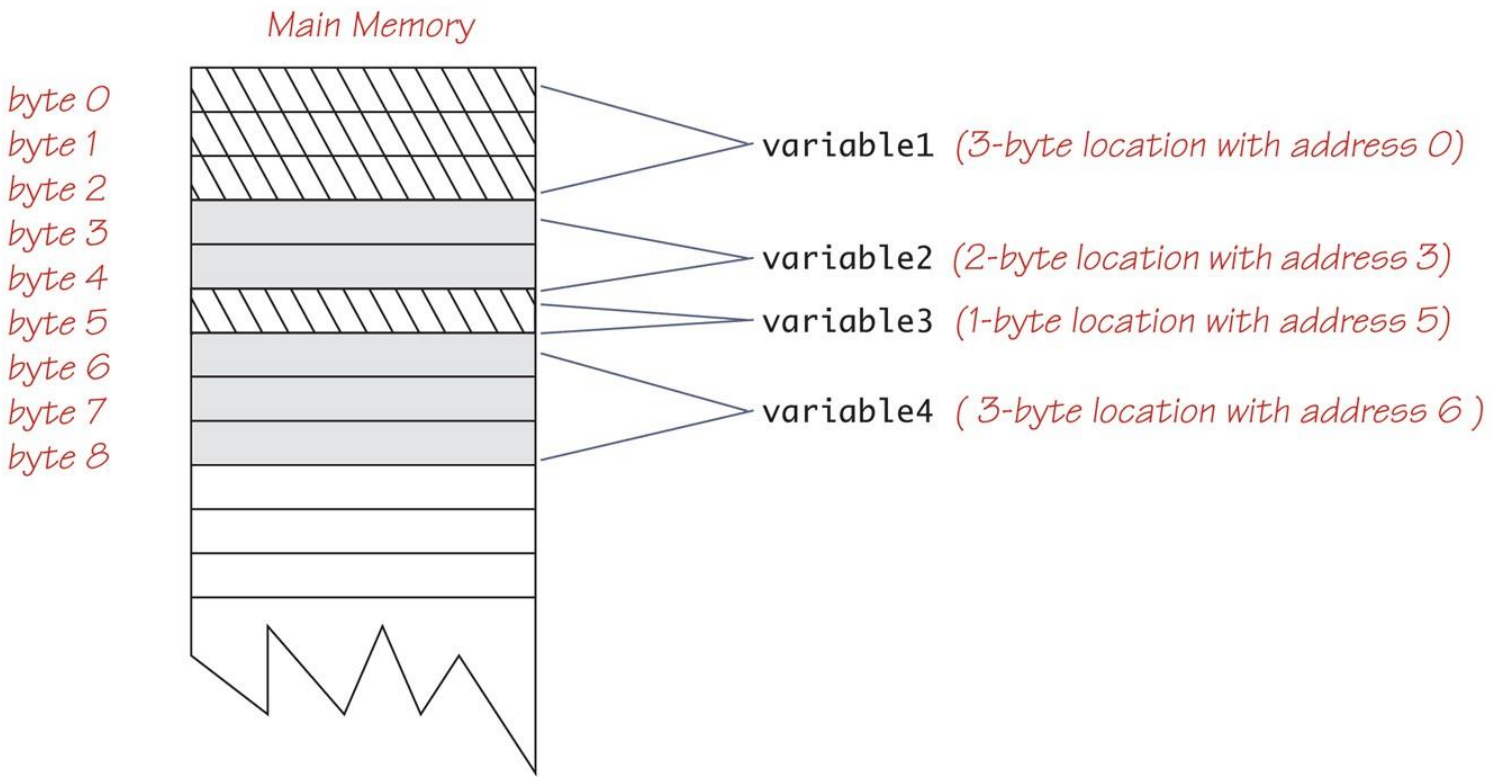

# Object – Oriented Programming

## LAB #5. CLASS 3

# Variables and Memory

- 메인 메모리는 데이터를 byte를 연속적으로 저장하는데 사용된다.
- 첫번째 byte의 주소는 데이터의 주소이다.

**Display 5.10 Variables in Memory**



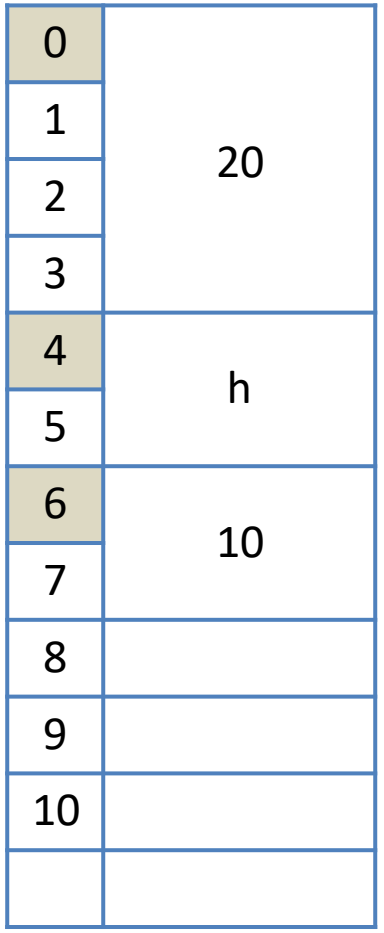
# Variables and Memory

- 기본형 타입은 실제 값을 메모리에 저장한다.

int i = 20;

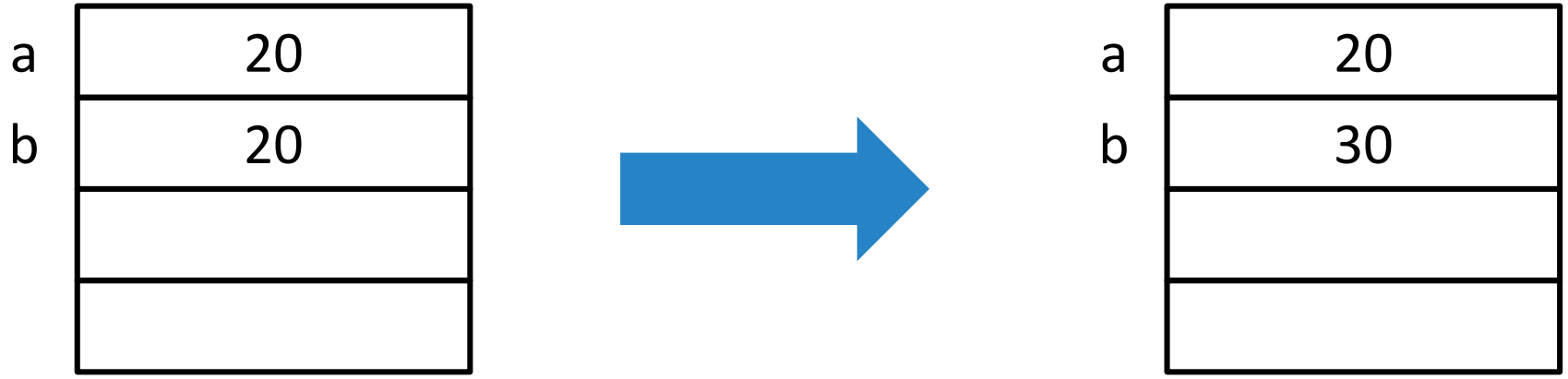
char c = ‘h’;

short sh = 10;



# Variables and Memory

```
int a, b;
a = 20;
b = a;
System.out.println("a:" + a);
System.out.println("b:" + b);
b = 30;
System.out.println("a:" + a);
System.out.println("b:" + b);
```



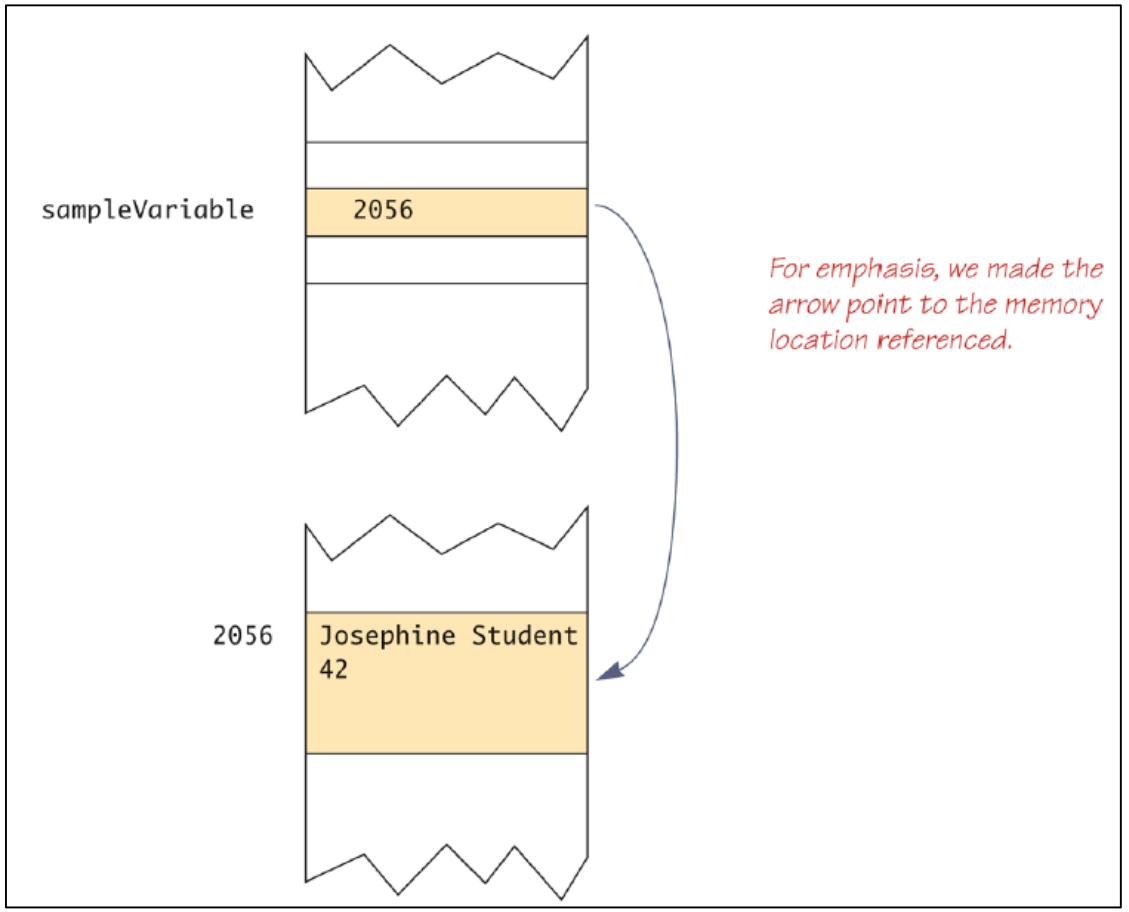
# Variables and Memory

- 클래스 타입은 memory address를 메모리에 저장한다.

ToyClass sampleVariable;

sampleVariable =

  new ToyClass(“Josephine Student“, 42);



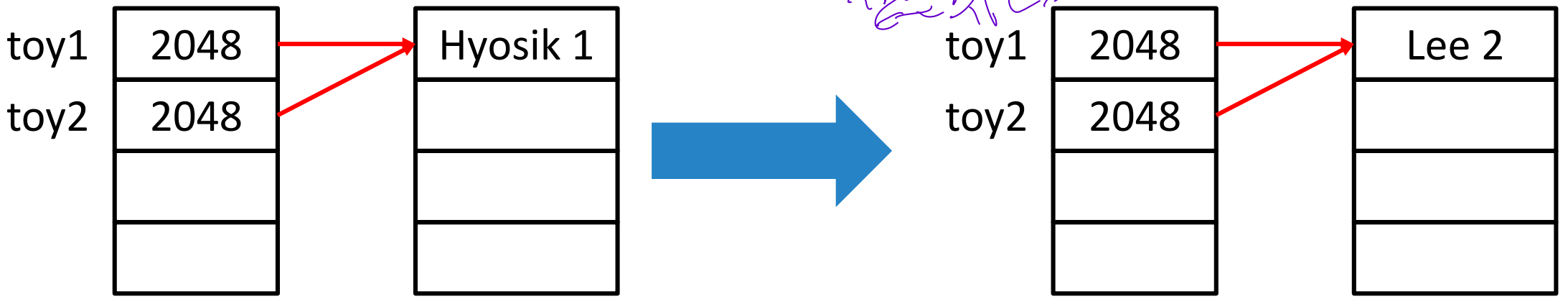
# ToyClass

```
private String name;

private int number;


public ToyClass(String initialName, int initialNumber){

   name = initialName;

   number = initialNumber;

}

public ToyClass(){

   name = "No name yet.";

   number = 0;

}

public void set(String newName, int newNumber){

   name = newName;

   number = newNumber;

}
```

```
public String toString(){

   return (name + " " + number);

}


public static void changer(ToyClass aParameter){

   aParameter.name = "Hot Shot";

   aParameter.number = 42;

}


public boolean equals(ToyClass otherObject){

 return ((name.equals(otherObject.name)) &&
                (number ==   otherObject.number) );

}
```

# Variables and Memory

```
ToyClass toy1, toy2;
toy1 = new ToyClass("Hyosik", 1);
toy2 = toy1;
System.out.println("toy1: " + toy1);
System.out.println("toy2: " + toy2);
toy2.set("Lee", 2);
System.out.println("toy1: " + toy1);
System.out.println("toy2: " + toy2);
```



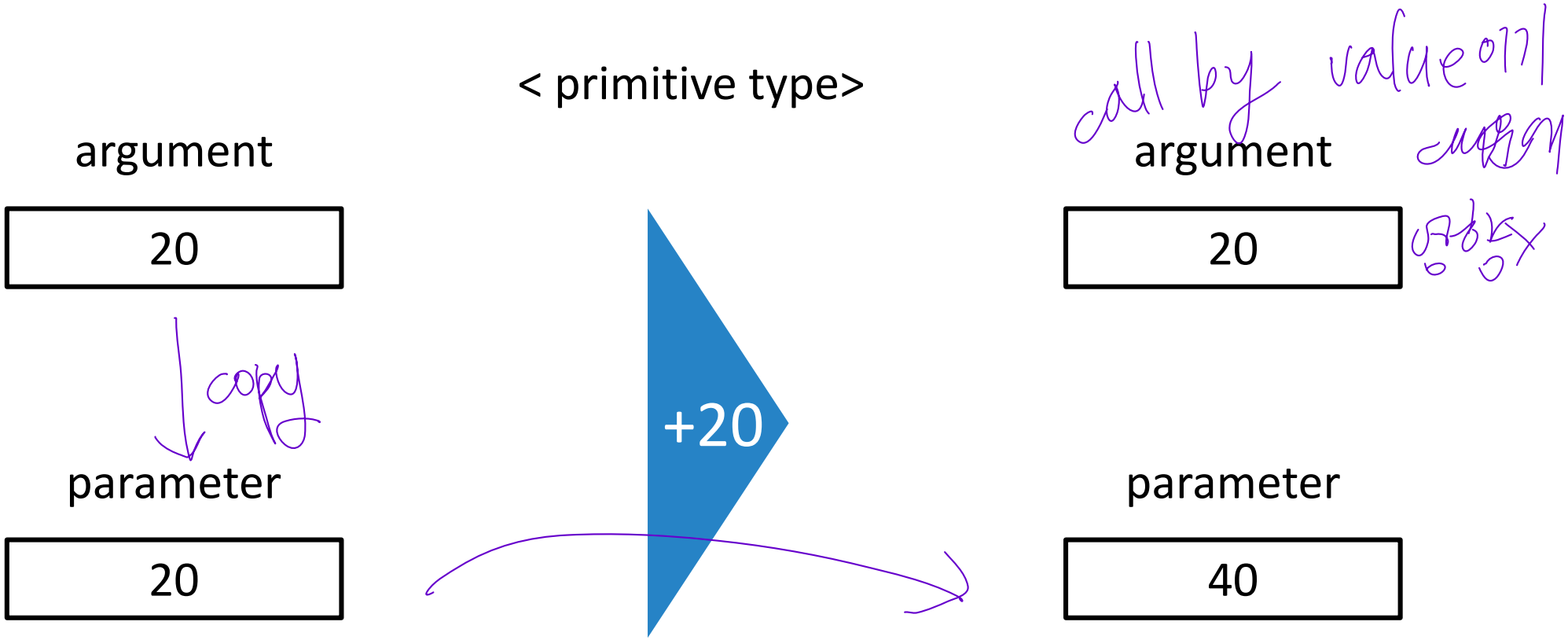
# Parameters

- Java의 parameter는 모두 call-by-value parameter

- 때문에, primitive type의 parameter의 값 변화는 argument의 값에 영향을 주지 않음

- 반면에, class type의 parameter의 값 변화는 argument의 값에 영향을 미침

# Parameters




**primitive type의 parameter의 값 변화는 argument의 값에 영향을 주지 않음**

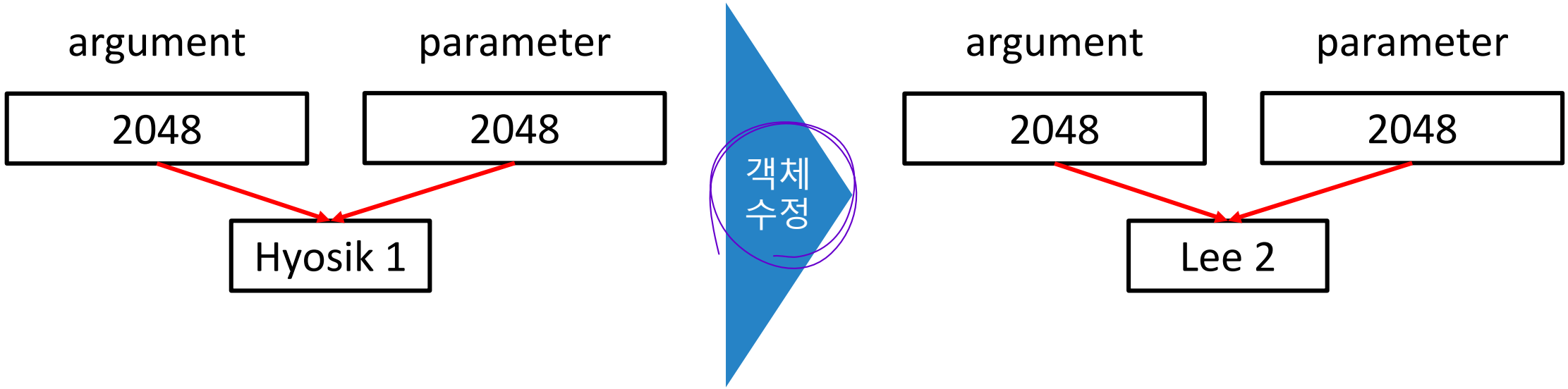
# Parameters

```
public static void add20(int aParameter) {
    System.out.println("aParameter: " + aParameter);
    aParameter += 20;
    System.out.println("aParameter: " + aParameter);
}
```

```
int a = 20;
System.out.println("a: " + a);
add20(a);
System.out.println("a: " + a);
```

# Parameters

< class type>




**class type의 parameter의 값 변화는 argument의 값에 영향을 미침**

# Parameters

```
public static void changer(ToyClass aParameter){
    aParameter.name = "Hot Shot";
    aParameter.number = 42;
}
```

```
ToyClass.changer(toy1);
System.out.println("toy1: " + toy1);
```

# ‘==’ and ‘equals’

- Class type에서 ‘==‘는 단순히 객체의 address를 비교

- 객체를 비교하기 위해서는 equals 사용

```
ToyClass toy1, toy2, toy3;
toy1 = new ToyClass("Hyosik", 1);
toy2 = toy1;
toy3 = new ToyClass("Hyosik", 1);

System.out.println(toy1 == toy2);
System.out.println(toy1 == toy3);
System.out.println(toy1.equals(toy3));
```

# null

- 클래스 유형의 변수에 할당될 수 있는 특수한 상수
  - String str = null;


- Null은 객체가 아니다.
- Null은 메모리 주소와 유사하며 변수에 null이 포함되어 있는지 테스트 하려면 ==, != 연산자를 사용해야 한다.
  - if (str == null)

# new

- 객체를 초기화하는 생성자를 호출하고 생성된 객체의 메모리의 주소를 반환한다.
  - 이 메모리의 주소를 클래스 유형의 변수에 할당한다.

- Anonymous Object : new 객체로 생성하여 반환 받은 주소를 변수에 할당하지 않는 것

```
if (variable1.equals(new ToyClass(“Joe”, 42)))
    System.out.println(“Equal”);
else
    System.out.println(“Not equal”);
```

# Date

```
private String month;
private int day;
private int year;

public Date(String month, int day, int year) {
    setDate(month, day, year);
}

public void setDate(String month, int day, int year) {
    this.month = month;
    this.day = day;
    this.year = year;
}
```

```
public String toString() {
    return (month + " " + day + ", " + year);
}

public boolean equals(Date otherDate) {
    return ((month.equals(otherDate.month))
            && (day == otherDate.day) && (year == otherDate.year) );

}
```

# Copy Constructors

- 기존의 생성한 객체를 복사하기 위한 생성자
- 클래스와 동일한 타입의 매개변수를 받는 생성자

```
public Date(Date aDate){
   if (aDate == null){ //Not a real date.
      System.out.println("Fatal Error.");
      System.exit(0);
   }

   month = aDate.month;
   day = aDate.day;
   year = aDate.year;
}
```

# Copy Constructors

```
Date date1 = new Date("April", 16, 2020);

Date date2 = date1;
Date date3 = new Date(date1);

System.out.println(date1);
System.out.println(date2);
System.out.println(date3);

date1.setDate("January", 1, 2019);

System.out.println(date1);
System.out.println(date2);
System.out.println(date3);
```

# Person

```
private String name;
private Date born;
private Date died;
public Person(String name, Date born, Date died) {
   this.name = name;
   this.born = born;
   this.died = died;
}
public String toString(){
   String diedString;
   if (died == null )
      diedString = ""; //Empty string
   else
      diedString = died.toString();
   return (name + ", " + born + "-" + diedString); }
```

```
public boolean equals(Person otherPerson){
   if (otherPerson == null )
      return false ;
   else if(died == null)
      return (name.equals(otherPerson.name)
                  && born.equals(otherPerson.born)
                  && otherPerson.died == null);
   else
      return (name.equals(otherPerson.name)
                  && born.equals(otherPerson.born)
                  && died.equals(otherPerson.died));
}
public Date getBorn() {
   return born;
}
```

# Person - Copy Constructors(Dangerous)

```
public Person(Person aPerson){
  if (aPerson == null) { //Not a real date.
    System.out.println("Fatal Error.");
    System.exit(0);
  }

  name = aPerson.name;
  born = aPerson.born;
  if(died == null)
     died = null;
  else
     died = aPerson.died;
}
```

# Person - Copy Constructors(Dangerous)

- Copy한 객체의 변화가 다른 객체에도 영향을 미침

```
Person person1, person2;
person1 = new Person("Lee", new Date("January", 1, 2020), null);
person2 = new Person(person1);
System.out.println(person1);
System.out.println(person2);
person2.getBorn().setDate("April", 16, 2020);
System.out.println(person1);
System.out.println(person2);
```

# Person - Copy Constructors(Safe)

```
public Person(Person aPerson){
  if (aPerson == null) { //Not a real date.
    System.out.println("Fatal Error.");
    System.exit(0);
  }

  name = aPerson.name;
  born = new Date(aPerson.born);
  if(died == null)
     died = null;
  else
     died = new Date(aPerson.died);
}
```

# Person – getter(Dangerous)

- private 변수임에도 불구하고 수정 가능

```
Person person3;
person3 = new Person("Lee", new Date("January", 1, 2020), null);
System.out.println(person3);
person3.getBorn().setDate("April", 16, 2020);
System.out.println(person3);
```

# Person - getter(Safe)

```
public getBorn(){
      return new Date(born);
}
```

# Deep Copy vs Shallow Copy

- Deep Copy : 원본과 공통되는 참조가 없는 복사

```
public getBorn(){
      return new Date(born);
}
```

- Shallow Copy : 그 외의 복사
  이 경우의 두 변수가 같은 참조를 가질 수가 있어, 보안상의 위험이 있다.

```
public getBorn(){
      return born;
}
```

# Package

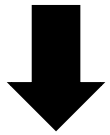
- 서로 관련 있는 클래스들의 모음

- 프로젝트를 편리하게 관리하도록 함

- 다른 라이브러리들끼리 구분 가능

- import 키워드를 사용하여 클래스를 간단히 사용 가능

```
public class Program {
    public static void main(String[] args) {
        java.util.Scanner scan = new java.util.Scanner(System.in);

    }
}
```

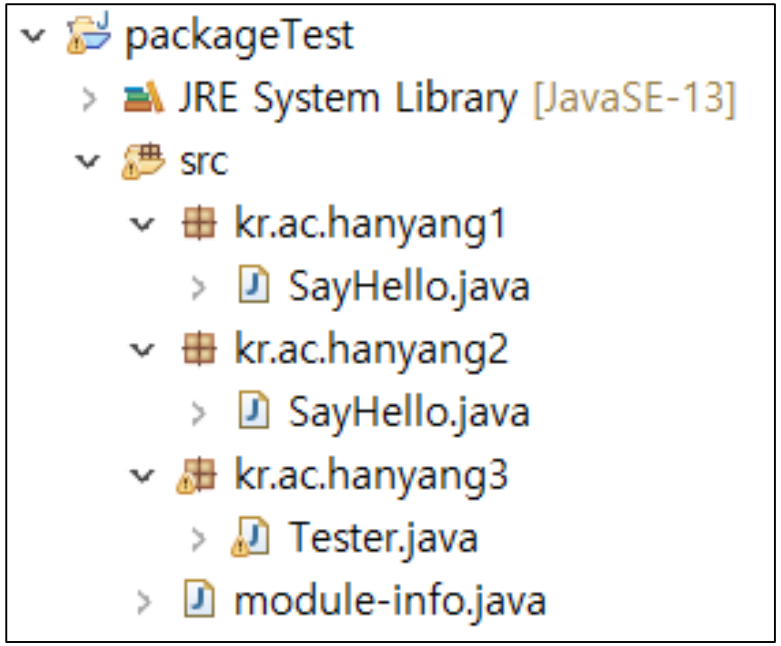


```
import java.util.Scanner;

public class Program {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);

    }
}
```

# Package – Name Clash

```java
package kr.ac.hanyang1;

public class SayHello {

    public void sayPackage() {
        System.out.println("This is hanyang1.");
    }
}
```

```java
package kr.ac.hanyang2;

public class SayHello {

    public void sayPackage() {
        System.out.println("This is hanyang2.");
    }
}
```



# Package – Name Clash

```java
package kr.ac.hanyang3;

import kr.ac.hanyang1.*;
import kr.ac.hanyang2.*;

public class Tester {

    public static void main(String[] args) {

        kr.ac.hanyang1.SayHello test1 = new kr.ac.hanyang1.SayHello();
        kr.ac.hanyang2.SayHello test2 = new kr.ac.hanyang2.SayHello();

        test1.sayPackage();
        test2.sayPackage();

    }
}
```

# Date

- java.util 패키지에 있는 클래스

- 날짜와 시간에 관한 정보를
  표현하는 클래스

- 현재 JAVA 버전에서는 사용되지 않음
  → Calendar 클래스로 대체

```
Date today = new Date();
System.out.println(today);

int year = today.getYear()+1900;
System.out.println(year);

int month = today.getMonth()+1;
System.out.println(month);

int day = today.getDay()+10;
System.out.println(day);

int hours = today.getHours();
System.out.println(hours);

int minutes = today.getMinutes();
System.out.println(minutes);
```

# Calendar

- java.util 패키지에 있는 클래스

- 날짜와 시간에 관한 정보를 표현하는 클래스

- Date 클래스의 대체물로 나온 클래스

- Month 는 1을 더해줘야 정상적인 값이 나옴

- Immutable하지 않음

```
Calendar cal = Calendar.getInstance();

int year = cal.get(Calendar.YEAR);
System.out.println(year);

int month = cal.get(Calendar.MONTH) + 1;
System.out.println(month);

int day = cal.get(Calendar.DAY_OF_MONTH);
System.out.println(day);

int hours = cal.get(Calendar.HOUR_OF_DAY);
System.out.println(hours);

int minutes = cal.get(Calendar.MINUTE);
System.out.println(minutes);
```

# LocalDateTime

- jdk 1.8부터 지원

- java.time 패키지에 있는 클래스

- 날짜와 시간에 관한 정보를
  표현하는 클래스

```
LocalDateTime currentDateTime = LocalDateTime.now();
LocalDateTime myDateTime = LocalDateTime.of(2020, 4, 16, 10, 00);

int year = myDateTime.getYear();
System.out.println(year);                                    // 2020

Month month1 = myDateTime.getMonth();
System.out.println(month1);                                  // APRIL
int month2 = myDateTime.getMonthValue();
System.out.println(month2);                                  // 4

int day1 = myDateTime.getDayOfMonth();
System.out.println(day1);                                    // 16
DayOfWeek day2 = myDateTime.getDayOfWeek();
System.out.println(day2);                                    // THURSDAY
int day3 = myDateTime.getDayOfYear();
System.out.println(day3);                                    // 107
```

- 추가적인 method는 다음을 참고
  https://docs.oracle.com/javase/8/docs/api/java/time/LocalDateTime.html

# LocalDateTime

```
LocalDateTime classDateTime = LocalDateTime.of(LocalDate.of(2020, 4, 16), LocalTime.of(10, 00));
LocalDateTime currentDateTime = LocalDateTime.now();

System.out.println(currentDateTime.isAfter(classDateTime));                    // false
System.out.println(currentDateTime.isBefore(classDateTime));                   // true
System.out.println(currentDateTime.isEqual(classDateTime));                    // false

LocalDateTime minusDateTime = classDateTime.minusDays(5);
System.out.println(classDateTime);                                  // 2020-04-16T10:00
System.out.println(minusDateTime);                                  // 2020-04-11T10:00
// minusDays, minusWeeks, minusMonths, minusYears
// minusHours, minusMinutes,  minusseconds, minusNanos

LocalDateTime plusDtaeTime = classDateTime.plusDays(5);
System.out.println(classDateTime);                                  // 2020-04-16T10:00
System.out.println(plusDtaeTime);                                   // 2020-04-21T10:00
// plusDays, plusWeeks, plusMonths, plusYears
// plusHours, plusMinutes,  plusseconds, plusNanos
```

# LocalDate

- jdk 1.8부터 지원

- java.time 패키지에 있는 클래스

- 날짜에 관한 정보를
  표현하는 클래스

```
LocalDate currentDate = LocalDate.now();
LocalDate myDate = LocalDate.of(2020, 4, 16);

int year = myDate.getYear();
System.out.println(year);                          //2020

Month month1 = myDate.getMonth();
System.out.println(month1);                        // APRIL
int month2 = myDate.getMonthValue();
System.out.println(month2);                        // 4

int day1 = myDate.getDayOfMonth();
System.out.println(day1);                          // 16
DayOfWeek day2 = myDate.getDayOfWeek();
System.out.println(day2);                          // THURSDAY
int day3 = myDate.getDayOfYear();
System.out.println(day3);                          // 107
```

- 추가적인 method는 다음을 참고

  https://docs.oracle.com/javase/8/docs/api/java/time/LocalDate.html

# LocalTime

- jdk 1.8부터 지원

- java.time 패키지에 있는 클래스

- 시간에 관한 정보를
  표현하는 클래스

```
LocalTime currentTime = LocalTime.now();
LocalTime myTime = LocalTime.of(10, 30);
System.out.println(myTime);                                    // 10:30
LocalTime myTime2 = LocalTime.of(10, 30, 15, 28);
System.out.println(myTime2);                          // 10:30:15.000000028

int hour = myTime.getHour();
System.out.println(hour);                                      // 10
int minute = myTime.getMinute();
System.out.println(minute);                                    // 30
int second = myTime2.getSecond();
System.out.println(second);                                    // 15
int nano = myTime2.getNano();
System.out.println(nano);                                      // 28
```

- 추가적인 method는 다음을 참고

https://docs.oracle.com/javase/8/docs/api/java/time/LocalTime.html

# 실습 과제

1. kr.co.lab05.employee 패키지를 추가하고 Employee 클래스 생성
2. 다음 내용을 참고하여 Employee클래스를 구현

| **Employee** | |
|---|---|
| Private | - name : String<br>- yearly_salary : double<br>- monthly_salary : double<br>- balance : double |
| Public | + Employee(name : String, yearly_salary : double)<br>+ getBalance() : double<br>+ increaseYearlySalary(byPercent : int) : void<br>+ receiveSalary () : void<br>+ toString() : String |

# 실습 과제

## 3. Employee 클래스의 Method 설명

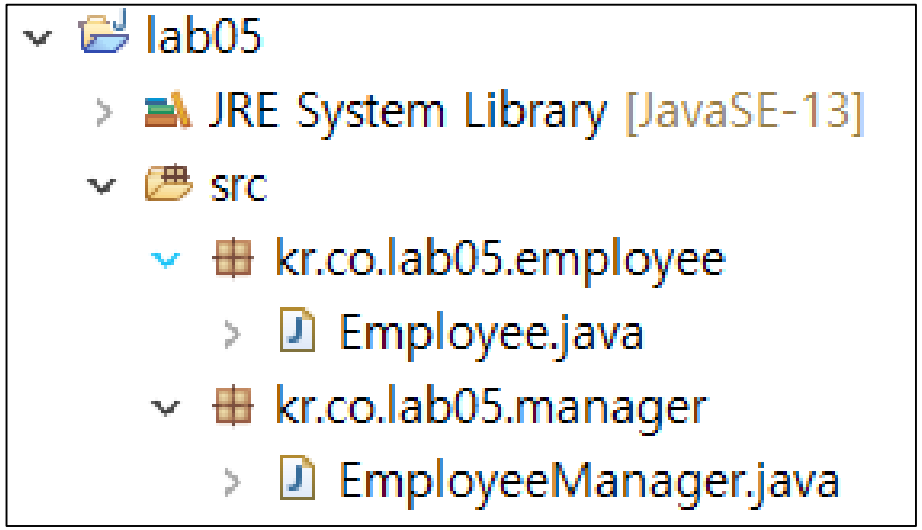
- Employee 생성자에서는 이름과 연봉을 매개변수로 전달받고 속성의 값으로 초기화한다.
  - monthly_salary는 연봉 / 12의 값으로 초기화하고 balance는 0으로 초기화한다.
- 연봉을 반환하는 getter를 생성한다.
  - getBalance()
- 연봉을 증가시켜주는 Method를 생성한다.
  - 매개변수로 받은 값을 백분율로 변환하여 현재 yearly_salary 값을 해당 비율만큼 증가시킨다.
  - ex) 매개변수의 값이 10이고 연봉이 4000일 경우, 연봉에 400을 더한다.
  - monthly_salary도 수정해주어야한다.
- 월급을 받는 Method를 생성한다.
  - monthly_salary 값을 balance값에 더해준다.
- toString Method를 생성한다.
  - return 값 예시 - 이름 : Lee 연봉 : 4500.0 월급 : 375.0 재산 : 0.0

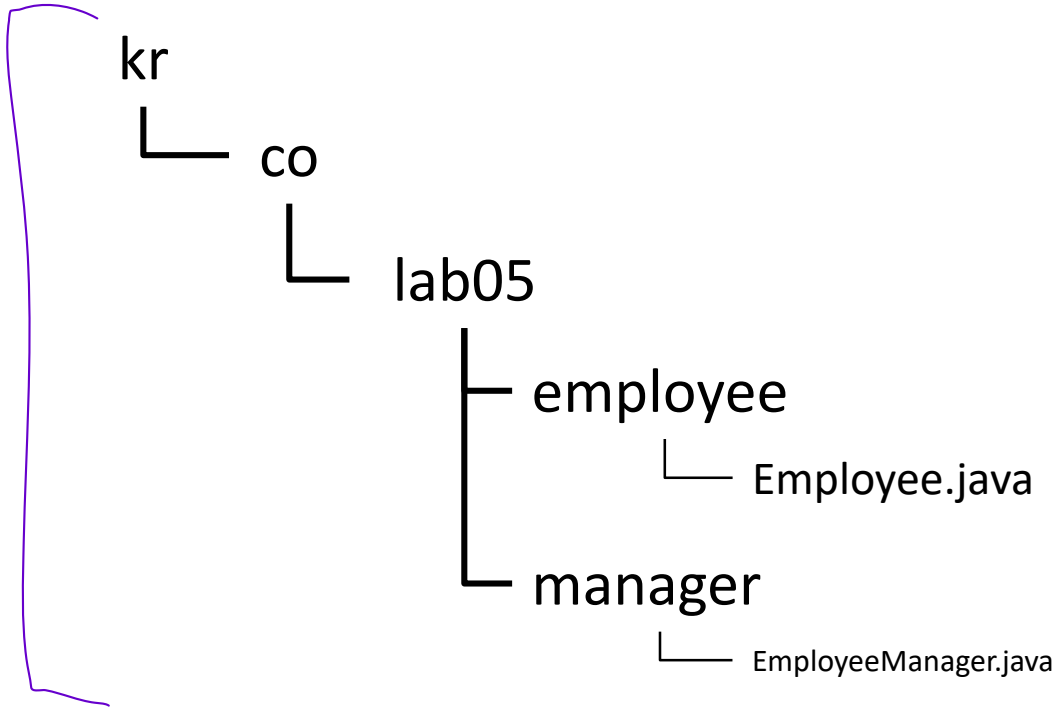
# 실습 과제

4. EmployeeManager Class 생성 (kr.co.lab05.manager 패키지에 위치)
   - 해당 클래스는 Main Method를 구현한다.
   - Main Method
     - 직원을 생성한다. (이름 : 자신의 성 / 연봉 : 4500 )
     - 계약일을 LocalDate.of()를 사용해서 생성하고 출력한다. ( 계약일 : 2020-04-16 )
     - 해당 직원의 모든 정보를 출력한다. ( ex) 이름 : Lee, 연봉 : 4500.0, 월급 : 375.0, 재산 : 0.0 )
     - 인센티브를 부여할 랜덤한 월을 정하는 난수를 생성한다. ( 범위 1 ~ 12 )
     - 직원의 재산이 20000이 되기 전까지 아래 활동을 반복하는 반복문을 생성한다.
       - 매달 지날수록 직원은 월급을 받는다. ( Employee의 receiveSalary() Method 사용 )
       - 현재의 Month가 인센티브를 받는 Month일 경우,
         - 인센티브는 월급의 100%를 받는다.
         - 인센티브를 받았다는 정보를 출력한다. ( ex) 1년차 5월에 인센티브를 받았습니다. )
           - 처음 입사할 경우 1년차 인 것으로 설정할 것
       - 근무한지 12개월이 지났을 경우 연봉을 증가시킨다.
         - 0~10까지의 난수를 생성한다.
         - 해당 난수 퍼센트만큼 연봉을 증가시킨다. ( Employee의 increaseYearlySalary(bypercent : int) Method 사용 )
         - 연봉이 올랐음을 출력한다. ( ex) 2년차 연봉이 3% 인상되었습니다.)
         - 인센티브를 부여할 월을 다시 랜덤 값으로 초기화한다. ( 범위 1 ~ 12 )
     - 해당 직원의 재산이 20000 이상이 되었을 날짜를 출력한다.
     - 해당 직원의 모든 정보를 출력한다. ( ex) Name : Lee 연봉 : 4500.0 월급 : 375.0 재산 : 0.0 )

# 실습 과제

## 예시 구조




제출: kr 폴더를 압축해서 제출

# 실습 과제

## 예시 출력

```
계약일: 2020-04-16
이름: Lee, 연봉: 4500.0, 월급: 375.0, 재산: 0.0
1년차 10월에 인센티브를 받았습니다.
2년차 5% 인상되었습니다.
2년차 7월에 인센티브를 받았습니다.
3년차 5% 인상되었습니다.
3년차 2월에 인센티브를 받았습니다.
4년차 0% 인상되었습니다.
4년차 9월에 인센티브를 받았습니다.
날짜: 2024-03-16
이름: Lee, 연봉: 4961.25, 월급: 413.4375, 재산: 20329.6875
```

5주차 실습 과제 관련 질문: lhs9394@naver.com