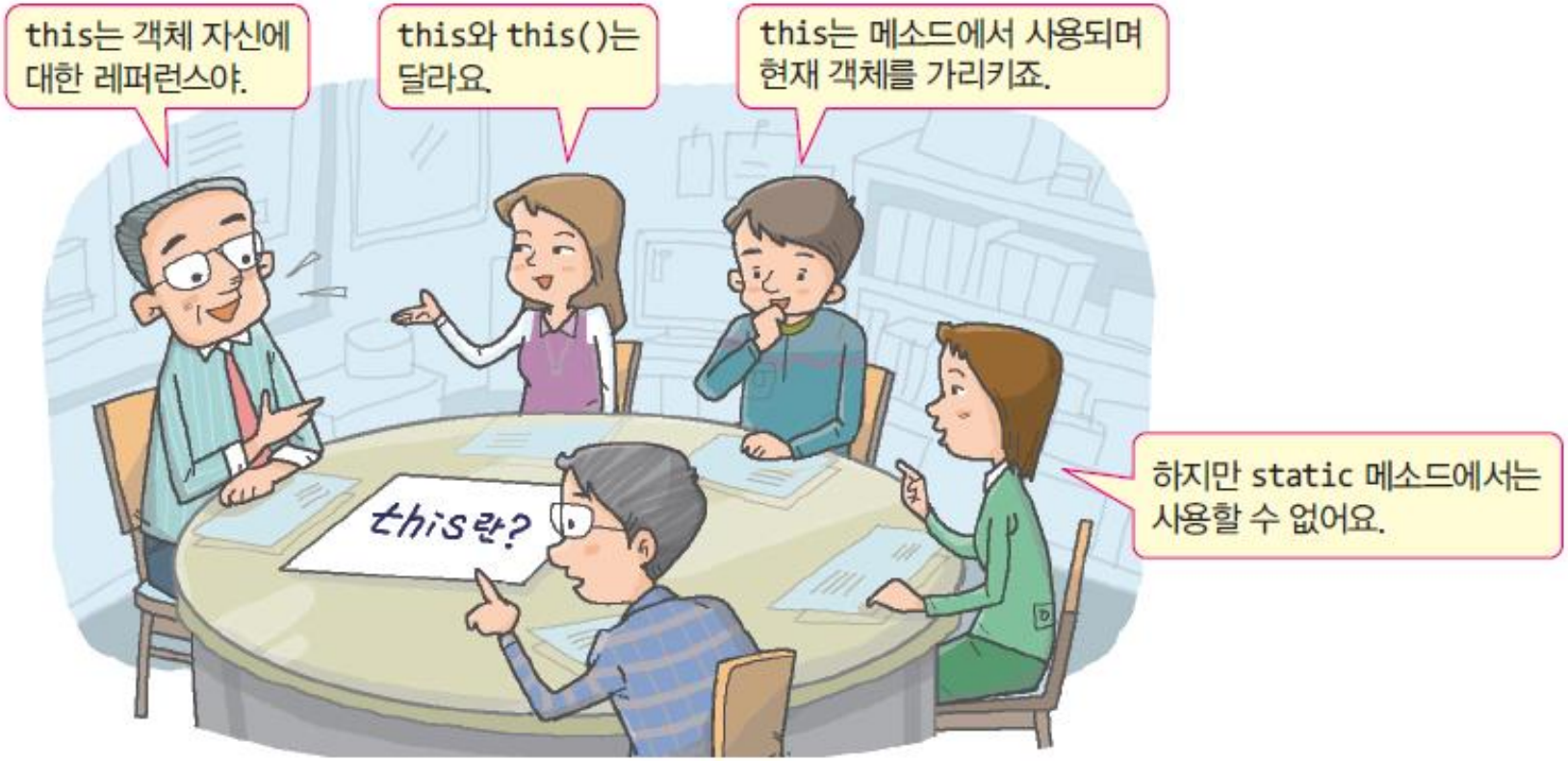
Constructor Overloading

```
public class CarTest {
    public static void main(String args[]) {
        Car car1 = new Car("blue", 100, 0);   // 첫 번째 생성자 호출
        Car car2 = new Car();                 // 두 번째 생성자 호출
    }
}
```



생성자를 통한 객체의 초기화

참조변수 this



참조변수 this

- this는 자신 객체를 의미함
  - "this."를 이용하면 동일 클래스 내의 변수 또는 메소드를 참조할 수 있고, "this()"를 사용하면 자신의 생성자를 호출 할 수 있음

- super는 부모 객체를 의미함
  - "super."를 이용하면 자식 클래스에서 부모 클래스의 변수 또는 메소드를 참조할 수 있으며, "super()"를 이용하면 부모 클래스의 생성자를 호출할 수 있음

참조변수 this

- C++와 유사하게 JAVA에서도 this Keyword를 지원함
- this는 현재 실행중인 객체를 참조하는데 사용됨
- Compiler에 의해 자동 관리
  - 개발자는 사용하기만 하면 됨
- 객체는 실행 Code와 객체 변수로 구성됨
  - 객체의 실행 Code는 변경되지 않는 부분이므로, 모든 객체가 공유하는 부분
  - 객체 변수는 각 객체마다 할당 되어야 함
- 객체마다 할당되어 있는 객체 변수를 사용하기 위해서 다음과 같이 this Keyword를 사용 가능

```
this.var_Name
```

  - var_Name은 객체 변수의 이름

참조변수 this

- 동일한 이름의 메소드의 인자와 메소드 지역 변수에 의해서 객체 변수가 가려질 경우에 객체 변수를 참조하기 위해서는 this를 사용함

객체 속성변수

생성자 매개변수

```
class triangle {
    int width;
    int height;

    triangle(int width, int height) { // 인자가 객체 변수 이름 동일
        this.width = width;
        this.height = height;
    }
}
```

Instance **변수와 지역 변수를 구별하기 위해 참조 변수 this 사용**

- 위의 경우 동일한 이름을 사용하였기 때문에 생성자의 인수가 클래스의 객체 변수를 가리키고 있는 경우
- 기존의 명시 방법으로는 객체 변수를 참조할 수 없음

참조 변수 this

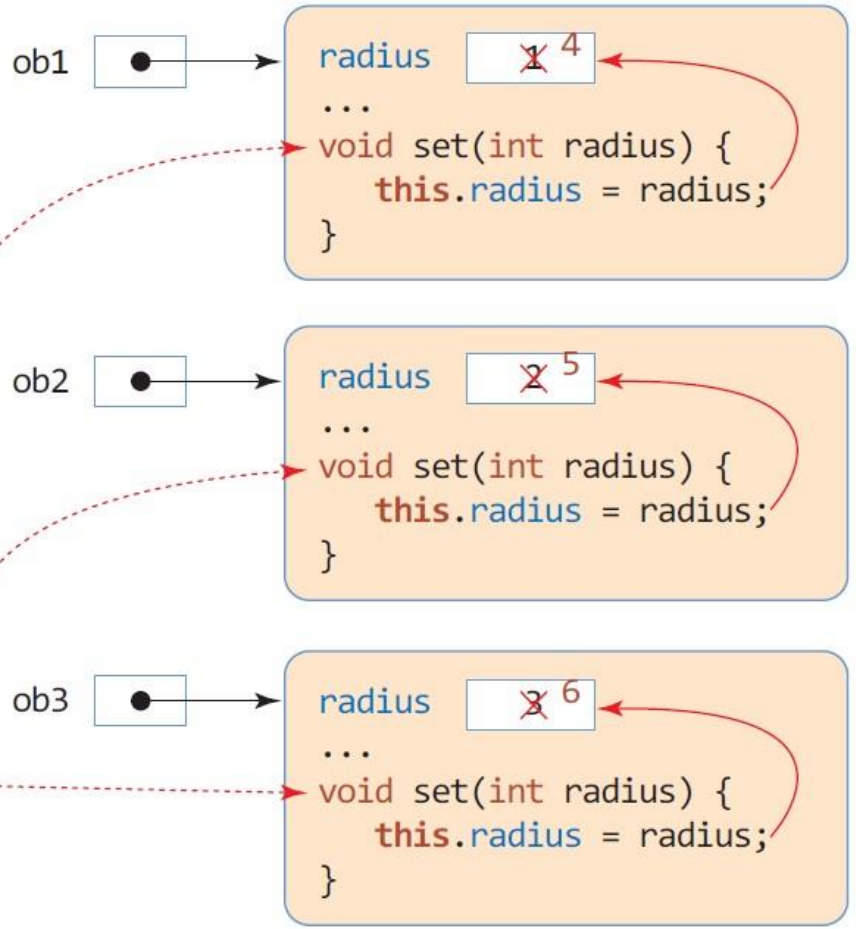
- 객체 자신에 대한 레퍼런스
  - 컴파일러에 의해 자동 관리, 개발자는 사용하기만 하면 됨
  - this.멤버 형태로 멤버를 접근할 때 사용

```
public class Circle {
   private int radius;

public Circle() {
    radius = 1;
}
public Circle(int r) {
    radius = r;
}
double getArea() {
return 3.14*radius*radius;
}
…
}
```

```
public class Circle {
   private int radius;

public Circle() {
    radius = 1;
}
public Circle(int radius) {
    this.radius = radius;
}
double getArea() {
return 3.14*radius*radius;
}
…
}
```

객체 속에서의 this

```
public class Circle {
   int radius;
   public Circle(int radius) {
      this.radius = radius;
   }
   void set(int radius) {
      this.radius = radius;
   }

   public static void main(String[] args) {
      Circle ob1 = new Circle(1);
      Circle ob2 = new Circle(2);
      Circle ob3 = new Circle(3);

      ob1.set(4);
      ob2.set(5);
      ob3.set(6);
   }
}
```



this() 메소드

- 모든 클래스에는 반드시 하나 이상의 생성자가 있어야 함
- 생성자가 여러 개 정의되어 있을 때, 필요에 따라 하나의 생성자에서 다른 생성자를 호출 할 수 있는데, 이 때 this keyword를 사용함
- 상속 관계에 있는 자식 클래스에서 부모 클래스의 생성자를 호출할 때는 super 키워드를 사용함
- 어떠한 클래스를 상속받으면 그 클래스(부모 클래스)의 생성자(생성자가 선언/정의 되어 있는 경우)는 상속되지 않음
- 자식 클래스가 Instance화 될 때 부모 생성자가 반드시 실행되어야 하는데, 이 때 사용하는 Keyword가 super
- 다른 생성자의 호출은 반드시 첫 번째 줄에 나타나야 함

this() 메소드

- 생성자도 메소드이기 때문에 다른 메소드를 호출할 수 있음
- this() 메소드는 생성자 내부에서만 사용할 수 있으며, 같은 클래스의 다른 생성자를 호출할 때 사용
- this() 메소드에 인수를 전달하면, 생성자 중에서 메소드 시그니처가 일치하는 다른 생성자를 찾아 호출 함
- 생성자 Overloading되면 생성자 간의 중복된 Code 발생
  - 초기화 내용이 비슷한 생성자들에서 이러한 현상 발생
  - 초기화 내용은 한 생성자에게 몰아서 작성
  - 다른 생성자는 초기화 내용을 작성한 생성자를 this(....)로 호출
- Code의 재 사용성을 높인 Code

this() 메소드

```
public class Car {
	private String company = "현대자동차";    //필드
	private String model;
	private String color;
	private int maxSpeed;

	public Car(String model) {    // this()를 통해서 중복 부분을 간단히 작성
		this(model, null, 0);
	}

	public Car(String model, String color) {
		this(model, color, 0);
	}

	public Car(String model, String color, int maxSpeed) {
		this.model = model;
		this.color = color;
		this.maxSpeed = maxSpeed;
	}
}
```

this() 메소드

```
public class Book {
    private String title;
    private String author;
    void show() {
        System.out.println(title + " " + author);
    }

    public Book() {
        this("", "");
        System.out.println("생성자 호출됨");
    }
    public Book(String title) {
        this(title, "작자미상");
    }

    public Book(String title, String author) {
         this.title = title;
         this.author = author;
    }
}
```

this() 메소드

```
public static void main(String [] args) {
        Book javaBook = new Book("Java", "황기태");
        Book bible = new Book("Bible");
        Book emptyBook = new Book();

        bible.show();
}
```

this() 메소드

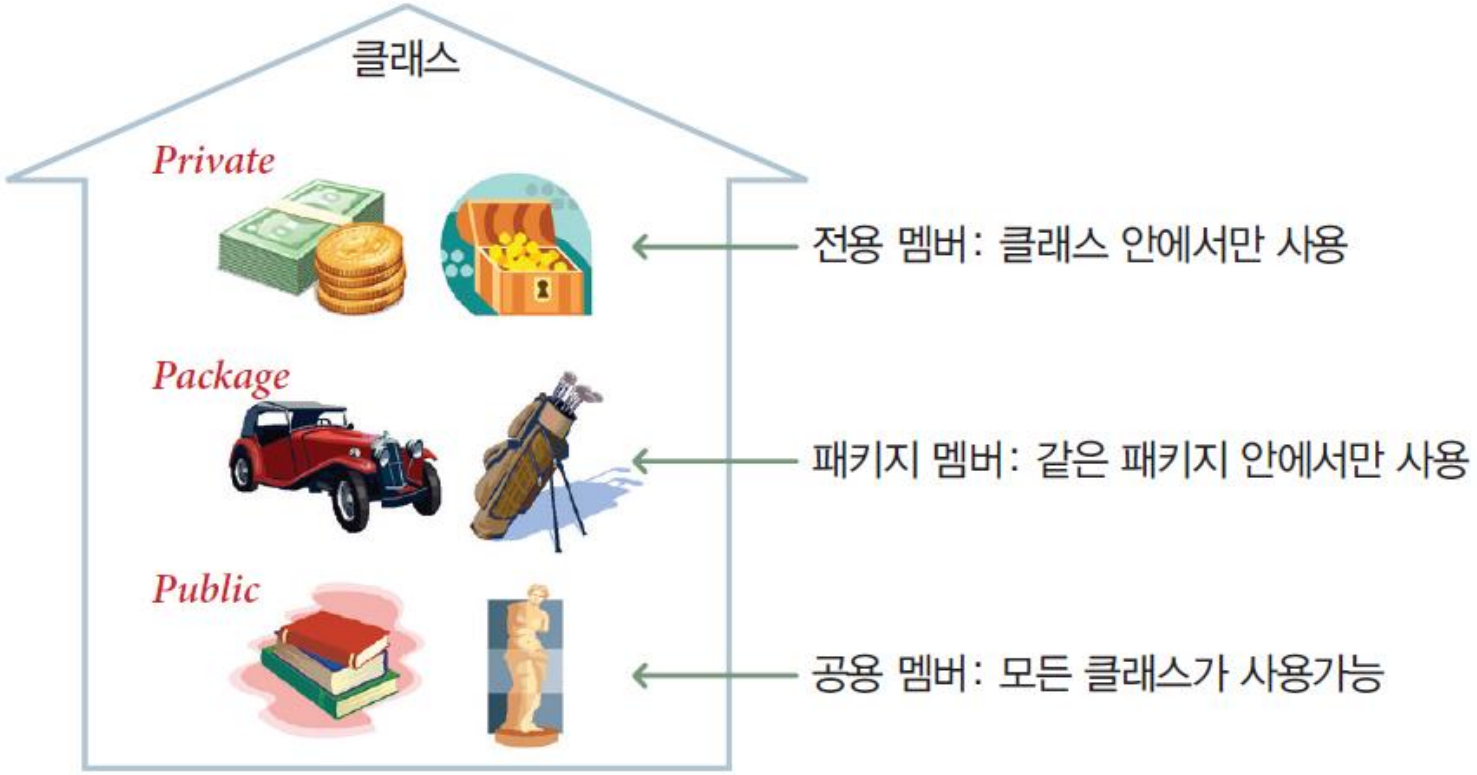
- 다른 생성자 호출은 생성자의 첫 문장에서만 가능
  - 첫 라인에 있지 않으면, Compiler Error를 발생 시킴
  - 생성자 내에서 초기화 작업 도중에 다른 생성자를 호출하면 앞에서 작업한 내용이 모두 다시 초기화 되므로 소용이 없음
  - this() 사용 실패 예

```
public Book() {   // 생성자
    System.out.println("생성자 호출됨");
    this("", "", 0);
    // 생성자의 첫 번째 문장이 아니기 때문에 컴파일 오류
}
```

오류

Access Control

- 다른 Class가 특정한 Field나 Method에 접근하는 것을 제어하는 것



Member 변수에 대한 접근 제어

Access Control

- 종류

public : 모든 클래스에서 접근이 가능

protected : 같은 패키지(폴더)에 있는 클래스와 상속 관계의 클래스들만 접근 가능

default : 같은 패키지에 있는 클래스들만 접근 가능

private : 같은 클래스내에서만 접근 가능

Access Control

- 접근 제어의 종류
  - Class 수준에서의 접근 제어
  - Member 변수 수준에서의 접근 제어

Access Control

- 사용법
  - 클래스 – public , default 만 가능하다
  - 멤버 변수 – 모든 접근 지정자 가능하다.
  - 멤버메소드 – 모든 접근 지정자 가능하다.
  - 생성자 – 모든 접근 지정자가 가능하다.

- 로컬변수에는 접근 지정자를 사용할 수 없다.

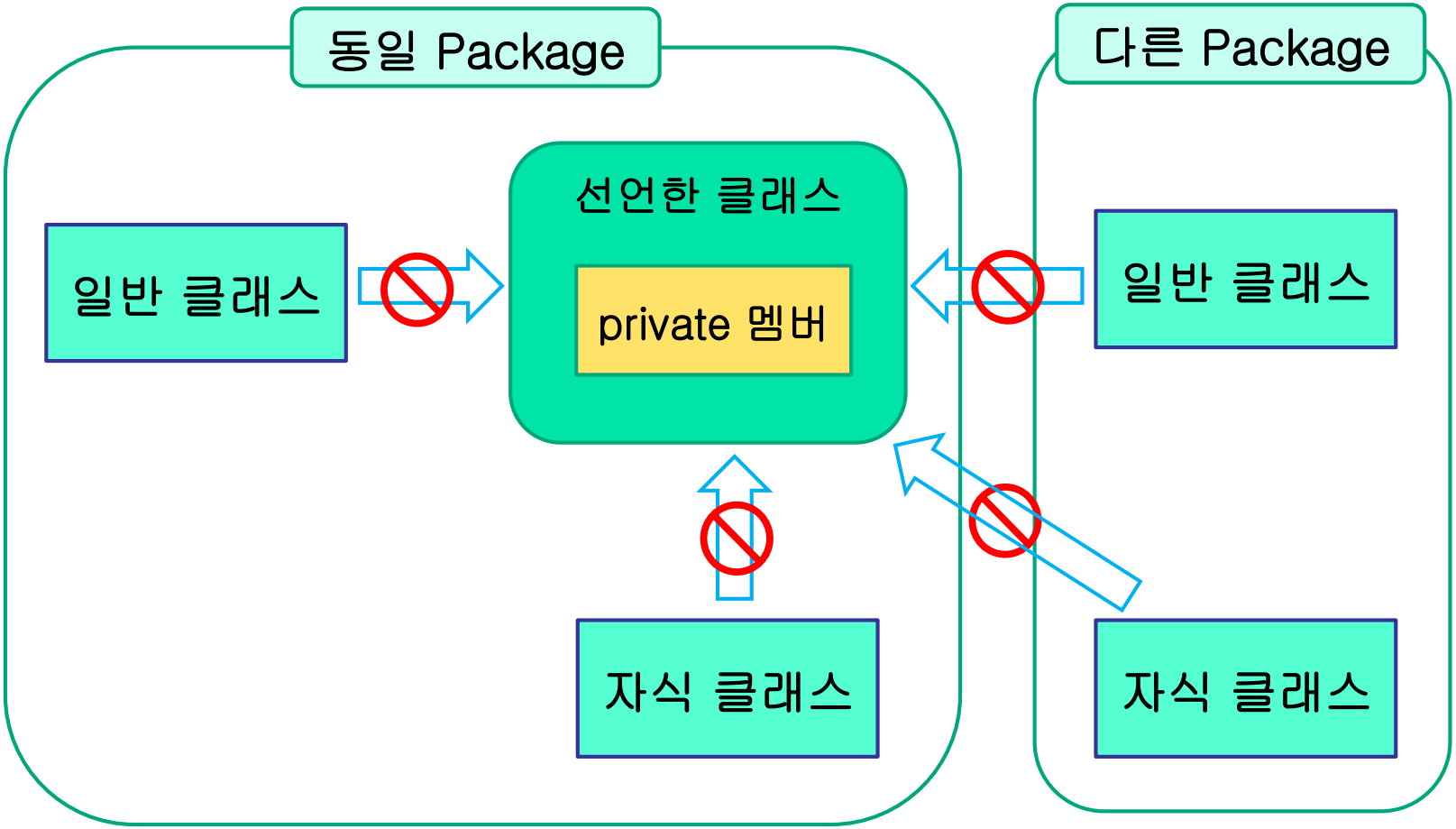
```
1  public class Student {
2
3   public  String name;    //이름
4   protected int grade;        //학년
5   private String address;   //주소
6   String tel;                     //전화번호
7
```

Access Control

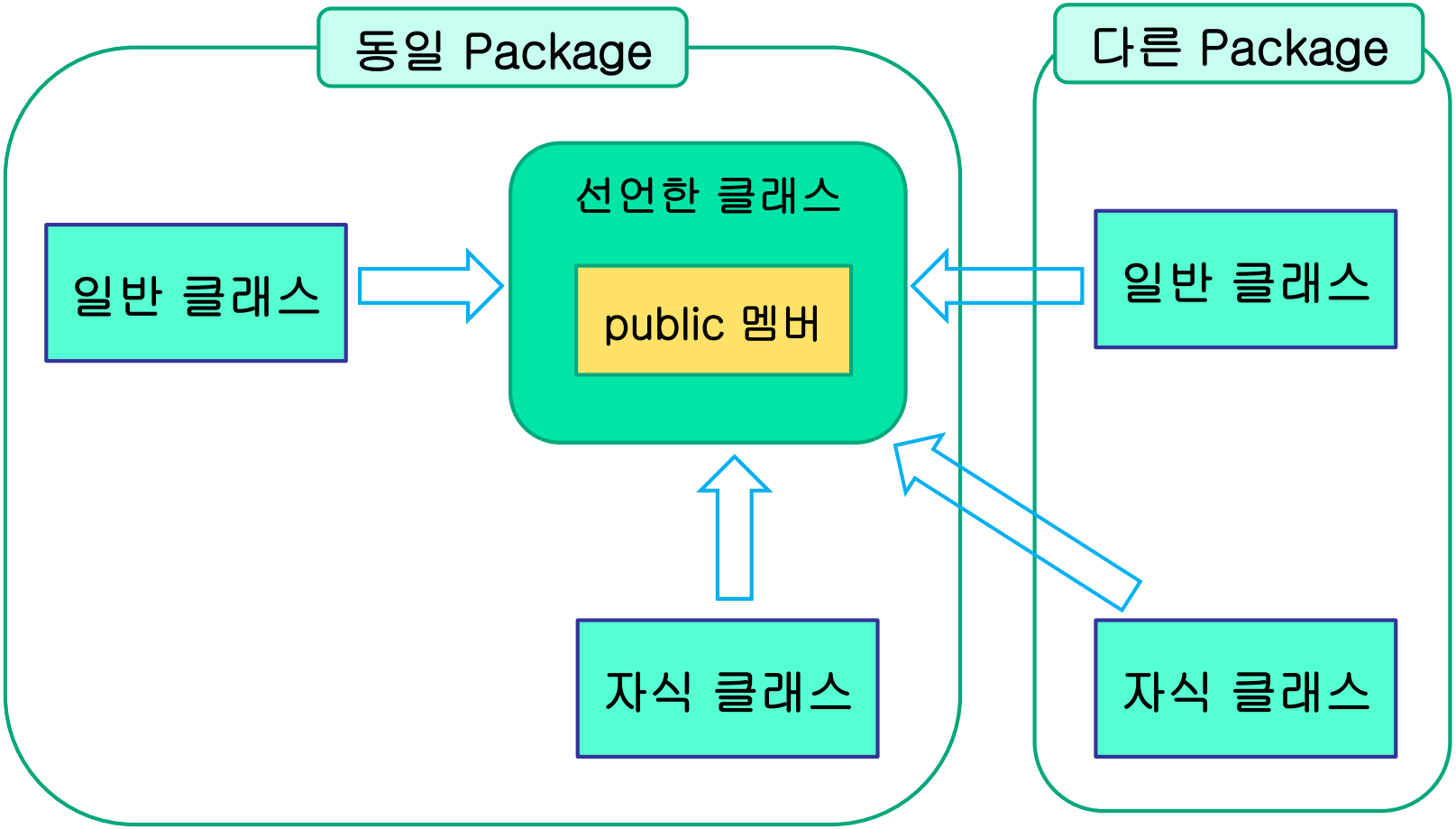
- Class 수준에서의 접근 제어
  - public
    - 다른 모든 Class가 사용할 수 있는 공용 Class
  - package
    - 수식자가 없으면 같은 Package안에 있는 Class들만이 사용
- Member 변수 수준에서의 접근 제어

분류	접근 지정자	클래스 내부	같은 패키지 내의 클래스	다른 모든 클래스
전용 멤버	private	○	X	X
패키지 멤버	없음	○	○	X
공용 멤버	public	○	○	○

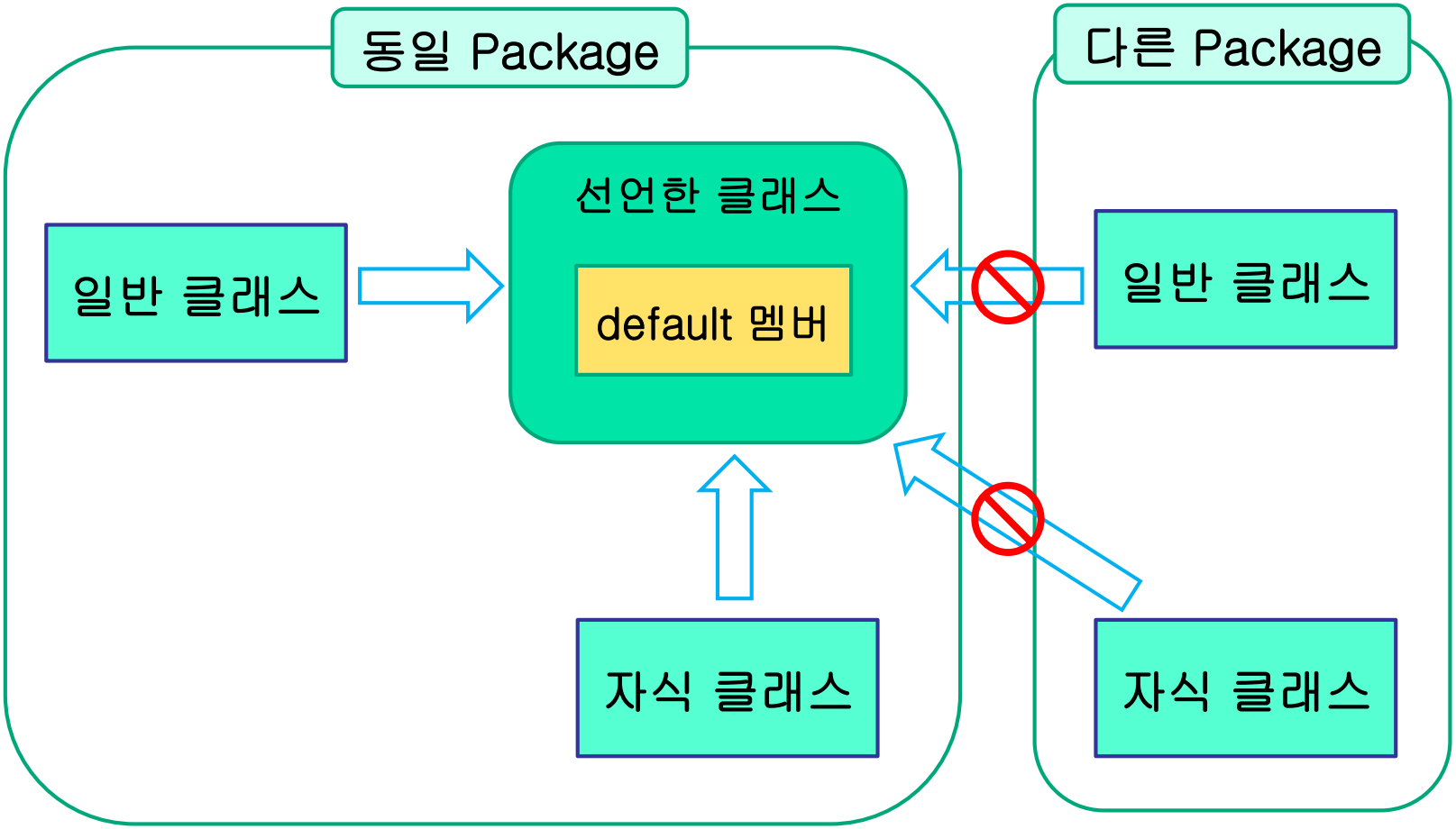
Access Control



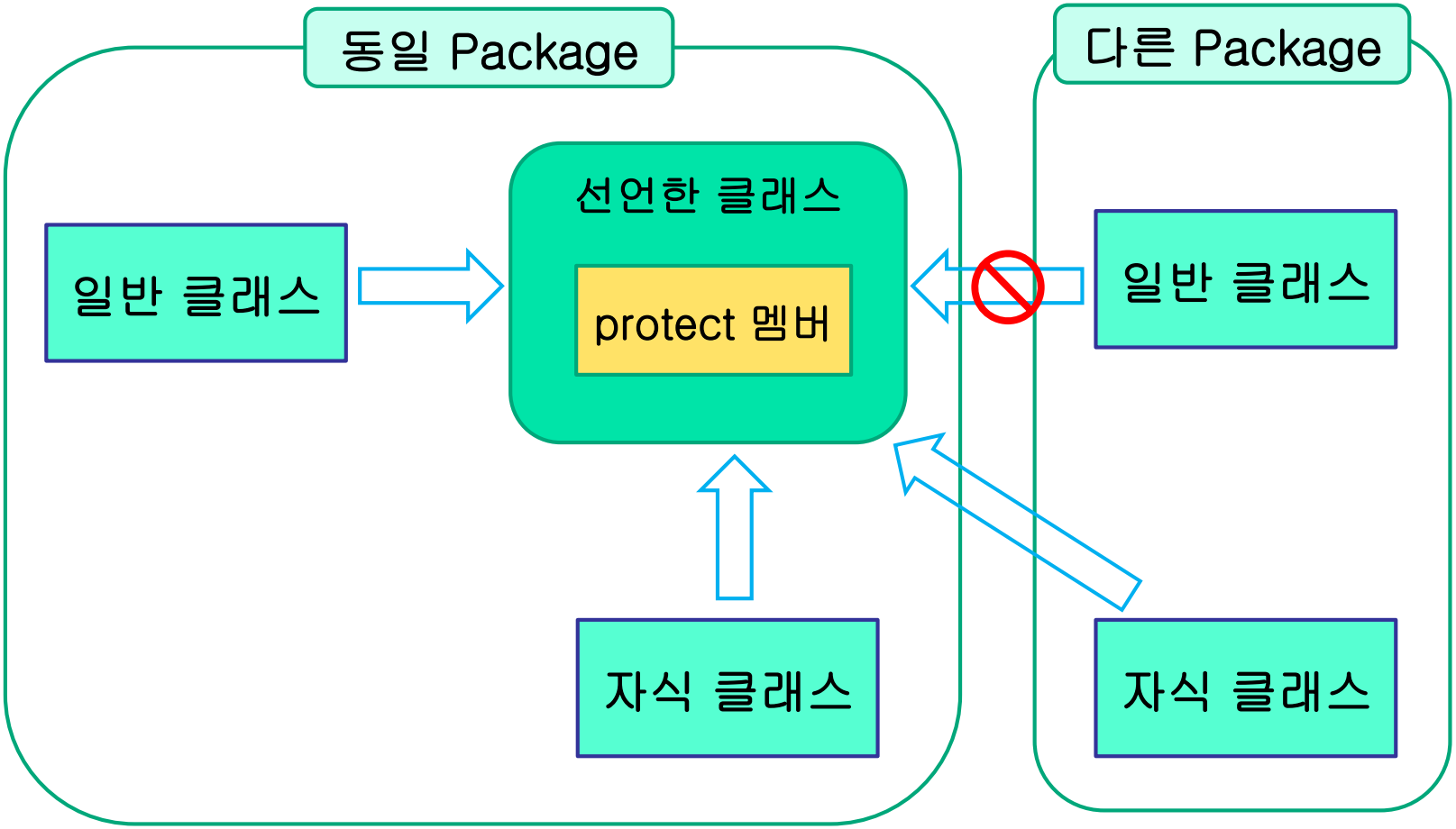
Access Control



Access Control

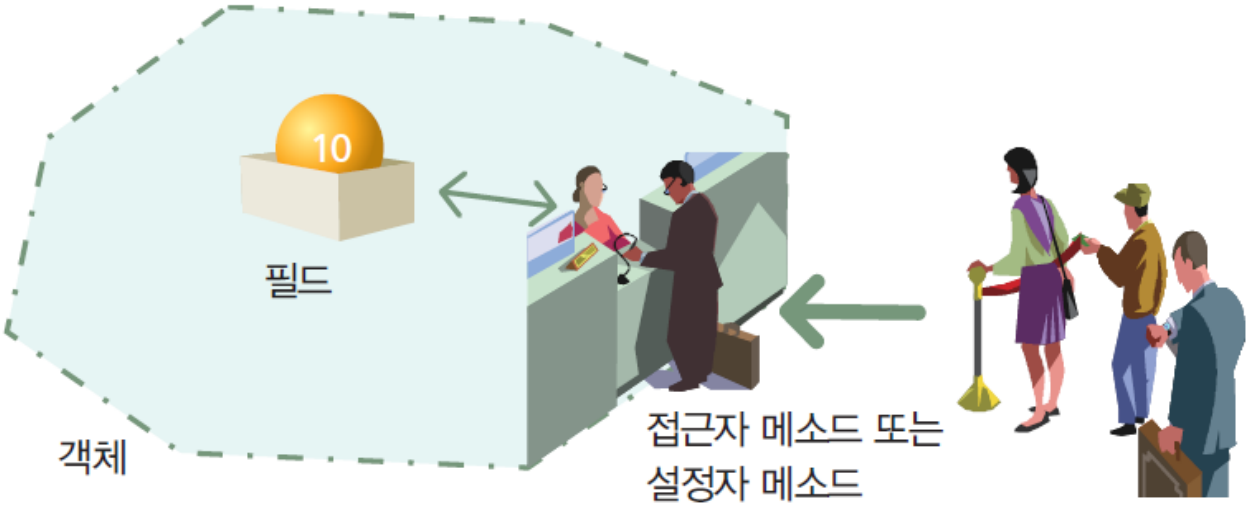


Access Control



Setter와 Getter

- Setters(설정자)
  - Field의 값을 설정하는 메소드
  - setXXX( ) 형식
- Getters(접근자)
  - Field의 값을 반환하는 메소드
  - getXXX() 형식



접근자와 설정자 메소드만을 통하여 필드에 접근

Setter와 Getter

- 설정자(mutator)란?
  - 일반적으로 세터(setter)라고 부르기도 함
  - private 설정된 Member 변수(Field)에 대하여 외부 클래스에서 설정이 불가하기 때문에 Field의 값을 설정하는 기능을 메소드 형태로 해당 클래스에 만들어 놓은 것
  - 특징
    - 설정자 이름은 set + 해당 변수명 (setXXX( ) 형식)
    - 반환 유형은 없음
    - 해당 변수와 같은 Type의 Data를 매개 변수로 받아서 해당 변수에 대입

Setter와 Getter

- 접근자(accessor)란?
  - 일반적으로 게터(getter)라고 부르기도 함
  - private 설정된 Member 변수(Field)에 대하여 외부 클래스에서 접근이 불가하기 때문에 Field의 값을 반환하는 기능을 메소드 형태로 해당 클래스에 만들어 놓은 것
  - 특징
    - 접근자 이름은 get + 해당 변수명(getXXX() 형식)
    - 매개 변수는 없고 반환 Type은 해당 변수와 같은 Type

Setter와 Getter

- 설정자와 접근자는 왜 사용하는가?
  - 접근자와 설정자를 사용해야만 나중에 클래스 Upgrade가 편리함
  - 입력 값에 대한 검증을 할 수 있음
    - 설정자에서 매개 변수를 통하여 잘못된 값이 넘어오는 경우, 이를 사전에 차단할 수 있음
    - 예) 시간을 25시, 나이를 음수로 변경

```
public void setSpeed(int speed) {    // setter
     if (speed < 0)
          this.speed = 0;        // speed < 0이면 speed = 0
     else
          this.speed = speed;
}
```

Setter와 Getter

- 설정자와 접근자는 왜 사용하는가?
  - 필요할 때마다 Field 값을 계산하여 반환할 수 있음
  - 접근자만을 제공하면 자동적으로 읽기만 가능한 Field를 만들 수 있음

Setter와 Getter

```
class Car {
    private String color;     // 색상
    private int speed;        // 속도
    private int gear;         // 기어

    public String getColor() {          // 색상 접근자
        return color;
    }

    public void setColor(String color) {  //색상 설정자
        this.color = color;
    }

    public void setSpeed(int speed) { // 속도 설정자
        if (speed < 0)
            this.speed = 0;
        else
            this.speed = speed;
    }
```

Field가 모두 private로 선언
-> 클래스 내부에서만
사용 가능

Setter와 Getter

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

    public void setGear(int gear) {  //기어 설정자
        this.gear = gear;
    }

   public int getGear() {            //기어 접근자
        return gear;
    }
}
```

Setter와 Getter

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

        myCar.setColor("RED");     // 색상 설정자 사용
        myCar.setSpeed(100);       // 속도 설정자 사용
        myCar.setGear(1);          // 기어 설정자 사용

        System.out.println("내 차의 색상은 "+  myCar.getColor()); //접근자
        System.out.println("내 차의 속도는 "+  myCar.getSpeed());
        System.out.println("내 차의 기어는 "+  myCar.getGear());
    }
}
```