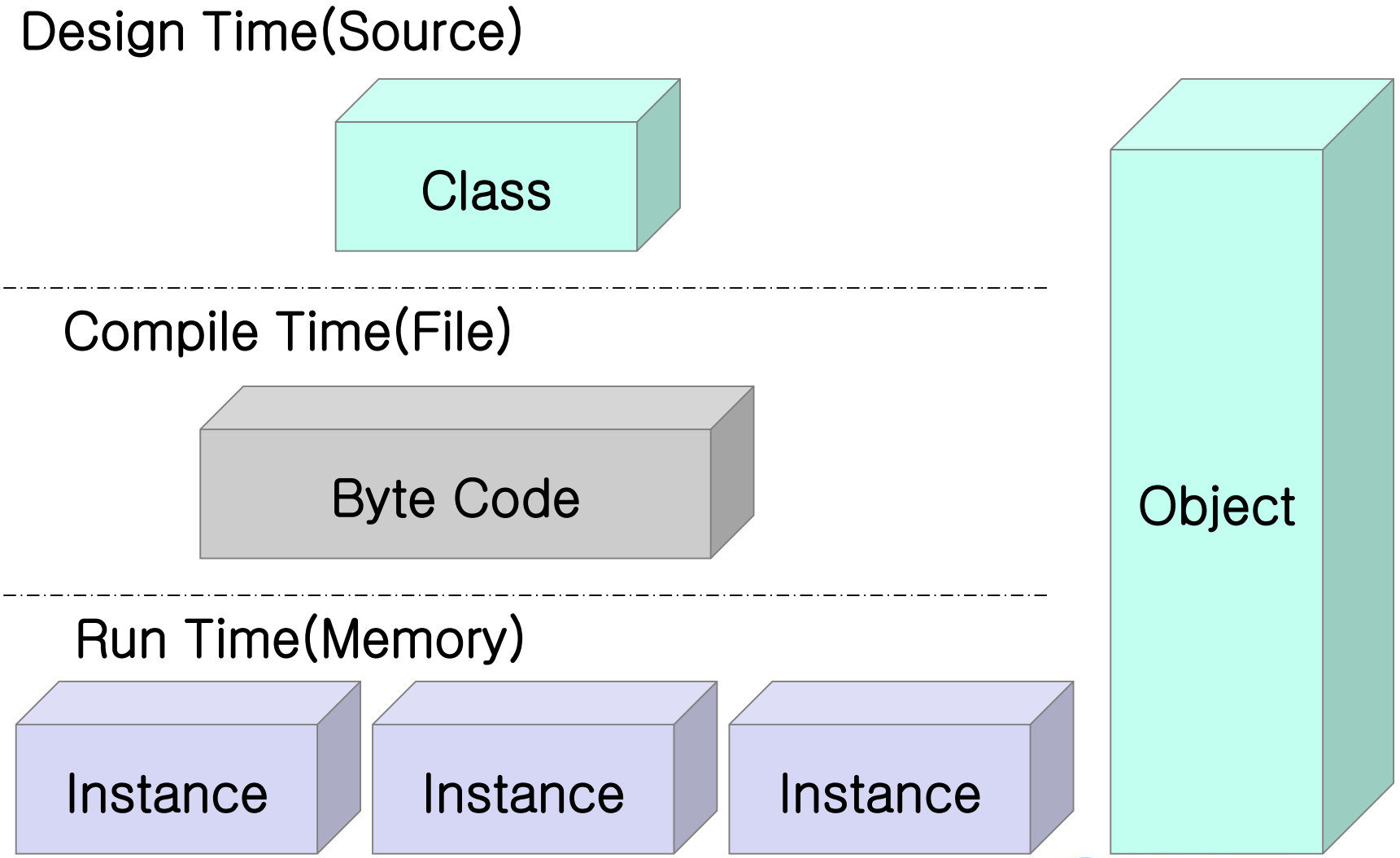
Method 사용하기

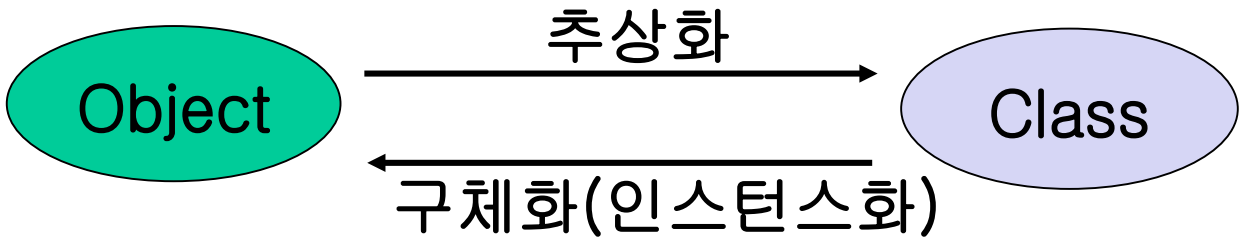
경복대학교
소프트웨어융합과
배희호 교수

Object, Class, Instance

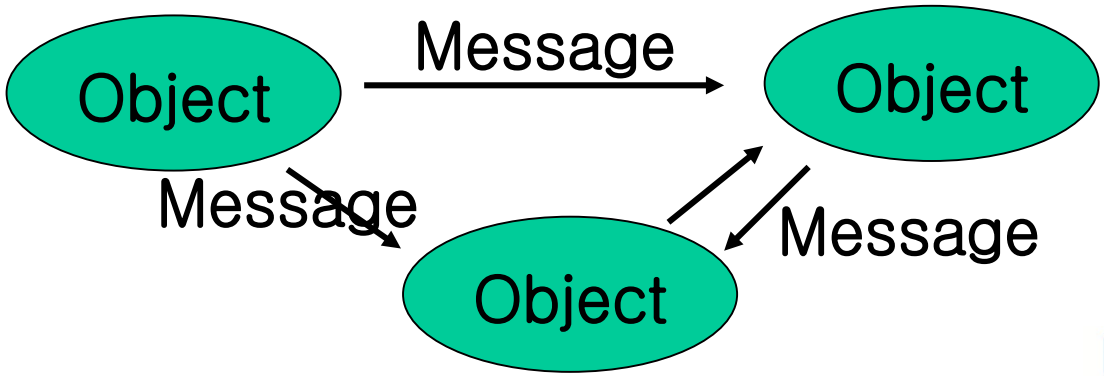


Class

- Object와 Class간의 관계
  - Object는 Class의 Instance : instantiation
  - Object는 Class의 복사물



- Object와 Object간의 관계

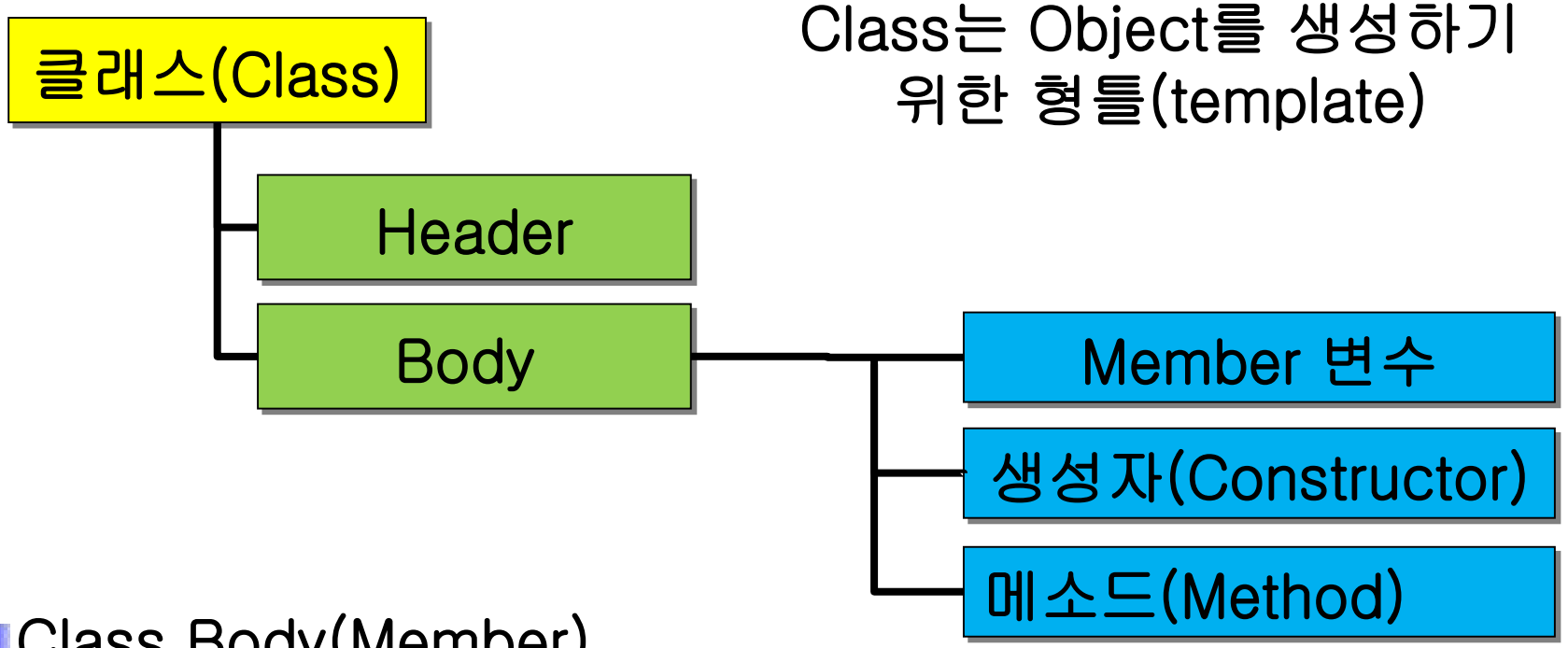


Class

- JAVA Program의 기본 단위
- Object를 생성하기 위해서는 먼저 Class를 작성하여야 함
- Class로부터 Object를 생성하는 방법을 제공
- Attribute 변수(Field)와 그 값을 사용하는 Method들을 정의
- 예) 자동차 Class
  - Attribute 변수(Field)
    - 차량번호, 용도, 적재량 또는 승차인원, 엔진 등
  - Method
    - 차의 용도를 알려주는 Method와 엔진 교환을 표현하는 Method가 필요

Class

- Class의 일반적인 구조



Class는 Object를 생성하기 위한 형틀(template)

- Class Body(Member)
  - Member 변수 = Class가 가지는 State(속성) = Field
  - Constructor = Class의 초기화를 담당
  - Method = Class가 가지는 Data를 조작하고 변환

Class

- JAVA에서는 class라는 Keyword를 사용하여 Class를 정의함 (Encapsulation)

Header

```
[접근 지정자] class 클래스 이름 [extends 부모 클래스 이름]
{
        Member 변수(Field);
        Constructor;
        Method;
}
```

body

Class

- Class의 일반적인 구조

- ✓ Class는 Field와 Method로 이루어짐
- ✓ Field는 Object의 Attribute를 나타냄
- ✓ Method는 Object의 Behavior를 나타냄

Class

- Class Modifier(접근 지정자)
  - 접근 지정자는 해당 Class에 대해 접근 권한 수준을 지정하는 것
  - 접근 지정자는 Class 선언 부분(Header)에 표시
  - Object Oriented Programming에서 불필요한 내부 사정은 외부에 보이지 않도록 하는 Encapsulation(캡슐화)에 해당

- Class Modifier 종류
  - public (package)
  - default (생략한 경우)
  - final
  - abstract

Class

- Class Modifier(접근 지정자)
  - public
    - 공용 Class로서 다른 Package에 있는 Class에서도 접근 가능
    - 하나의 Source File에는 최대 하나의 public Class만 허용
    - Source File의 이름은 반드시 이 Class 이름과 같아야 함
  - default (생략한 경우)
    - default 접근 지정자는 “default”라고 입력하는 것이 아니고, 접근 지정자 부분을 생략하는 것
    - 같은 Package내에 있는 Class들만 접근 허용

Class

- Class Modifier(접근 지정자)
  - final
    - 자식 Class를 만들 수 없는 Class
    - 이 Class로부터 새로운 Class가 상속되어 생성될 수 없음
    - abstract 또는 extends절과는 함께 사용할 수 없음
  - abstract
    - **Abstract Method(**추상 메소드 : 실행문이 없는 **Method)**를 포함하거나 직접 실행되지 않는 **Class**
    - 추상 **Class**로 **Object**를 생성할 수 없음
    - 이 Class는 그 자체가 Instance화 될 수 없으며 반드시 자식 Class가 상속 받아 추상 Method를 구체화 시켜야 함

Class

- Class Modifier(접근 지정자)
  - 다른 Package의 Class에서도 접근을 가능하게 하기 위해서는 반드시 public 접근 지정자를 사용하여야 함
  - JAVA에서는 하나의 Program에는 하나의 Class만을 정의하는 것이 일반적이지만 만일 여러 개의 Class가 하나의 Program에 정의된다면 public 접근 지정자는 한 Class에서만 사용해야 함
    - JAVA 응용 Program인 경우에는 main( ) Method가 있는 Class에 public을 사용해야 함

Class

- Class 이름
  - File 이름과 public Class 이름이 일치해야 함
  - 하나의 File에는 하나의 public Class만 포함할 수 있음
  - 하나의 Class는 하나의 Class File(class 확장자)로 Compile 됨
  - 일반적으로 Class 이름은 대문자로 시작함

Class

- Member 변수(Field)
  - Member 변수는 Class 내의 Method 밖에서 선언된 모든 것을 의미
  - 내부 상태를 표현하기 위해서, Class는 여러 가지 종류의 Member 변수를 포함함
  - Member 변수는 Object가 가질 수 있는 Attribute를 나타내는데 사용
  - Class가 가질 수 있는 변수는 Instance 변수, Class 변수(Static 변수), final 변수 등 있음
  - Instance 변수는 Object Reference 변수와 Object Attribute 변수로 구분됨

Class

- Member 변수 접근 방법
  - Class 변수와 Object Attribute 변수에 접근하기 위해서는 점(“.”) 연산자를 사용

Class 변수 : 클래스 이름.클래스 변수
Object Attribute 변수 : 객체 이름.객체 속성 변수

Class

- Member 변수의 접근 지정자
  - JAVA는 객체 지향의 특성인 캡슐화(Encapsulation)와 정보 은폐(Information Hiding)를 제공하기 위해 Member 변수에 한정자를 사용
  - Member 변수 접근 지정자는 다른 Class에서 변수의 접근 허용 정도를 나타냄
  - C++에서와 마찬가지로 JAVA에서도 접근 지정자를 사용하여 Class 내의 변수에 대한 접근을 제한함

Class

- 접근 지정자의 개념
  - JAVA는 Class 내의 Member의 접근을 제한할 수 있는 방법으로 접근 지정자를 제공
  - 접근 지정자는 Object Oriented Language의 중요한 특성 중에 하나인 캡슐화(Encapsulation)와 정보 은폐(Information Hiding)를 제공
  - 접근 지정자 public, protected, default, private는 Method와 Member 변수들의 접근을 제어하기 위해서 차등 있는 접근 권한을 부여
  - 접근 지정자 없는 Member 변수는 같은 Package 내에서 사용이 가능

Class

- 접근 제한의 목적
  - Object를 작성할 때(Object가 Class로부터 생성되므로 Class를 작성할 때) Program 작성자는 숨겨야 하는 정보(private)와 공개해야 하는 정보(public)를 구분하여 작성 할 수 있는데 Object를 사용하는 사람은 Object 중에 공개하는 정보에만 접근할 수 있음
  - 이러한 기법을 제공함으로써 Object의 사용자로부터 정보를 은폐(Information Hiding)를 할 수 있게 됨

Class

- 접근 권한에 대하여 주목해야 할 부분
  - 한 Method에서 다른 Class의 Member 변수나 Method로의 접근 가능성
  - 한 Class가 다른 Class를 상속받을 때, 각 접근 지정자의 사용에 따라 자식 Class가 부모 Class의 Method 혹은 Member 변수에 접근할 수 있는 권한의 변화

Class

- 생성자 선언
  - 생성자는 Object를 생성할 때 실행
  - Object의 Member 변수(Field)를 초기화하는 데 주로 사용
  - 생성자 이름은 Class 이름과 같음
  - 반환 값을 갖지 않음
  - Parameter List가 서로 다른 여러 개의 생성자가 있을 수 있음 (Overload)

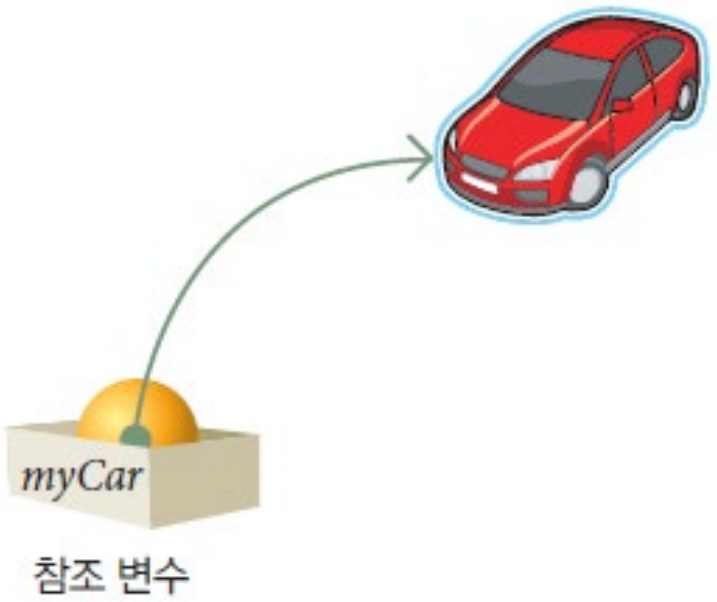
```
public BankAccount();
   // 초기 잔고가 0인 계좌 구성

public BankAccount(double initialBalance);
   // 초기 잔고가 initialBalance인 계좌 구성
```

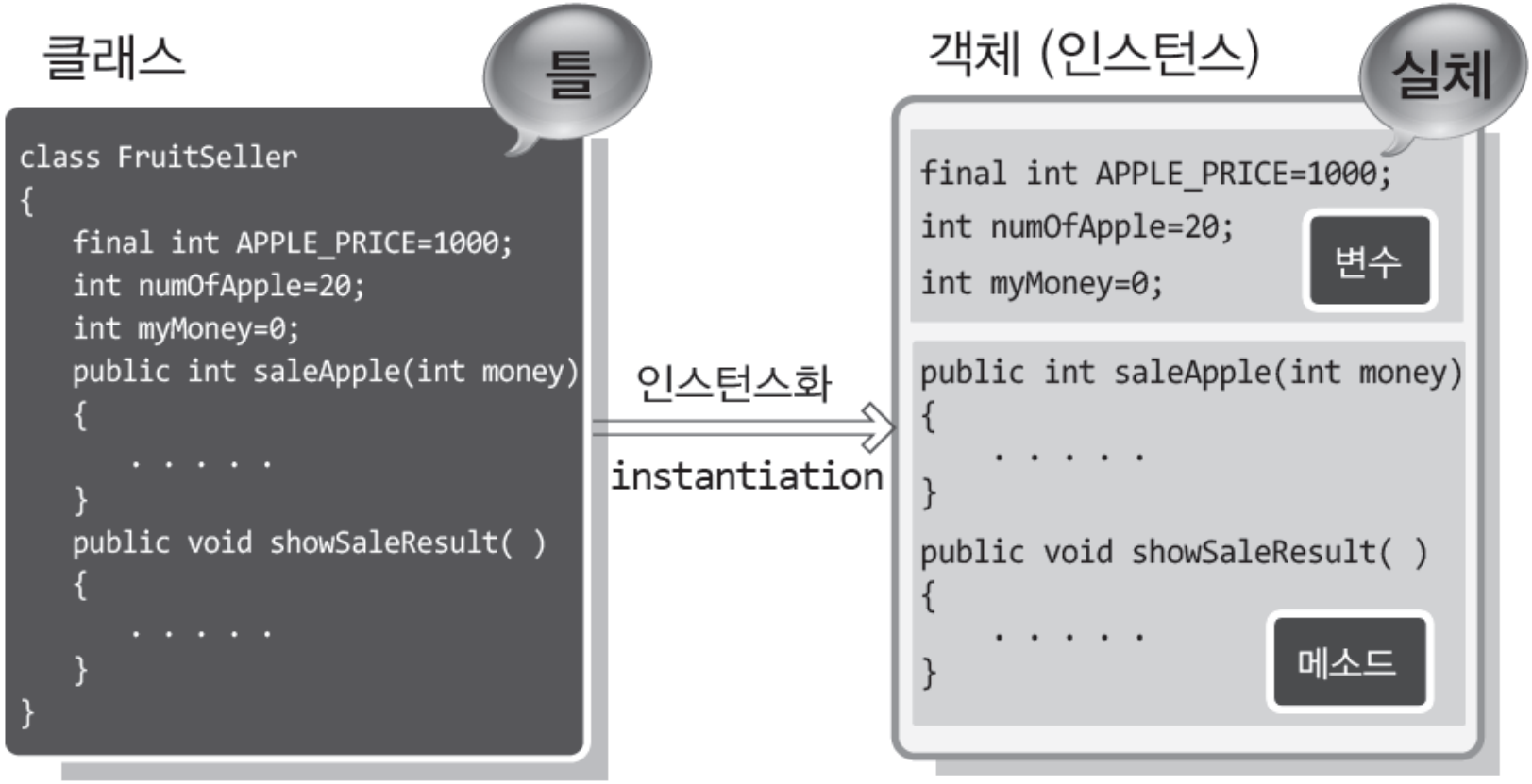
Object

- Object 생성 방법

```
Car myCar = new Car( );
```



Object



참조 변수의 선언 인스턴스의 생성

```
FruitSeller seller = new FruitSeller( );
FruitBuyer buyer = new FruitBuyer( );
```

Object

- Object 생성 방법
  - JAVA에서는 Object를 new 연산자로 생성

```
Class 타입 변수명 = new 생성자();
```

  - new 연산자를 통해 Object를 저장할 Memory를 할당
  - 곧이어 Constructor를 호출하여 Object를 초기화함
  - 생성자가 종료되면 new 연산자는 생성된 Object에 접근할 수 있도록 Reference(Object에 대한 주소, 또는 식별자(Identifier))를 반환
  - Constructor
    - Object 생성 시 Object를 초기화하고 Heap에 저장

Object

- Object의 선언(Reference 변수 선언)
  - Object의 선언은 null 값을 가진 변수만을 의미

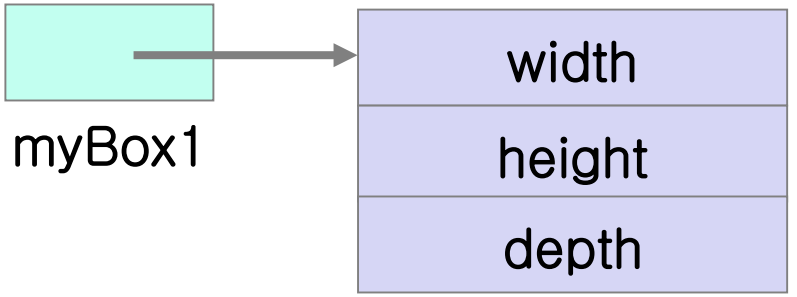
Box myBox1;

Box myBox2;

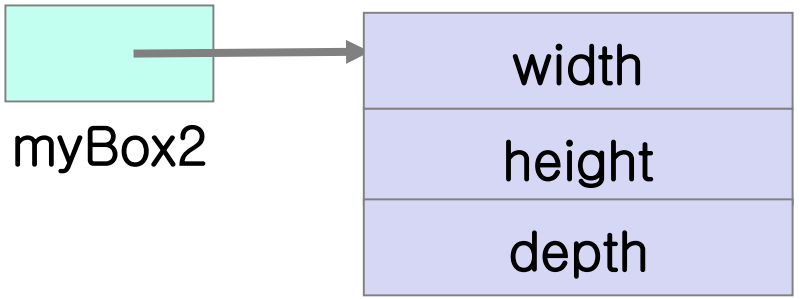
Object

- Object의 생성(Reference 변수와 Object의 연결)
  - Object에 대한 Memory가 할당되고, 변수(Object Reference 변수)는 Object에 대한 Reference(Address)를 가짐

myBox1 = new Box();

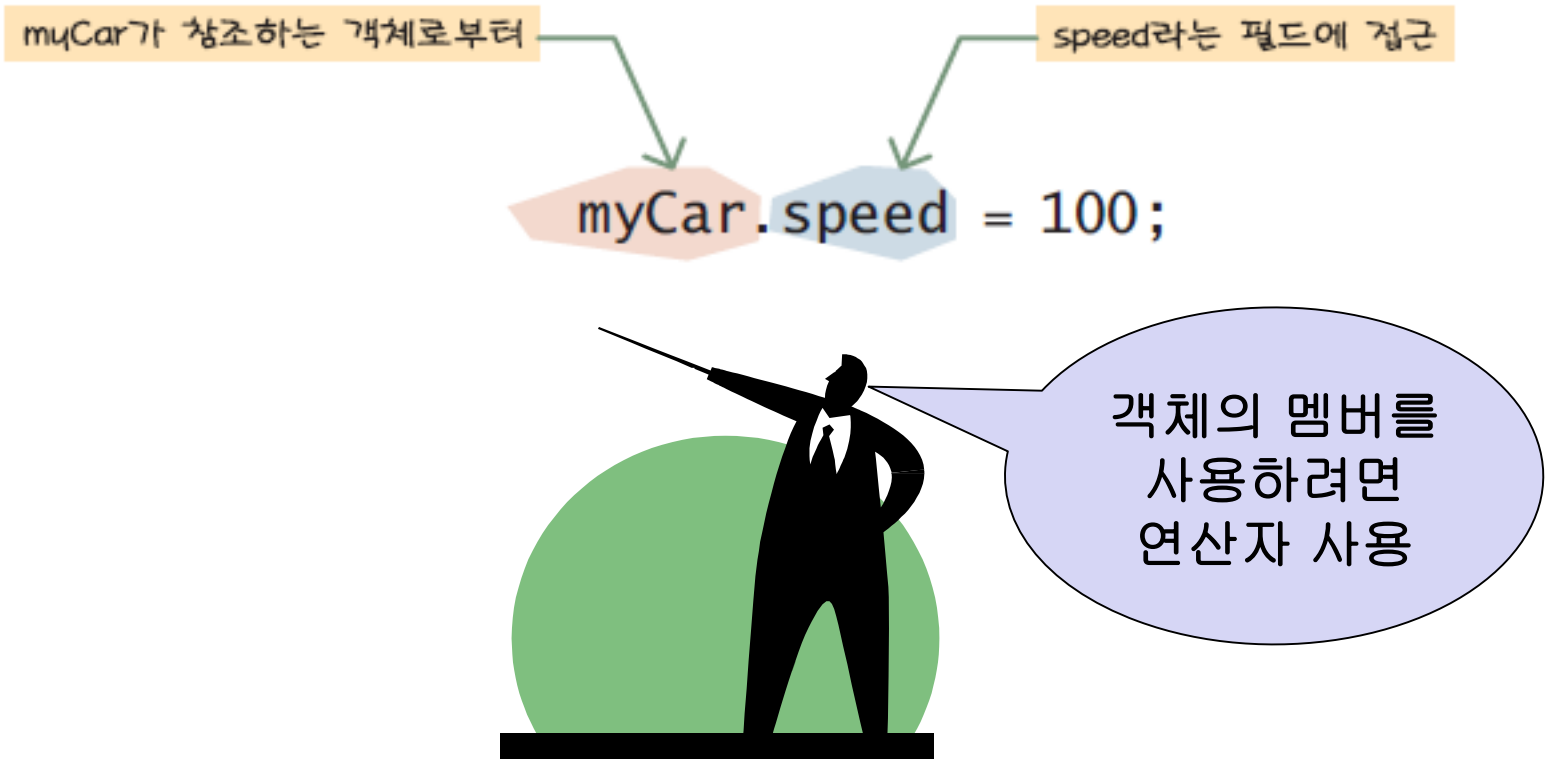


myBox2 = new Box();



Object

- Object의 Field와 Method 접근
  - 점(.) 연산자 사용



Object

- Object 내에 존재하는 변수의 접근

```
Fruitseller seller = new FruitSeller();
seller.numOfApple = 20;
```

- Object 내에 존재하는 Method의 접근(호출)

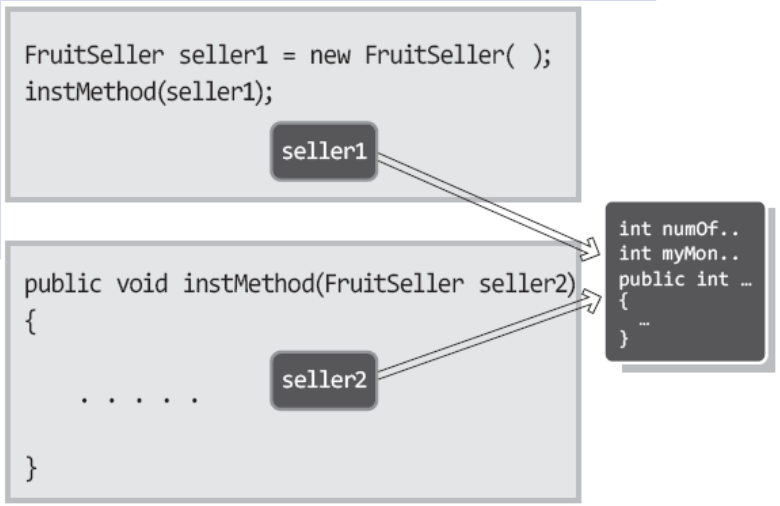
```
Fruitseller seller =new FruitSeller();
seller.saleApple(10);
```

Object

- Reference 변수와 Method의 관계

```
public void myMethod( ) {
        FruitSeller seller1 = new FruitSeller( );
        instMethod(seller1);
         ……
}
```

```
public void instMethod(FruitSeller seller2) {
         ……
         ……
         ……
}
```



Method

- Method는 Class내에서 Object가 할 수 있는 행동들을 정의한 것
- Class의 핵심이라 할 수 있음
- JAVA의 Class는 모두 main() Method를 가지고 있음
  - 이는 JAVA Interpreter가 자동 실행할 Method 이름이 main()으로 정해져 있었기 때문임
- 일반적으로 Class 내에서 main() Method를 반드시 정의할 필요는 없음

Method는 되도록 한 가지 기능만 담당하도록 작성
(한 Method에 너무 많은 기능 X)

Method

- Method 종류
  - Object Method
    - 모든 Member Field를 접근 가능
  - Class Method
    - Member Field중 Class Variable만을 접근 가능
    - 앞에 static이라는 Keyword가 붙는 것을 제외하고 Object Method를 정의하는 방법과 동일

Method

- Method 기본 구조

```
[접근 제어자] [반환 타입] 메소드 이름(매개 변수) {

    // 실행할 Code (Method가 수행할 작업)

    return 반환 값; // 반환 타입이 void면 생략
}
```

- 접근 제어자

접근 제어자	의미
public	모든 Class에서 호출 가능 (가장 많이 사용)
protected	같은 Package와 상속받은 Class에서만 호출 가능
default (생략 시)	같은 Package에서만 호출 가능
private	같은 Class 내에서만 호출 가능

Method

- 반환 Data Type

반환 타입	설명
int	정수 반환
double	실수 반환
String	문자열 반환
void	반환 값 없음 (결과를 돌려주지 않음)

- Method 이름 작성 규칙
  - 소문자로 시작 (관습)
  - 동사형으로 작성하는 것이 좋음
  - 어떤 기능인지 쉽게 알 수 있게 의미를 잘 담아야 함
  - camelCase 사용 (단어가 이어질 때 첫 글자를 대문자로)
    - printMessage, calculateSum, getUserName

Method

- 매개 변수(Parameter)
  - Method가 호출될 때 받는 값
  - 0개일 수도 있고, 여러 개일 수도 있음 (최대 7개 정도)
  - 매개 변수는 필요한 만큼만 작성
    (불필요한 매개 변수는 지양)
  - 매개 변수 Data Type과 이름(Identifier)을 쌍으로 작성하고, 콤마(,)로 구분함
  - 만일 Method에 매개 변수가 없을 경우에 공란이 됨

- return 문
  - Method가 결과 값을 돌려줄 때 사용
  - 반환 Data Type이 void가 아니면 반드시 return 필요
    - void시 return; 문도 가능은 함

Method

- Method 접근
  - Method에 접근하는 방법은 Class Method와 Object Method가 다른 방법으로 접근 할 수 있음
  - Class Method는 Class 이름으로 Object Method는 Object 이름으로 접근 함

Class Method 접근 형식

클래스 이름.클래스 메소드 이름(매개 변수)

Object Method 접근 형식

객체 이름.객체 메소드 이름(매개 변수)

Method 연습 1

- 매개 변수 없이 "오늘도 좋은 하루!"를 출력하는 Method를 만들어 보세요

```
public void happyDay( ) {


      return ;
}
```

Method 연습 2

- 이름(name)과 나이(age)를 매개 변수로 받아, "안녕하세요 ! 저는 OOO이고, 나이는 OO살입니다."를 출력하는 Method show()를 만들어 보세요

```
public void show( ) {


      return ;
}
```

Method 연습 2[심화]

- Method 호출 방법

```
public static void main(String[] args) {
    private Scanner keyboard = new Scanner(System.in);
    private String name;
    private int age;

    System.out.print("이름을 입력해주세요: ");
    name = keyboard.nextLine();
    System.out.printf("%s님의 나이를 입력해주세요: ", name);
    age = keyboard.nextInt();

    Introduce kim = new Introduce();
    kim.show(name, age);
}
```

Method 연습 2[심화]

```
public class Introduce {
    private String name;
    private int age;

    public void show(String name, int age) {
        System.out.printf("안녕하세요! 저는 %s이고,
                                                나이는 %d살입니다.₩n");
    }
}
```

Method 연습 3

- 두 숫자를 곱한 결과를 반환하는 Method를 작성해 보세요

```
public int sum(int num1, int num2) {


      return (   ) ;
}
```

Method 연습 3[심화]

- Method 사용
  - Main Class에서 호출하여야 사용됨

```
public class Main {
    public int sum(int num1, int num2){
        return num1 + num2;
    }

    public static void main(String[] args) {
        System.out.println(sum(1, 2));   // error!
    }
}
```

Cannot make a static reference to the non-static method add(int, int) from the type Main

Method 연습 3[심화]

- Method 사용

```
public class Main {
    public static int sum(int num1, int num2){
        return num1 + num2;
    }

    public static void main(String[] args) {
        System.out.println(Main.sum(1, 2));      // OK!
    }
}
```

Method 연습 3[심화]

- static method(정적 메소드)
  - static이라는 Keyword를 Method 앞에 붙으면 이 Method는 정적 메소드(static method)가 됨
  - Object를 생성하지 않고 사용할 수 있는 Method
  - 예) Math 클래스에 들어 있는 각종 수학 메소드들

```
double value = Math.sqrt(9.0);
```

두수 더하기[심화]

- Method 사용

```
public class Main {
    public static int sum(int num1, int num2){
        return num1 + num2;
    }

    public static void main(String[] args) {
        System.out.println(Main.sum(1, 2));      // OK!
        System.out.println(sum(1, 2));           // OK!
                                    // 클래스 내부에서는
                                    // 클래스 이름 생략 가능!
    }
}
```

Method 연습 3[심화]

- Method 사용
  - Object의 Method 호출을 위해서는 .(dot) 연산자 사용

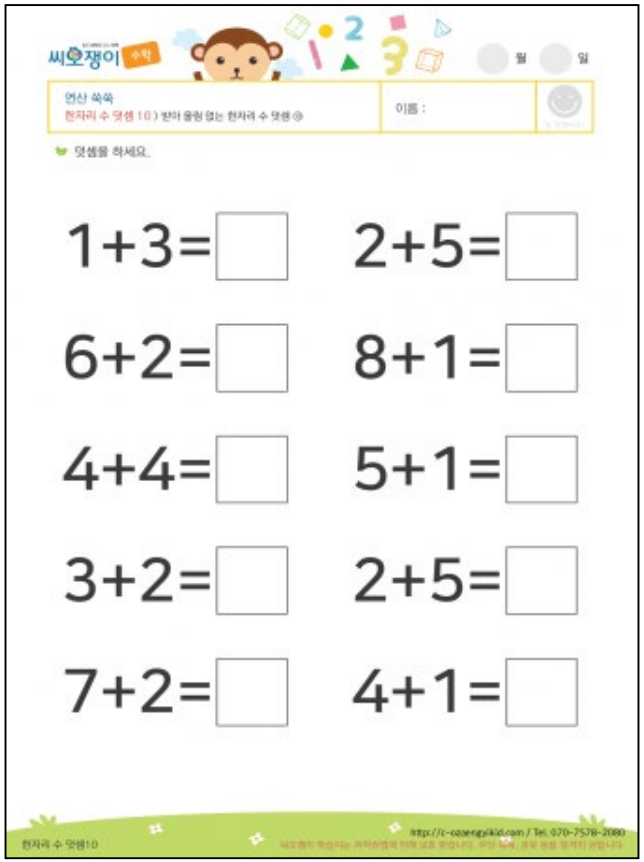
```
public class Main {
    public int sum(int num1, int num2) {
        return num1 + num2;
    }

    public static void main(String[] args) {
        Main adder = new Main();          // 객체 생성

        System.out.println(adder.sum(1, 2));    // OK!
    }
}
```

정수 더하기

- 정수 2개를 전달받아(Parameter) 두 수를 더한 결과를 return해주는 sum() 메소드를 포함한 클래스 Adder를 사용하여라.



정수 더하기

- Adder 클래스를 만들기
  (단, 클래스 내에 변수 2개를 선언하여 내부에 저장 후 값을 출력하는 방식으로 하자)

```
public class Add {
        private int num1;
        private int num2;

}
```

정수 더하기

- Adder 클래스를 만들기
  - 생성자 만들기
    - Class 이름과 “동일한” 이름을 가진 특수한 Method
    - 반환형이 없는 Method

```
public class Adder {
      private int num1
      private int num2;

      public Adder(int num1, int num2) {
           this.num1 = num1;
           this.num2 = num2;
     }
}
```

정수 더하기

- Adder 클래스를 만들기
  - Getter()/Setter() 만들기

```
public class Adder {
      private int num1
      private int num2;

      public Adder(int num1, int num2) {
           this.num1 = num1;
           this.num2 = num2;
     }

      public int getNum1() {
           return num1;
      }
          ..................

}
```

정수 더하기

- Adder 클래스를 만들기
  - Custom Method

```
public class Adder {
      private int num1;
      private int num2;
      ..........................

      public void input(int num1, int num2) {
           this.num1 = num1;
           this.num2 = num2;
      }

       public int sum(int num1, int num2) {
             return num1 + num2;
      }
}
```

정수 더하기

```
public class Adder {
    private int num1;
    private int num2;

    public Adder(int num1, int num2) {
        this.num1 = num1;
        this.num2 = num2;
    }

    public int sum() {
        return num1 + num2;
    }

    @Override
    public String toString() {
        String result;
        result = "num1 = " + num1 + ", num2 = " + num2;
        result += String.format("₩n %d + %d = %d", num1, num2, sum());
        return result;
    }
}
```

정수 더하기(I)

- Main 클래스

```
public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        System.out.print("정수 2개를 입력 : ");
        int num1 = keyboard.nextInt();
        int num2 = keyboard.nextInt();

        //   System.out.printf("%d + %d = %d₩n", num1, num2, num1 + num2);

        Adder adder = new Adder(num1, num2);
        System.out.println(adder);
}
```

정수 더하기(II)

- Object 생성 사용하기

```
public static void main(String[] args) {
    Scanner keyboard = new Scanner(System.in);

    System.out.print("정수 2개를 입력 : ");
    int num1 = keyboard.nextInt();
    int num2 = keyboard.nextInt();

     System.out.printf(" %d + %d = %d₩n" + num1, num2,
                                                num1 + num2);

     Adder add = new Adder(num1, num2);
     add.input(num1, num2);
     System.out.printf(" %d + %d = %d₩n" + add.getNum1(),
                                add.getNum2(), add.sum(num1, num2));
}
```

자연수의 합

- N에서 M까지의 자연수의 합을 구하여 보자
  - 단, n,과 m은 0보다 큼

  - 구조화 Programming
  - 객체 지향 Programming

자연수 합(Structured Programming)

```
public class Main {
    public static void main(String[] args) throws IOException {
        Scanner keyboard = new Scanner(System.in);
        int start, last, result = 0;
        while(true) {
            System.out.print("어디서부터 더할까요? ");
            start = keyboard.nextInt();
            if (start > 0)
                break;
            else {
                System.err.print("입력 오류");
                System.in.read();
            }
        }
        do {
            System.out.print("어디까지 더할까요? ");
            last = keyboard.nextInt();
        } while (last <= 0);
```

자연수 합(Structured Programming)

```
        for (int i = start; i <= last; i++)
            result += i;
        System.out.printf("%d + … + %d = %d₩n", start, last, result);
    }
}
```

자연수 합(Structured Programming)

- 같은 Class 안에 있는 static 메소드를 호출할 때

```
public class Main {
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        int start, last, result = 0;
        do {
            System.out.print("어디서부터 더할까요? ");
            start = keyboard.nextInt();
        } while (start <= 0);
        do {
            System.out.print("어디까지 더할까요? ");
            last = keyboard.nextInt();
        } while (last <= 0);
        result = add(start, last);

        System.out.printf("%d + … + %d = %d₩n", start, last, result);
    }
```

자연수 합(Structured Programming)

- 같은 클래스 안에 있는 static 메소드를 호출할 때

```
    private static int add(int start, int last) {
        int result = 0;
        for (int i = start; i <= last; i++)
            result += i;
        return result;
    }
}
```

자연수 합(Structured Programming)

- 다른 클래스에 있는 static 메소드를 호출할 때

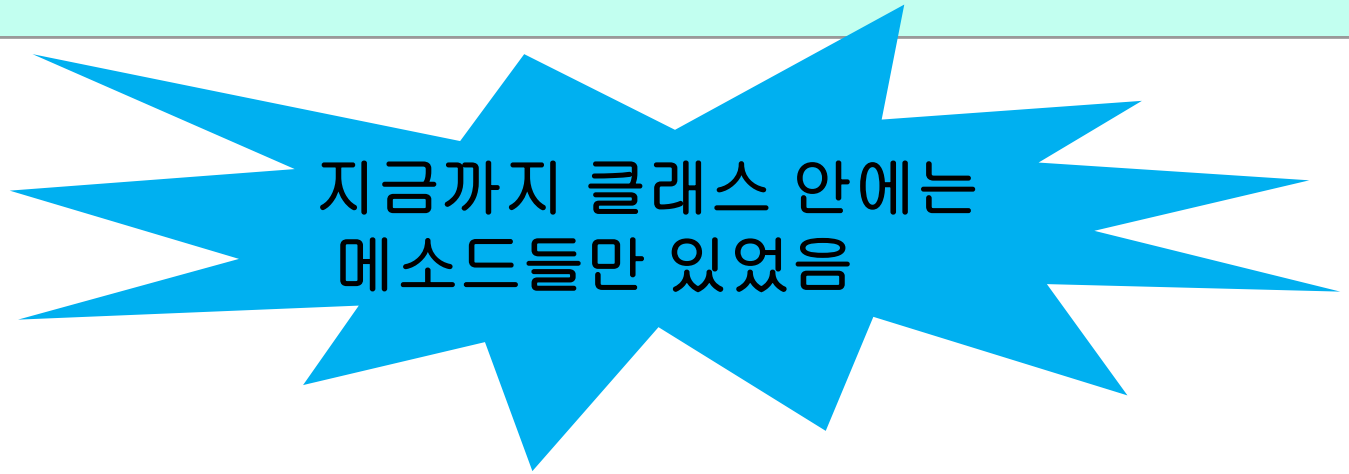
```
public class Main {
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        int start, last, result = 0;
        do {
            System.out.print("어디서부터 더할까요? ");
            start = keyboard.nextInt();
        } while (start <= 0);
        do {
            System.out.print("어디까지 더할까요? ");
            last = keyboard.nextInt();
        } while (last <= 0);
        result = Adder.add(start, last);

        System.out.printf("%d + … + %d = %d₩n", start, last, result);
    }
}
```

자연수 합(Structured Programming)

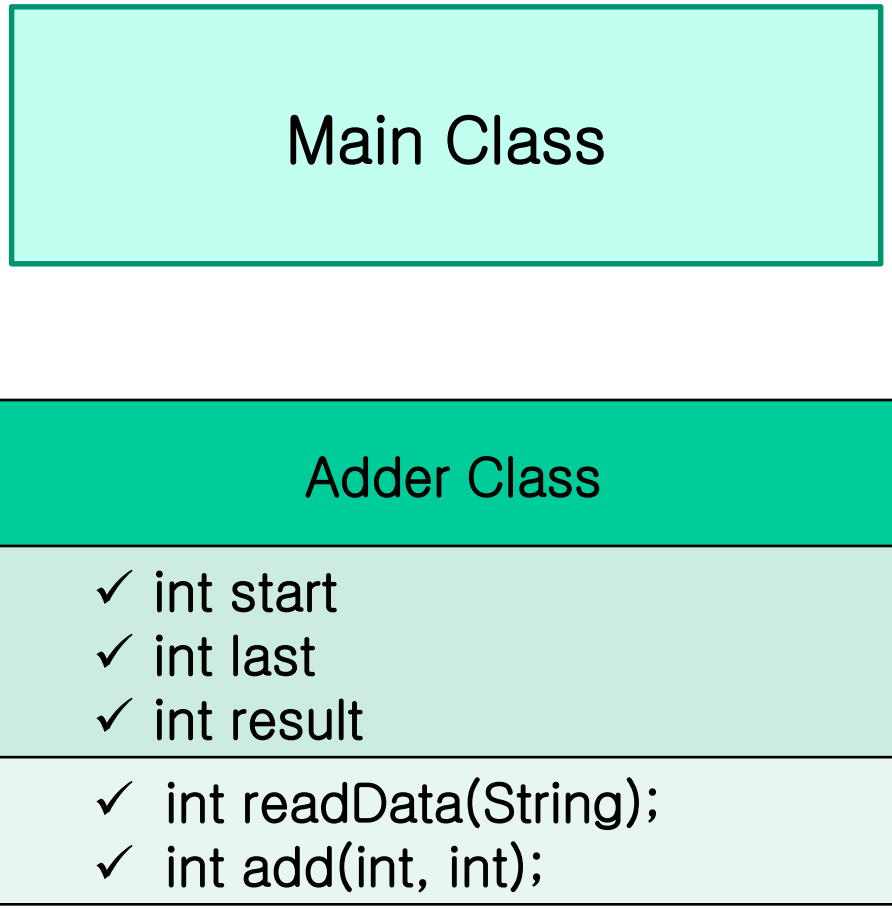
- 다른 클래스에 있는 static 메소드를 호출할 때

```
public class Adder {
    public static int add(int start, int last) {
        int result = 0;
        for (int i = start; i <= last; i++)
        result += i;
        return result;
    }
}
```



자연수 합(Object Oriented)

- 추상화



자연수 합(Object Oriented)

- Adder Class

```
public class Adder {
    private int start;        // 필드
    private int last;
    private int result;

    public Adder() {      // 생성자
        result = 0;
    }

    public Adder(int start, int last) {    // 생성자
        this.start = start;
        this.last = last;
        result = 0;
    }
```

자연수 합(Object Oriented)

- Adder Class

```
public void setStart(int start) {      //setter
    this.start = start;
}

public void setLast(int last) {
    this.last = last;
}

public int getStart() {                //getter
    return start;
}

public int getLast() {
    return last;
}
```

자연수 합(Object Oriented)

- Adder Class

```
public int readData(String prompt) throws IOException {
    Scanner keyboard = new Scanner(System.in);
    int data = 0;
    while(true) {
        System.out.print(prompt);
        data = keyboard.nextInt();
        if (data > 0)
            break;
        else {
            System.err.print("입력 오류");
            System.in.read();
        }
    }
    return data;
}
```

자연수 합(Object Oriented)

- Adder Class

```
    public void add() {
        for (int i = start; i <= last; i++)
            result += i;
    }

    @Override
    public String toString() {
        return start + " + ..... + " + last + " = " + result;
    }
}
```

자연수 합(Object Oriented)

- Main Class

```
public static void main(String[] args) throws IOException {
        Adder test = new Adder();

        test.setStart(test.readData("어디서부터 더할까요? "));
        test.setLast(test.readData("어디까지 더할까요? "));

        test.add();

        System.out.print(test);
}
```

최대, 최소값(I)

- 3개의 정수를 전달받아, 최대값과 최소값을 구하는 메소드를 만들어보자.

```
public static void main(String[] args) {
    Scanner keyboard = new Scanner(System.in);

    System.out.print("정수 3개를 입력 : ");
    int num1 = keyboard.nextInt();
    int num2 = keyboard.nextInt();
    int num3 = keyboard.nextInt();

    MaxMin.showInfo(num1, num2, num3);
}
```

최대, 최소값(I)

```
public class MaxMin {
    static int max(int num1, int num2, int num3) {
        int max = num1;

        // 방법1
        if (max < num2) max = num2;
        if (max < num3) max = num3;


        // 방법2
        max = max < num2 ? (num2 < num3 ? num3 : num2) :
                (max < num3 ? num3 : max);

        return max;
    }
```

최대, 최소값(I)

```
    static int min(int num1, int num2, int num3) {
        int min = num1;
        if (min > num2) min = num2;
        if (min > num3) min = num3;

        min = min > num2 ? (num2 > num3 ? num3 : num2) :
                                                (min > num3 ? num3 : min);

        return min;
    }

    static void showInfo(int num1, int num2, int num3) {
        System.out.println("최대값: " + max(num1, num2, num3));
        System.out.println("최소값: " + min(num1, num2, num3));
    }

}
```

최대, 최소값(II)

```
public static void main(String[] args) {
    Scanner keyboard = new Scanner(System.in);

    System.out.print("정수 3개를 입력 : ");
    int num1 = keyboard.nextInt();
    int num2 = keyboard.nextInt();
    int num3 = keyboard.nextInt();

    MaxMin maxMin = new MaxMin(num1, num2, num3);
    System.out.println(maxMin);
}
```

최대, 최소값(II)

```
public class MaxMin {
    private int num1;
    private int num2;
    private int num3;

    public MaxMin(int num1, int num2, int num3) {
        this.num1 = num1;
        this.num2 = num2;
        this.num3 = num3;
    }

    public int max() {
        int max = num1;
        if (max < num2) max = num2;
        if (max < num3) max = num3;
        // 방법2
        // max = max < num2 ? (num2 < num3 ? num3 : num2) :
                                                    (max < num3 ? num3 : max);
        return max;
    }
```

최대, 최소값(II)

```
    public int min() {
        int min = num1;
        if (min > num2) min = num2;
        if (min > num3) min = num3;

        min = min > num2 ? (num2 > num3 ? num3 : num2) :
            (min > num3 ? num3 : min);
        return min;
    }

    @Override
    public String toString() {
        String result = "";
        result += "num1 = " + num1 +  ", num2 = " + num2 +
                ", num3 = " + num3;
        result += "₩n최대값: " + max();
        result += "₩n최소값: " + min();
        return result;
    }
}
```