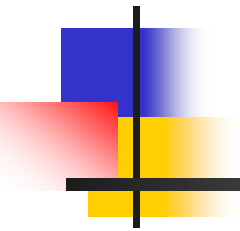


Method

경복대학교
소프트웨어융합과
배희호 교수

Method

- 객체 지향 설계에서 Method(메소드) 설계는 Code의 유지보수성, 재 사용성, 확장성을 높이기 위해 중요한 요소
- 잘 설계된 Method는 가독성이 좋고, 예측 가능하며, Object 간의 결합도를 낮추는 역할을 함
- Class 내부에서 정의된 명령어 집합으로 Class로부터 생성된 Object를 사용하는 방법으로 Object에 명령을 내리는 Message
- Class 내부의 Method에서 다른 Method 호출 가능
- Class 외부에서는 특정 Method만 호출 가능
- 예) String Object의 경우
  - length()
  - reverse()

Method

- Message와 Receiver
  - Method의 Message와 Receiver란 OOP에서 Object들은 서로 Message를 주고받으며 상호 작용 함
  - 이 과정에서 Method를 호출하는 Object를 "수신자(Receiver)", 호출되는 Method를 "메시지(Message)"라고 함
  - Method가 소속된 Object는 Receiver
  - Message는 Method와 Parameter 값
  - 예) a.getSize(x)
    - a: Receiver
    - getSize(x) : Message

Method

- Class 내에 Instance Method와 Static Method가 섞여 있음

```
class Adder {
    private int sum;                          // Instance 변수
    public void add(int value) {              // Instance  Method
        sum =+ value;
    }
    public int getSum() {                     // Instance Method
            return sum;
    }
    public static int add(int m, int n) {    // Class Method
            return m + n;
    }
}
```

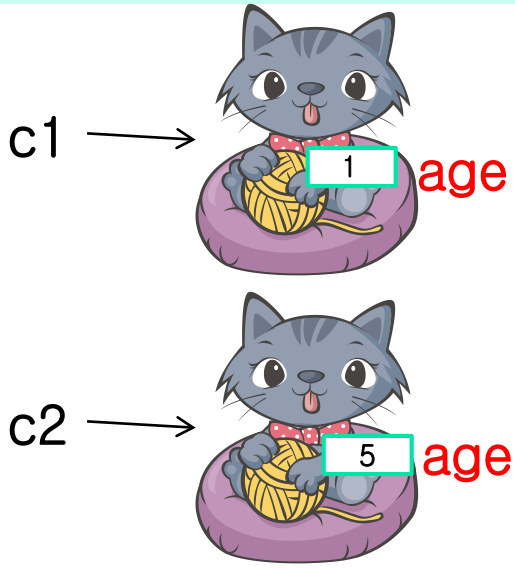
```
class AdderDemo {
     public static void main(String[] args) {
         System.out.println( Adder.add(3, 5));       // Class Method 호출
         Adder a = new Adder();
         a.add(3);
         System.out.println(a.getSum());             // Instance Method 호출
     }
}
```

Method

- Method를 통해 각 Object의 Instance 변수에 접근하는 방법

```
public class Cat {
        int age;                    // Instance 변수
        public void eat() { System.out.println("냠냠!");}
        public void yaong() {System.out.println("야옹");}
        public void setAge(int n) { age = n; }
        int getAge() { return age; }
}
```

```
Cat c1 = new Cat();
Cat c2 = new Cat();

c1.setAge(1);
System.out.println(c1.getAge());

c2.setAge(5);
System.out.println(c2.getAge());
```



Method

static 키워드가 붙지 않은 메소드

인스턴스 메소드

```
class Cat {
        int age;                    // 변수(상태변수)
        public void eat() { System.out.println("냠냠!");}
        public void yaong() {System.out.println("야옹");}
        public void setAge(int n) { age = n; }
        int getAge() { return age; }
}
```

static 아닌 Method
(인스턴스 메소드)는
Object를 지정하여 호출해야 함

static Method를 호출하는
방식으로는 호출할 수 없음
Cat.setAge(1) X
Cat.getAge() X

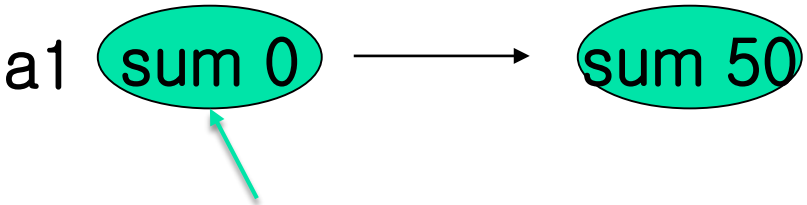
```
Cat c1 = new Cat();
Cat c2 = new Cat();

c1.setAge(1);
System.out.println(c1.getAge());

c2.setAge(5);
System.out.println(c2.getAge());
```

Method 예제

```
class Adder {
    private int sum;                        // Instance 변수
    public void add(int value) {        // Instance Method
        sum += value;
    }
    public int getSum() {                // Instance Method
            return sum;
    }
}
```

```
class Main {
     public static void main(String[] args) {
        Adder a1 = new Adder();
        Adder a2 = new Adder();
        a1.add(50));
        System.out.println(a1.getSum());
        a2.add(5));
        a2.add(3));
        System.out.println(a2.getSum());
     }
}
```

Method 예제



add(int value), getSum()은 객체의 상태를 조작하거나 상태 값을 읽는 방법이므로 메소드라고 부름


```
50
8
```

Method

- 좋은 Method 설계 원칙
  - 단일 책임 원칙(SRP) 적용
    - 하나의 Method는 하나의 역할만 수행해야 함
    - 여러 개의 역할을 수행하면 유지 보수성이 떨어지고, Test가 어려워 짐
  - Method 이름을 명확하게 작성
    - Method 이름만 보고 기능을 예측할 수 있어야 함
    - 동사 + 명사 형식 사용
    - 예) getUserById(), calculateTotalPrice())
  - 입력 Parameter 최소화
    - Method의 매개 변수는 최대한 적게 사용해야 함
    - 권장 : 3개 이하
    - 많은 매개 변수가 필요한 경우, Object로 묶어서 전달하는 것이 좋음

Method

- 좋은 Method 설계 원칙
  - Method의 반환 값을 신중하게 결정
    - 가능하면 null을 반환하지 않기
    - Optional<T> 또는 예외 처리를 활용
  - Method의 길이를 짧게 유지
    - 한 Method는 5~20줄 내외로 유지하는 것이 이상적
    - 너무 길어지면 의미 있는 단위로 분리
  - Overloading과 Overriding 적절히 활용
    - Overloading : 같은 Method 이름을 유지하면서 매개변수를 다르게 정의
    - Overriding : 부모 Class의 Method를 자식 Class에서 재정의
  - Error 처리 및 예외(Exception) 관리
    - 예외를 적절하게 처리하고, 명확한 Message를 제공해야 함

Method

- 주석 기입 및 API 문서 제작
  - Method를 구현하기(Method Body에 Code를 적어 넣기) 전에 Method에 대한 주석을 작성
  - Method 입력, 출력, 기능에 대한 개념을 명확하게 함

Method Detail

withdraw

```
public void withdraw(double amount)
```

계좌에서 출금한다.

Parameters:
amount - 출금액수

---

getBalance

```
public double getBalance()
```

잔고를 보여준다.

Returns:
잔고

Method

- 주석 기입 및 API 문서 제작

```
/**
   계좌에서 출금한다.
   @param amount 출금액수
*/

public void withdraw(double amount){
        // 구현은 나중에 함
}
```

```
/**
   잔고를 보여준다.
   @return 잔고
*/

public double getBalance() {
   // 구현은 나중에 함
}
```

Method

- 주석 기입 및 API 문서 제작
  - 정해진 규칙에 맞추어 주석을 달면 javadoc 도구를 사용해 API 문서를 만들어 낼 수 있음

```
$ javadoc BankAccount.java
```

```
/**
   BankAccount는 은행계좌로서 입금, 출금이 가능하며
   현재 잔고를 확인할 수 있다.
*/

public class BankAccount {
   . . .
}
```

Method

- 주석 기입 및 API 문서 제작

public class **BankAccount**
extends java.lang.Object

BankAccount는 은행계좌로서 입금, 출금이 가능하며 현재 잔고를 확인할 수 있다.

Constructor Summary

BankAccount()

Method Summary

double	getBalance() 잔고를 보여준다.
void	withdraw(double amount) 계좌에서 출금한다.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Method

- Method의 접근 제어자
  - 클래스 내에서 Method의 접근 범위를 설정하는 중요한 요소
  - 이를 통해 다른 클래스나 객체가 해당 Method를 어떻게 사용할 수 있는지 제어할 수 있음
  - 일반적으로 사용되는 접근 제어자
    - public
    - private
    - protected
    - default(아무 접근 제어자도 명시하지 않은 경우)

Method

- public
  - 해당 Method는 어디서든 접근할 수 있음
  - Method가 외부에서도 호출되어야 할 때 사용
  - 예) 외부 API나 Library에서 제공하는 Method

- private
  - 해당 Method는 같은 클래스 내에서만 접근할 수 있음
  - 클래스 내부에서만 사용되고, 외부에서는 호출될 필요가 없는 Method에 사용
  - 예) 내부 Logic을 처리하는 보조 Method

Method

- protected
  - 해당 Method는 같은 클래스와 상속받은 하위 클래스에서 접근할 수 있음
  - 또한 같은 Package내에 있으면 다른 클래스에서도 접근 가능
  - 상속받은 클래스가 부모 클래스의 Method를 사용할 수 있도록 할 때 사용 (상속 관계에서 유용)

- default (패키지-프라이빗)
  - 접근 제어자를 명시하지 않으면 같은 Package 내에서만 접근할 수 있음
  - 외부 Package에서는 접근할 수 없음
  - 특정 Package내에서만 사용되는 Method를 만들 때 사용(Package내에서 협력하는 클래스들끼리 사용하는 경우)

Method

- final Keyword와 함께 사용하는 접근 제어자
  - final Keyword를 메소드에 추가하면, 상속받은 클래스에서 Overriding할 수 없게 됨

Method

- Method 종류
  - Class Method
    - Class 변수들 만을 접근하여 원하는 연산을 하는 Method
  - Instance Method
    - Class 변수와 Instance 변수들을 접근할 수 있는 Method

Method

- Class Method
  - Class Method는 Class 자체에 속하는 Method로, Object를 생성하지 않고도 호출할 수 있는 Method
  - 즉, Class Level에서 동작하며, Instance 변수에 접근할 수 없음
  - Method를 static으로 선언하면 그 Method는 개별 Object에 작용하지 않는다는 의미
  - Class Method는 아래와 같은 형식으로 호출

```
ClassName. methodName(parameters)
```

  - Math 클래스 Method들은 대부분 static 메소드

Method

- Math class

Math.sqrt(x)	square root
Math.pow(x, y)	power x^y
Math.exp(x)	e^x
Math.log(x)	natural log
Math.sin(x), Math.cos(x), Math.tan(x)	sine, cosine, tangent (x in radian)
Math.round(x)	closest integer to x
Math.min(x, y), Math.max(x, y)	minimum, maximum

Method

- Math class

$$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

```
(-b + Math.sqrt(b * b - 4 * a * c)) / (2 * a)
```

Method

- Class Method 제약 조건
  - Class Method는 오직 static Member만 접근 가능
    - Object가 생성되지 않은 상황에서도 사용이 가능하므로 Object에 속한 Instance Method, Instance 변수 등은 사용 불가
    - Instance Method는 static Memeber들을 모두 사용 가능
  - Class Method에서는 this Keyword를 사용할 수 없음
    - Object가 생성되지 않은 상황에서도 호출이 가능하기 때문에 현재 실행중인 Object를 가리키는 this Refeence를 사용할 수 없음

Method

- Class Method 제약 조건

```
class StaticMethod{
      int n;
      void f1(int x) {this.n = x;}          // 정상
      void f2(int x) {this.m = x;}          // 정상
      static int m;
      static void s1(int x) {n = x;}        // 오류
      static void s2(int x) {f1(3);}        // 오류
      static void s3(int x) {m = x;}        // 정상
      static void s4(int x) {s3(3);}        // 정상
      static void s5(int x) {this.n = x;}   // 오류
}
```

Method

- 변수의 종류
  - Fields (상태 변수)
  - Local Variables(지역 변수) Parameter(매개 변수)

메소드에 속하며 메소드가 실행되는 동안에만 존재함

```
public class BankAccount {
      private double balance;
      public void deposit(double amount)   {
          double newBalance = balance + amount;
          balance = newBalance;
    }
}
```

Method

- 매개 변수(Parameter)

메소드 선언

```
public class BankAccount {
    private double balance;
    public void deposit(double amount)   {
       double newBalance = balance + amount;
       balance = newBalance;
    }
}
```

파라미터

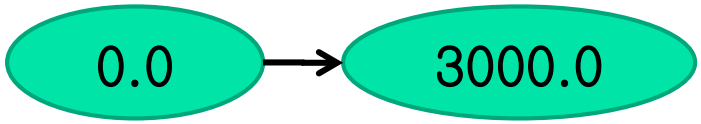
call-by-value
(값 복사)

BankTester 클래스의 main 메소드

메소드 활용

```
BankAccount harrysChecking = new BankAccount();
harrysChecking.deposit(3000.0);
```

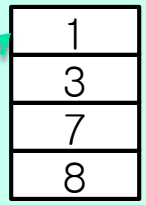
인자(argument)



Method

- 가변 Parameter 메소드
  - Method를 구현할 때 Parameter 수를 미리 정하지 않을 수도 있음
  - Keyword ...을 사용

```
class Adder {
    public int add(int... values) {
        int sum = 0;
        for (int i = 0; i < values.length; i++)
            sum += values[i];
        return sum;
    }
}
Adder adder = new Adder();
adder.add(1, 3, 7);
adder.add(1, 2);
adder.add(1, 3, 7, 8);
```

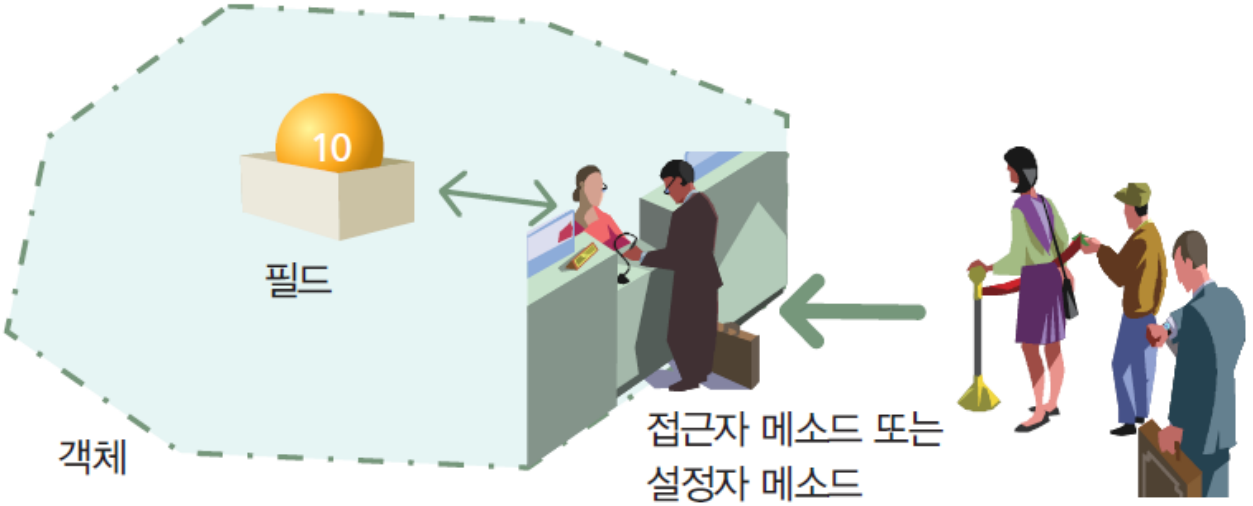


① 인자들이 배열로 만들어짐

② 배열을 가리키는 참조가 Parameter로 복사

Setter와 Getter

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



접근자와 설정자 Method만을 통하여 Field에 접근

Setter와 Getter

- 설정자(mutator)란?
  - 일반적으로 세터(setter)라고 부르기도 함
  - private 설정된 Member 변수(Field)에 대하여 외부 Class에서 설정이 불가하기 때문에 Field의 값을 설정하는 기능을 Method 형태로 해당 Class에 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 Class에서 접근이 불가하기 때문에 Field의 값을 반환하는 기능을 Method 형태로 해당 Class에 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 Class 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());
    }
}
```