JAVA WEEK 2

Min-su Kim

KOSMOS

dlxhshzh@korea.ac.kr

October 8, 2018

Overview

Quiz 1

```java
public static void main(String[] args)
{
    int      num = 1;
    if (num > 5 && (num++) == 1)
        System.out.println("plus one!");
    else
        System.out.println(num);
}
```

- 출력 결과?

Quiz 1 Solution

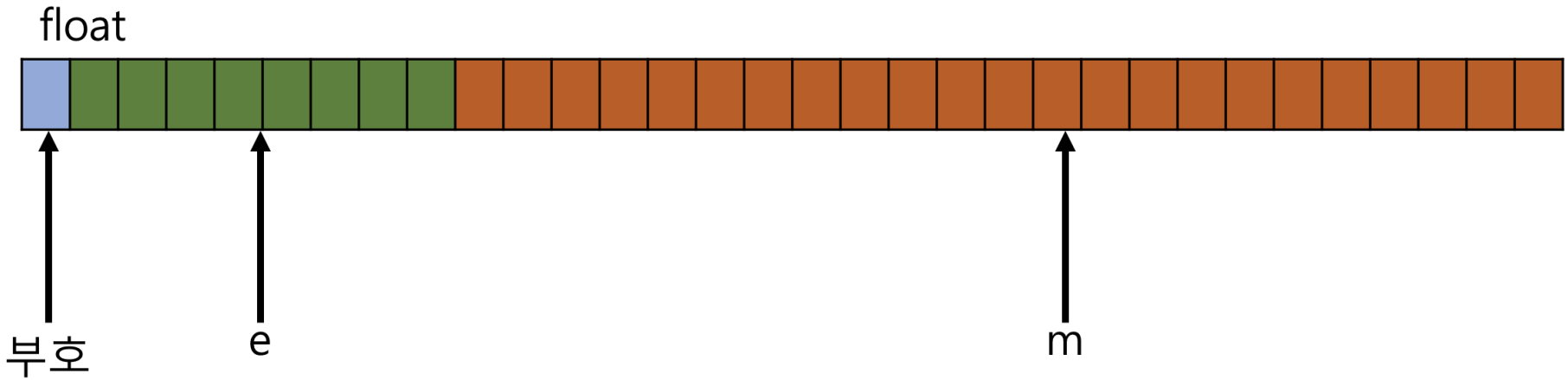
- 1
- "Short circuit rule" says...
- 더 이상 계산하지 않아도 되는 것은 계산하지 않음

Data Format

1. 정수의 저장
   - 값을 이진수로 저장
   - $53 = 0011\ 0101_{(2)}$
   - 음수의 경우, 원래 정수를 2의 보수로 변환해서 저장
   - $-53 = 1100\ 1011_{(2)}$
   - 2의 보수의 정의에 의해서
   - $-53 = \sim 53 + 1$

Data Format

1. 실수의 저장
   - "IEEE 754"에 의한 근사값 저장
   - float는 $\pm(1.m)x2^{e-127}$로 표현될 때,



Bitwise Operator

- 비트 단위로 연산하는 연산자

연산자	기능
&	비트단위로 AND 연산 a & b;
|	비트단위로 OR 연산 a | b;
^	비트단위로 XOR 연산 a ^ b;
~	모든 비트 반전 ~a;

Bitwise Operator

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

a	b	a | b
0	0	0
0	1	1
1	0	1
1	1	1

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

a	~a
0	1
1	0

Bitwise Operator Examples

$53 = 00110101_{(2)}$
$40 = 00101000_{(2)}$

- $53 \ \& \ 40 = 00100000_{(2)}$
- $53 \mid 40 = 00111101_{(2)}$
- $53 \ \hat{} \ 40 = 00011101_{(2)}$

Bitwise Operator

연산자	기능
<<	비트열을 왼쪽으로 이동, 빈 공간 0으로 채움 n << 2;
>>	비트열을 오른쪽으로 이동, 빈 공간 맨 앞 비트와 같은 값으로 채움 n >> 3;
>>>	비트열을 오른쪽으로 이동, 빈 공간 0으로 채움 n >>> 3;

Bitwise Operator Examples

$53 = 00110101_{(2)}$
$40 = 00101000_{(2)}$

- <<
  1. $53 << 1 = 01101010_{(2)} = 106$
  2. $40 << 1 = 01010000_{(2)} = 80$
- >>
  1. $53 >> 1 = 00011010_{(2)} = 26$
  2. $40 >> 1 = 00010100_{(2)} = 20$
- 결국 <<는 2를 곱하고
- >>는 2를 나누는 것과 같음
- (기존 *, / 연산자들 보다 <<, >> 연산자의 연산 속도가 더 빠름)

What is Method Overloading?

- 이름이 같은 메소드를 매개변수를 통해 구분하여 선언하는 것
- 컴퓨터의 입장에서, 메소드를 처음부터 구별할 수 있어야 선언 가능

Method Overloading Example

- "println()" method in Java
- System.out.println(String);
- System.out.println(int);
- System.out.println(boolean);
- ...
- 메소드에 들어가는 인자의 type에 따라 각각 다른 메소드가 호출되지만, 같은 기능
- 즉, 메소드를 사용할 때 편리하게 사용 가능

Method Overload Example

```
public static int power (int a, int n)
{
    int      result = 1;
    for (int i = 0; i < n; ++i)
        result *= 1;
    return result;
}
public static int power (int a)      //! 매개 인자의 개수로 구별
{
    return a * a;
}
```

- parameter의 개수로 구별 가능

Method Overload Example

```
public static int power (int a, int n)
{
    int      result = 1;
    for (int i = 0; i < n; ++i)
        result *= 1;
    return result;
}
public static int power (double a, int n)    //! 매개변수 type
{
    double  result = 1;
    for (int i = 0; i < n; ++i)
        result *= 1;
    return result;
}
```

- parameter의 type으로 구별 가능

Method Overload Example

```
public static int returnOne()
{
    return 1;
}
public static double returnOne()    //! 구별 불가!
{
    return 1.0;
}
```

- 구별 불가능 why?
- return type은, 컴퓨터가 함수의 선언부를 보자마자 알 수가 없음.
- So, error

Quiz 2

```
public static int power (int a, int n)
{
    int     result = 1;
    for (int i = 1; i <= n; ++i)
        result *= a;
    return result;
}
```

- 함수가 이럴 경우,
- System.out.println(power(2.2, 3));의 출력 결과

Quiz 2 Solution

- compile error!
- (double) type을 (int) type으로 형변환 하는 것
- 화살표 반대 방향이므로 error

Quiz 3

```
public static double power (double a, int n)
{
    double  result = 1;
    for (int i = 1; i <= n; ++i)
        result *= a;
    return result;
}
```

- 함수가 이럴 경우,
- System.out.println(power(2, 3));의 출력 결과

Quiz 3 Solution

- 8.0
- int에서 double로의 형변환이므로 가능

Quiz 4

```java
public static int power (int a, int n)
{
    int     result = 1;
    for (int i = 1; i <= n; ++i)
        result *= a;
    return result;
}
public static double power (double a, int n)
{
    double  result = 1.0;
    for (int i = 1; i <= n; ++i)
        result *= a;
    return result;
}
▶ Run | 🐞 Debug
public static void main(String[] args)
{
    System.out.println(power(2.2, 3));
}
```

- 함수가 이럴 경우,
- System.out.println(power(2.2, 3));의 출력 결과

Quiz 4 Solution

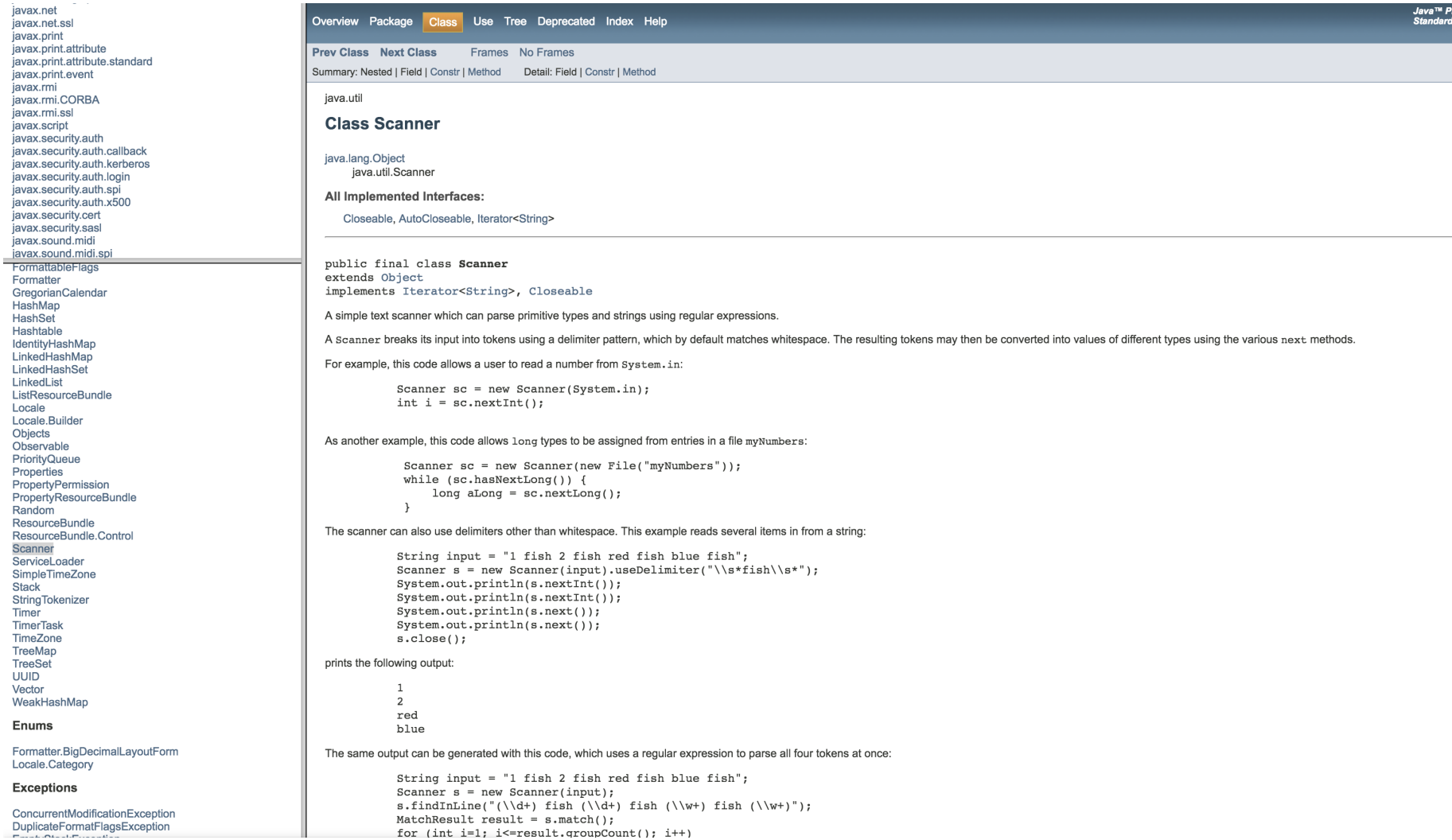
- 10.648
- 이건 진짜 method overloading

What is Java API ?

- 자주 사용하는 클래스와 메소드들의 라이브러리
- https://docs.oracle.com/javase/9/docs/api/index.html?overview-summary.html
- 위 주소에서 API들을 검색하고, 사용할 수 있음
- ctrl + f로 검색하고 싶은 클래스 이름을 찾기 가능

Java API Example

- Scanner Class



javax.net
javax.net.ssl
javax.print
javax.print.attribute
javax.print.attribute.standard
javax.print.event
javax.rmi
javax.rmi.CORBA
javax.rmi.ssl
javax.script
javax.security.auth
javax.security.auth.callback
javax.security.auth.kerberos
javax.security.auth.login
javax.security.auth.spi
javax.security.auth.x500
javax.security.cert
javax.security.sasl
javax.sound.midi
javax.sound.midi.spi
FormattableFlags
Formatter
GregorianCalendar
HashMap
HashSet
Hashtable
IdentityHashMap
LinkedHashMap
LinkedHashSet
LinkedList
ListResourceBundle
Locale
Locale.Builder
Objects
Observable
PriorityQueue
Properties
PropertyPermission
PropertyResourceBundle
Random
ResourceBundle
ResourceBundle.Control
Scanner
ServiceLoader
SimpleTimeZone
Stack
StringTokenizer
Timer
TimerTask
TimeZone
TreeMap
TreeSet
UUID
Vector
WeakHashMap

Enums

Formatter.BigDecimalLayoutForm
Locale.Category

Exceptions

ConcurrentModificationException
DuplicateFormatFlagsException

Overview Package Class Use Tree Deprecated Index Help

Prev Class Next Class Frames No Frames

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.util

Class Scanner

java.lang.Object
java.util.Scanner

All Implemented Interfaces:

Closeable, AutoCloseable, Iterator<String>

```
public final class Scanner
extends Object
implements Iterator<String>, Closeable
```

A simple text scanner which can parse primitive types and strings using regular expressions.

A `Scanner` breaks its input into tokens using a delimiter pattern, which by default matches whitespace. The resulting tokens may then be converted into values of different types using the various `next` methods.

For example, this code allows a user to read a number from `System.in`:

```
Scanner sc = new Scanner(System.in);
int i = sc.nextInt();
```

As another example, this code allows `long` types to be assigned from entries in a file `myNumbers`:

```
Scanner sc = new Scanner(new File("myNumbers"));
while (sc.hasNextLong()) {
    long aLong = sc.nextLong();
}
```

The scanner can also use delimiters other than whitespace. This example reads several items in from a string:

```
String input = "1 fish 2 fish red fish blue fish";
Scanner s = new Scanner(input).useDelimiter("\\s*fish\\s*");
System.out.println(s.nextInt());
System.out.println(s.nextInt());
System.out.println(s.next());
System.out.println(s.next());
s.close();
```

prints the following output:

```
1
2
red
blue
```

The same output can be generated with this code, which uses a regular expression to parse all four tokens at once:

```
String input = "1 fish 2 fish red fish blue fish";
Scanner s = new Scanner(input);
s.findInLine("(\\d+) fish (\\d+) fish (\\w+) fish (\\w+)");
MatchResult result = s.match();
for (int i=1; i<=result.groupCount(); i++)
```

Procedural and Object Oriented

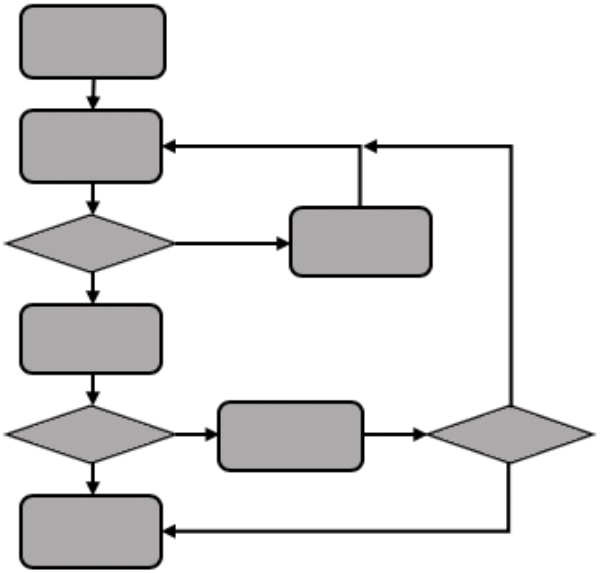
Procedural Programming

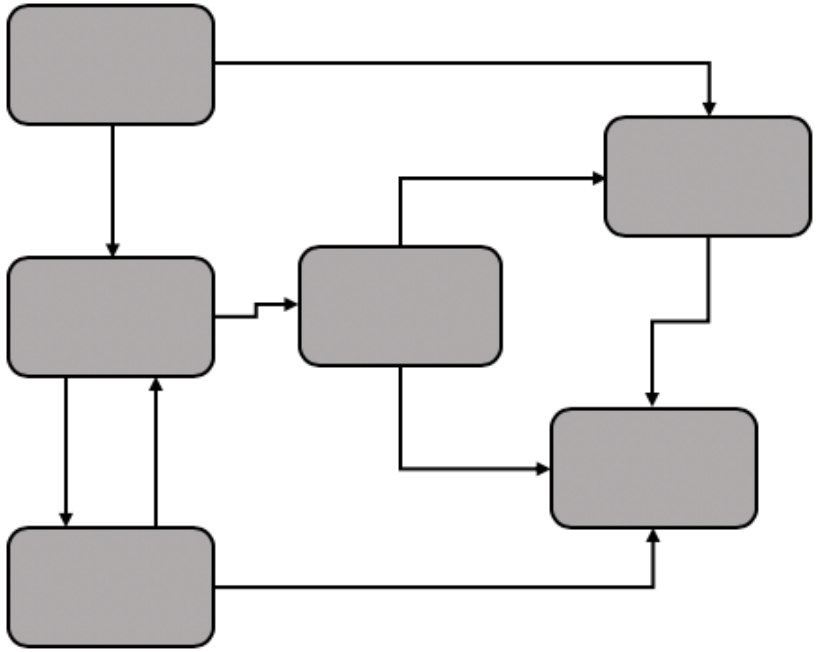
- 흐름을 기반으로 프로그래밍
- 흐름, 처리를 기준으로 프로그램을 설계

Object Oriented Programming

- 객체를 기반으로 프로그래밍
- 객체, 자료를 기준으로 프로그램을 설계

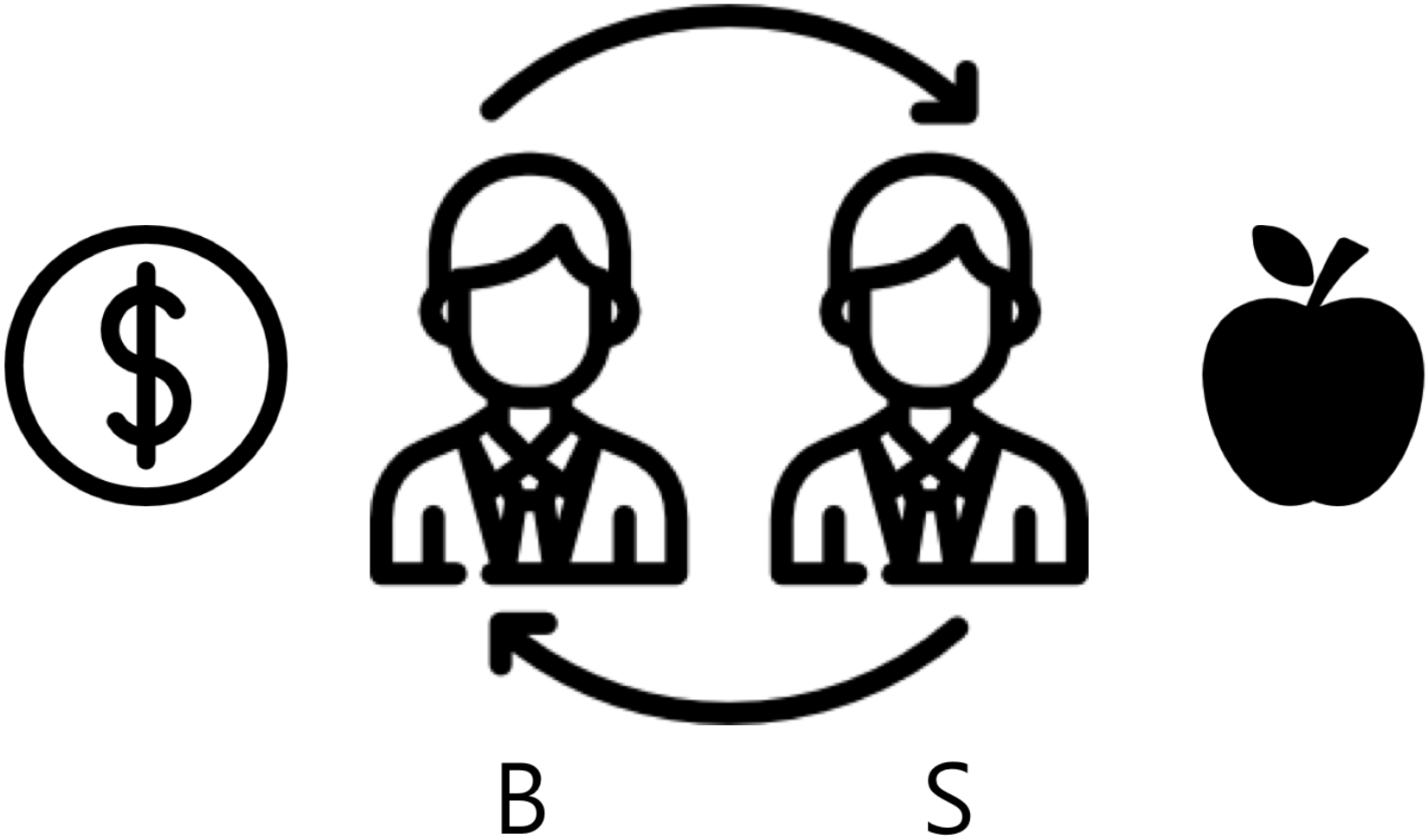
Structure





- 왼쪽이 절차지향, 오른쪽이 객체지향

Example: Exchanging

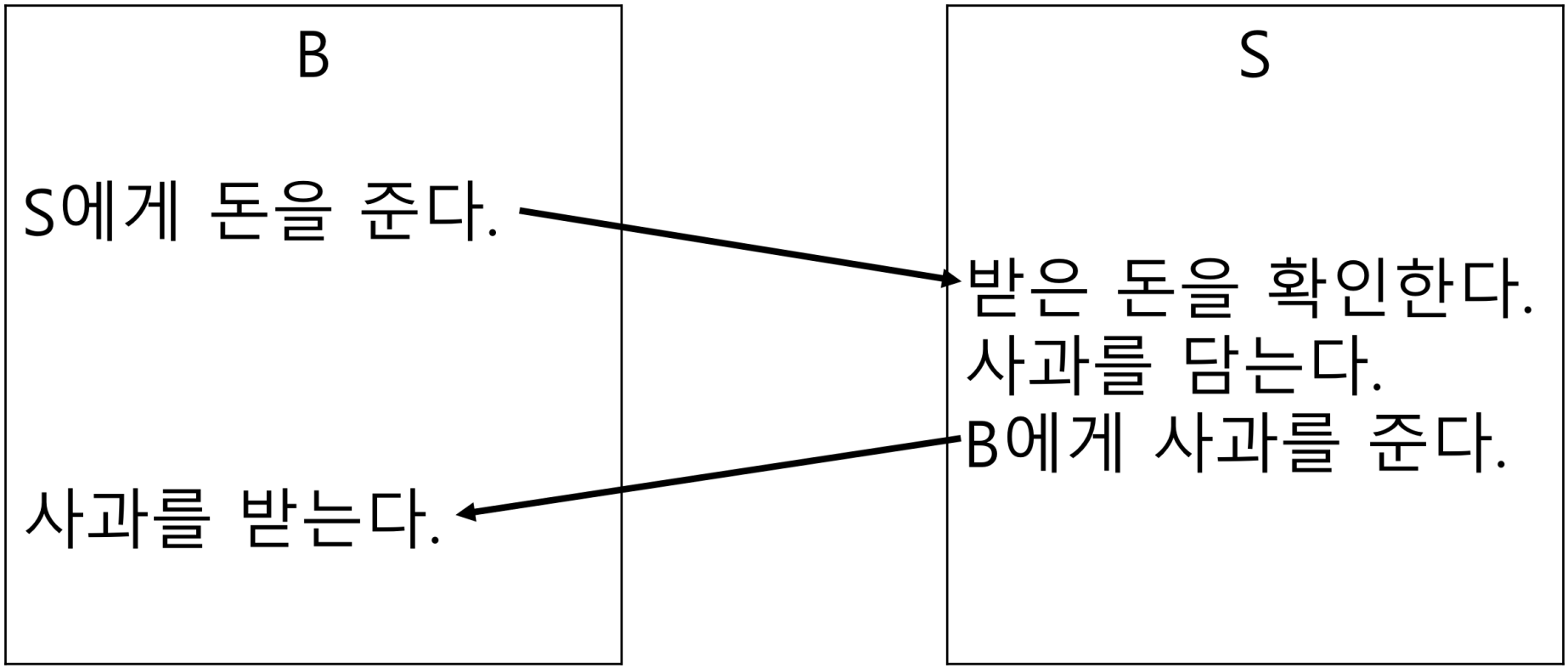


- 사과 거래

In Procedural...

1. B가 S에게 돈을 준다
2. S가 돈을 확인한다
3. S가 사과를 담는다
4. S가 B에게 사과를 준다

In OOP...



Difference

절차지향

- 먼저 흐름을 설계한 후, 필요한 데이터를 설계

객체지향

- 먼저 필요한 데이터를 설계한 후, 그들 사이의 흐름(관계)을 설계

http://blog.naver.com/atalanta16/220249264429

OOP

- 객체(Object): 정보를 저장하고 수행하는 단위
  - B, S, 사과, 돈
- 클래스(class): 객체를 설계하기 위한 틀
- 인스턴스(instance): 클래스를 바탕으로 생성된 객체

Class

- 어떤 정보를 담을지, 어떤 기능을 수행할지를 정의하는 틀.
- 여기서 정보는 변수, 기능은 메소드를 뜻함
- 변수 $\rightarrow$ 배열 $\rightarrow$ 구조체 $\rightarrow$ 클래스

Class Example

```
class Student
{
    public String  name;
    public int     studentNumber;
    public String  nationality;
    public int     gender;

    public String getName()
    {
        return this.name;
    }
    public int getSN()
    {
        return this.studentNumber;
    }
    public String getGender()
    {
        return this.gender;
    }
    public String getNationality()
    {
        return this.nationality;
    }
}
```

Instance

- 클래스를 기반으로 생성한 객체
- (클래스명) (변수명) = new (클래스명)();
- 인스턴스 내부의 값을 사용하는 방법은 C언어의 구조체와 같음

Instance Example

```
class test
{
    ▶ Run | 🐞 Debug
    public static void main(String[] args)
    {
        Student s = new Student();

        s.name = "김민수";
        s.studentNumber = 2017320168;
        s.nationality = "Korea";
        s.gender = "male";
```

Instance Example

```
        System.out.println(s.name);
        System.out.println(s.studentNumber);
        System.out.println(s.nationality);
        System.out.println(s.gender);
    }
}
```

- 김민수
- 2017320168
- Korea
- male

Constructor

- 인스턴스를 생성하는 메소드
- 인스턴스에 필요한 메모리를 할당하는 역할을 함
- 인스턴스 내의 변수들의 값을 초기화해주는 역할을 함
- 인스턴스가 생성될 때 한 번만 실행
- class name == constructor name

Constructor

- 인스턴스를 생성할 때,
- (클래스명) (변수명) = new (클래스명)();이 사실은
- (클래스명) (변수명) = new (Constructor);을 뜻함
- 이전 예시인 Student class에도 Constructor가 존재

Constructor

- But, 명시적으로 선언하지는 않음
- 명시적으로 선언해주지 않으면 default Constructor가 class 내에 자동 생성

Constructor Example

```
public Student(String name, int studentNumber,
               String nationality, String gender)
{
    this.name = name;
    this.age = age;
    this.gender = gender;
    this.nationality = nationality;
}
```

- 이 Constructor를 추가해주면...

Constructor Example

```
class test
{
    ▶ Run | 🐞 Debug
    public static void main(String[] args)
    {
        Student kim = new Student("김민수", 2017320168, "Korea", "male");
        Student lee = new Student("이수현", 2017320117, "China", "male");
        Student asd = new Student();    //! Compile error!
    }
}
```

- 이전에 인스턴스 내부의 변수의 값을 하나씩 각각 초기화 했던 것보다 편리하게 초기화 가능
- 여러 사람들에 대한 인스턴스를 편리하게 생성 가능
- Constructor를 명시적으로 선언해주었으므로, default Constructor를 호출하게 되면 compile error

Constructor

- Constructor도 결국 메소드이기 때문에, 오버로딩 가능

Constructor Example

```
class test
{
    int     num;
    String  color;

    test()
    {
        this.num = 1;
        this.color = "RED";
    }
    test(int num)
    {
        this.color = "RED";
    }
    test(String color)
    {
        this.num = 1;
    }
    test(int num, String color)
    {
        this.num = num;
        this.color = color;
    }
}
```

This

- 한 인스턴스에서, 그 인스턴스 내의 변수에 접근할 때 사용

This Example

```
class This
{
    int      n = 20;

    void asd()
    {
        int      n = 10;
        System.out.println(n);
        System.out.println(this.n);
    }
}
```

This Example

```
public static void main(String[] args)
{
    This     asd = new This();
    asd.asd();
}
```

- 10
- 20

Reference Type

- 이전의 8개의 자료형과 더불어, 참조 타입이 존재
- 참조 타입은 클래스 이름과 같기 때문에 그 종류에 제한이 없음
- Student, Scanner, ...
- 참조 타입으로 선언된 변수는 참조 변수
- 참조 변수 내에는 인스턴스의 메모리의 주소가 저장됨
- 참조 변수를 메소드의 매개변수로 넣으면 참조 변수에 저장된 주소가 전달됨
- C언어의 포인터와 비슷하다고 생각

Example

```java
class asd
{
    int     n = 10;
    ▶ Run | 🐞 Debug
    public static void main(String[] args)
    {
        asd     A = new asd();
        int     n = 10;
        A.change(A, n);
        System.out.println(A.n);
        System.out.println(n);
    }
    void change(asd A, int n)
    {
        A.n = 0;
        n = 0;
    }
}
```

- 0
- 10

END

-끝-