04_1 Class and Object

Object-Oriented Programming

Class와 Object에 대해 강의하겠습니다.

Class

- The most important feature for OOP in Java

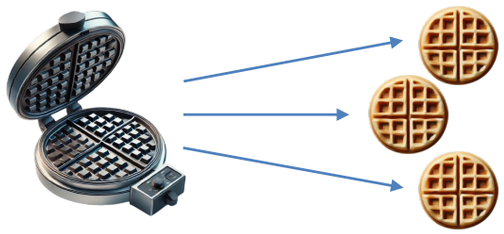
- Programming in Java
  - By defining several classes
  - Everything should be in a class
  - All programmer-defined types are classes

페이지 2

Class는 Java의 OOP 요소 중 가장 중요한 것 중의 하나 입니다.
단순히 말해서 Java 프로그래밍을 한다는 것은
Class들을 구현해 나가는 것이고
모든 code가 어떤 class 내에 존재해야 하며,
프로그래머가 새로 만들어 내는 type은 모두 다 class 입니다.

Class vs Object

- A class is a template of an object
- ex)
  - class = waffle maker (와플메이커)
  - object = waffle (와플) made with that waffle maker



- **Object** and **Instance** are often used interchangeably

페이지 3

그렇다면, class와 object는 어떻게 다를까요?
Class는 object를 만들어 낼 수 있는 template이라고 할 수 있습니다.
예를 들자면 class를 와플메이커라고 한다면
object는 그 와플메이커로 만들어 낸 와플이라 할 수 있습니다.
하나의 와플메이커로 수 많은 와플을 구워낼 수 있듯이
하나의 class로 무한정한 오브젝트들을 생성해 낼 수 있습니다.
그럼, instance는 무슨 뜻일까요?
instance는 class과 구체화 된 하나의 개체라는 뜻으로
object와 거의 같은 뜻의 용어라고 볼 수 있겠습니다.
실제로 object와 instance는 같은 의미로 사용됩니다.

Creating Object

```java
class Student {
    String name;  // instance variable
    int id;       // instance variable
}

public class StudentDemo {
    public static void main(String[] args) {
        Student s1 = new Student(); // s1: object (instance)
        Student s2 = new Student(); // s2: object (instance)
    }
}
```

s1 and s2 are the instance (object) of class Student

페이지 4

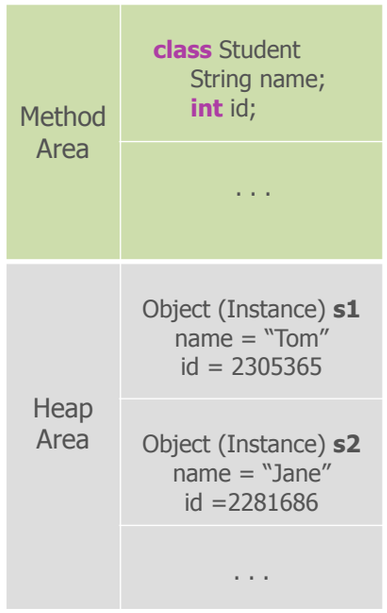
Student class에는 두 개의 instance variable이 있습니다.
name은 학생의 이름을 나타내는 String variable이고
id는 학번을 나타내는 int variable입니다.
StudentDemo class의 main method에서
s1과 s2라는 Student class의 object 두 개를 생성합니다.
object를 생성할 때에는 new 명령어를 사용합니다.

Instance Variables

```
class Student {
    String name;  // instance variable
    int id;       // instance variable
}

...

Student s1 = new Student(); // s1: object (instance)
Student s2 = new Student(); // s2: object (instance)
s1.name = "Tom";
s1.id = 2305365;
s2.name = "Jane";
s2.id = 2281686;
```



페이지 5

Student class의 첫 instance인 s1를 생성하는 순간에
class Student에 대한 정보가 JVM의 method area에 생성됩니다.
이제 instance가 new 명령을 통해 생성될 때마다
instance 즉 object를 위한 memory가 heap area에 잡히게 됩니다.
이렇게 instance들이 memory의 독립적인 공간으로 분리되니까
각 instance의 instance variable에는
다른 value들이 assign 될 수 있습니다.
예를 들면 s1의 name은 "Tom", s1의 id는 2305365,
s2의 name은 "Jane", s2의 id는 2281686 이 될 수 있는 것입니다.
이렇게 instance variable들을 access할 때에는
objectName 쩜 variable 과 같이 합니다.

Using Class Members (1/2)

SampleClass.java

```java
public class SampleClass {
    int x; // instance variable
    void sayHello(int y) { // method
        System.out.println("Hello " + x + " " + y);
    }
}
```

- `class SampleClass` has
  - instance variable: x
  - method: `sayHello`

페이지 6

이제 SampleClass라는 class를 고려해 봅시다.
SampleClass에는 instance variable인 int type x가 있고
sayHello라는 method가 있습니다.
sayHello는 y라는 int parameter를 받아서
instance variable인 x와 y를 더해
Hello x_value y_value 를 프린트 합니다.

Using Class Members (2/2)

ClassDemo.java

```
public class ClassDemo {
    public static void main(String[] args) {

        SampleClass c1 = new SampleClass();
        SampleClass c2 = new SampleClass();


        c1.x = 2;
        c2.x = c1.x + 1;

        c1.sayHello(5);     // Hello 2 5
        c2.sayHello(7);     // Hello 3 7
    }
}
```

The **new** operator: to **create** the object (instance) of the class

Accessing instance variable:
objectName.varName

Calling method:
objectName.methodName()

페이지 7

Class의 멤버인 instance variable과 method의 이용에 대해
정리를 해 보겠습니다.
new operator는 어떤 class의 object를 생성하기 위해 사용합니다.
instance variable의 value를 읽어오거나
instance variable에 어떤 value를 assign할 때에는
dot operator (.) 를 사용하게 됩니다.
Object c1의 x value는 2로 assign되었고
Object c2의 x value는 c1의 x value에 1을 더한 값이니까 3이 됩니다.
비슷하게 어떤 object의 method를 call할 때에도
dot operator (.) 를 사용합니다.
c1의 x value가 2 이므로 "Hello 2 5" 라고 프린트하게 됩니다.
비슷하게 c2.sayHello(7) 은 y가 7, c2의 x가 3이므로
"Hello 3 7" 을 프린트하게 됩니다.

Accessing Class Members Inside the Class

Access without object or class name

```
public class SampleClass {
    int x;
    public void sayHello(int y) {
        System.out.println("Hello " + x + " " + y);
    }
    public int squareX() {
        sayHello(x+1);
        return x * x;
    }
}
```


페이지 8

다시 SampleClass class를 고려하겠습니다.
이 class에 squareX라는 method가 하나 더 추가 되었습니다.
squareX() method 에서는
먼저 sayHello(x+1)을 call하여 "Hello x_value x_value + 1" 을 프린트합니다.
그리고 x의 제곱을 return 합니다.
그런데 좀 이상한 것이 있습니다.
이전 슬라이드에서 instance variable에 접근하거나
method를 call할 때에는
object name을 먼저 쓰고 dot operator를 사용한다고 하였는데
여기에서는 x에 접근하거나 sayHello를 call할 때
dot operator를 사용하지 않은 것을 볼 수 있습니다.
이것은 x와 sayHello를 같은 class인 SampleClass 내에서
접근하기 때문입니다.
앞 슬라이드에서 처럼 instance variable이나 method를
그것들이 속한 class의 외부에서 접근할 때에는
dot operator를 반드시 사용해야 합니다.

Example: Car Rental System (1/4)

```java
public class Car {
    // Fields of a Car class
    String model;        // Instance Variable
    String licensePlate; // Instance Variable
    static int totalCars; // Class (Static) Variable
    static final String COMPANY_NAME = "SuperCar Rentals"; // Named Constant

    // Method to display car details
    void displayCarDetails() {
        System.out.println("Model: " + model + ",
                           License Plate: " + licensePlate);
    }

    // Static method to display total cars
    static void displayTotalCars() {
        System.out.println("Total Cars: " + totalCars);
    }
}
```


페이지 9

이제 자동차 렌탈 시스템 프로그램을 예제로 살펴보도록 하겠습니다.
이 프로그램은 Car, Customer, CarRentalSystem의 세개의 class들로 이루어져 있습니다.
먼저 class Car는 자동차의 정보를 담고 있습니다.
Instance variable로는 model 이름과 번호판을 나타내는 String variable들이 있습니다.
그 아래에는 static int type의 totalCars variable이 있는데
static variable은 class variable이라고도 합니다.
이 static variable은 instance variable과 달리
Car class에 유일하게 존재하고 모든 object들이 공유합니다.
즉, Car class의 모든 object들이 같은 값을 가지게 됩니다.
따라서 static variable은 heap area의 instance 영역이 아니라
method area의 class info에 위치합니다.
static variable인 totalCars는 Car object가 하나 생성될 때마다 1씩 증가하여
어떤 순간의 자동차의 총 숫자를 나타내게 됩니다.
COMPANY_NAME은 이 system에서 유일한 렌탈 시스템 업체 이름이므로
named constant로 정의되어 static final String type이 되고
"SuperCar Rentals" 로 초기화 되어 있습니다.
displayCarDetails() method는
각 Car object의 model과 번호판을 print하는 method입니다.
displayTotalCars() method는
static variable인 totalCars를 프린트하는 static method입니다.
이 method 내에는 instance variable은 사용되지 않았고
오직 static variable인 totalCars만 사용되었으므로
이 method를 모든 Car object들이 공유할 수 있게 됩니다.
이러한 method를 static method라 하며
역시 static method의 정보도 Method area의 class info에만
저장되고 있습니다.
static member에 대한 것은 5장에서 더 자세히 공부하도록 하겠습니다.

Example: Car Rental System (2/4)

```java
public class Customer {
    // Fields of a Customer class
    String name;         // Instance Variable
    int customerID;      // Instance Variable

    // Method to display customer details
    void displayCustomerDetails() {
        System.out.println("Customer Name: " + name + ",
                           Customer ID: " + customerID);
    }

    // Method to rent a car
    void rentCar(Car car) {
        System.out.println(name + " rented a car: " + car.model +
                           " with license plate: " + car.licensePlate);
    }
}
```


페이지 10

class Customer는 이 업체에서 자동차를 렌탈하는
고객 정보를 담고 있습니다.
고객의 이름과 고객번호를 나타내는 instance variable들이 있고
displayCustomerDetails() 는
Customer의 정보를 보여주는 method입니다.
rentCar method는
Car class의 parameter인 car를 받아들여
rentCar method를 call하는 customer object가
car를 rent했다는 메시지를 보여주게 됩니다.
그러나 이 프로그램에는 어떤 customer가 어떤 car를 rent했는지에 대한 것을
기록해 두지는 않고 있습니다.

Example: Car Rental System (3/4)

```
public class CarRentalSystem {
    public static void main(String[] args) {
        // Creating Car objects
        Car car1 = new Car();
        car1.model = “Kia K9";
        car1.licensePlate = "ABC123";
        Car.totalCars++;

        Car car2 = new Car();
        car2.model = “Genesis G80";
        car2.licensePlate = "XYZ789";
        Car.totalCars++;

        // Displaying car details
        car1.displayCarDetails();
        car2.displayCarDetails();

        // Displaying total number of cars
        Car.displayTotalCars();
```

```
OUTPUT:
Model: Kia K9, License Plate: ABC123
Model: Genesis G80, License Plate: XYZ789
Total Cars: 2
```


페이지 11

CarRentalSystem class는 Car와 Customer object들을 생성하고
그 정보들을 보여주며
customer에게 car를 rent해 주는 메시지를 보여줍니다.
먼저 car1이라는 한 대의 Car object를 생성합니다.
아까 보았던 Car.totalCars를 하나 증가시켜 주어서
현재 Car object의 갯수를 유지합니다.
한 가지 유의할 점은 totalCars라는 static variable을 access하는데
특정 object인 car1같은 것을 이용하지 않고
Car라는 Class 이름에 dot operator를 써서
대문자 Car dot totalCars로 access할 수 있다는 것입니다.
이것은 totalCars가 static variable이라서
Car class에 오직 하나만 존재하기 때문에 가능한 것입니다.
물론 car1.totalCars로 access해도 됩니다.
같은 방법으로 car2라는 Car object를 하나 더 생성합니다.
그리고 각 Car object의 displayCarDetails() method를 call하여
자동차의 정보를 보여줍니다.
그리고 현재의 totalCars value도 보여줍니다.

Example: Car Rental System (4/4)

```
        // Creating Customer objects
        Customer customer1 = new Customer();
        customer1.name = "John Doe";
        customer1.customerID = 101;

        Customer customer2 = new Customer();
        customer2.name = "Jane Smith";
        customer2.customerID = 102;

        // Displaying customer details
        customer1.displayCustomerDetails();
        customer2.displayCustomerDetails();

        // Customers renting cars
        customer1.rentCar(car1);
        customer2.rentCar(car2);

        // Display company name
        System.out.println("Company: " + Car.COMPANY_NAME);
    }
}
```

```
OUTPUT:
Customer Name: John Doe, Customer ID: 101
Customer Name: Jane Smith, Customer ID: 102
John Doe rented a car: Kia K9 with license plate: ABC123
Jane Smith rented a car: Genesis G80 with license plate: XYZ789
Company: SuperCar Rentals
```


페이지 12

여기서는 두 명의 Customer object를 생성합니다.
각각의 이름과 customer ID를 assign합니다.
그리고 displayCustomerDetails() method를 이용하여
Customer의 정보를 보여줍니다.
그리고 두 명의 customer들에게 car를 rent한다는
message를 보여줍니다.
마지막으로 이 Car rental 업체의 이름을 보여줍니다.

Multiple Classes in Multiple Java Files

Class1.java (public access)

```java
public class Class1
{
    int a;
    int add(int b) { return a + b; }
}
```

Class2.java (package (default) access)

```java
class Class2
{
    int a;
    int sub(int b) { return a - b; }
}
```

AATest.java

```java
public class AATest {
    public static void main(String[] args) {
        Class1 p = new Class1();
        Class2 q = new Class2();
        p.a = 3;
        q.a = 5;
        System.out.println(p.add(1) + " " + q.sub(1));
    }
}
```

```
OUTPUT:
4 4
```


페이지 13

이 슬라이드에서는 여러 개의 Java source file이
하나의 프로그램을 이루는 것을 보여주고 있습니다.
한 source file에는 하나의 class만 담겨 있습니다.
class 이름과 source file의 이름은 같아야 합니다.
Class1은 public access 권한이고
Class2는 default, 즉, package access 권한입니다.
source file들이 모두 같은 폴더에 있기만 하면
public과 package 권한은 차이가 없습니다.
access modifier에 대해서는 5장에서 더 자세히 공부할 것입니다.
AATest class는 main method를 가지고 있습니다.
그래서 이 프로그램을 compile하게 되면
AATest.class 파일이라는 실행 파일이 나오게 되는 거죠.
여기서는 Class1과 Class2의 object들을 생성하고
각 object들의 instance variable a에 3과 5 value를 assign하고
3에 1 더한 값과 5에서 1 뺀 값인 4, 4를 보여주고 있습니다.
main method를 가지고 있는 class인 AATest인
public 권한을 가져야 합니다.
일반적으로 Java 코딩할 때에는
이와 같이 한 소스파일에 하나의 class만 있는 것이 일반적입니다.
그렇게 하면, 소스 파일 이름이 곧 class 이름을 뜻하기 때문에
파일의 내용을 쉽게 짐작할 수 있습니다.
그러나 우리 auto-judge server에서는
하나의 source file만을 submit해야 하기 때문에
모든 class들을 하나의 source file에 작성해야 합니다.
이 경우에 대해서는 다음 슬라이드에서 살펴 보도록 하겠습니다.

Multiple Classes in a Single Java File

AAATest.java

```java
public class AAATest {
    public static void main(String[] args) {
        Class3 p = new Class3();
        Class4 q = new Class4();
        p.a = 3;
        q.a = 5;
        System.out.println(p.add(1) + " " + q.sub(1));
    }
}
class Class3
{
    int a;
    int add(int b) { int c = 3;  return a + b + c; }
}
class Class4
{
    int a;
    int sub(int b) { return a - b; }
}
```

There should be **only one public class** in a single java file:
- having main method
- the public class' name = source file name

```
OUTPUT: 7 4
```


페이지 14

이 슬라이드에서는 하나의 Java 소스 파일에
여러 개의 class를 넣어 프로그램하는 것을 보여주고 있습니다.
이 경우에는 main을 가지고 있는 class만 public 권한을 가져야 합니다.
여기서는 AAATest class가 public입니다.
또, 그 public class의 이름이 source file의 이름이 되어야 합니다.
나머지 다른 class들은 모두 package 권한을 가지게 하면 됩니다.
여기서는 Class3과 Class4가 그렇습니다.

Local Variables

- Local variable
  - A variable declared **within a method** definition
  - All method **parameters** are local variables
- ex)
  - AAATest.main's local: args, p, q
  - Class3.add's local: b, c
  - Class4.sub's local: b
- No global variable in Java

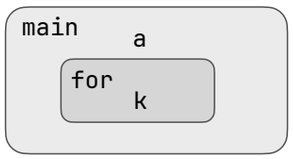
```java
public class AAATest {
    int k; // instance variable
    public static void main(String[] args) {
        Class3 p = new Class3();  // local
        Class4 q = new Class4();  // local
        p.a = 3;
        q.a = 5;
        System.out.println(p.add(1) + " " + q.sub(1));
    }
}
class Class3
{
    int a; // instance variable
    int add(int b) { int c = 3; return a + b + c; }
}
class Class4
{
    int a; // instance variable
    int sub(int b) { return a - b; }
}
```


페이지 15

Local variable은 어떤 method 안에 선언되어 사용되는
variable을 말합니다.
또, method의 parameter들은
모두 local variable로 간주됩니다.
예를 들면 args, p, q는 main의 local variable들입니다.
b와 c는 Class3의 add method의 local 입니다.
또다른 b는 Class4의 sub method의 local 입니다.
Java에는 global variable이 없습니다.
모든 class들은 적절한 parameter와 method call 등으로
다른 class와 인터랙션해야 합니다.
가끔 이런 엄격한 규칙이 프로그래밍을 어렵게 만든다고 느껴질 수도 있지만
global variable이 발생시킬 수 있는
error의 가능성을 방지할 수 있다는 장점이 있습니다.

Scope of Local Variables

```java
public class ATest {
    public static void main(String[] args) {
        int a = 5;                    // a: local in main
        for (int k = 0; k < 5; k++) { // k: local in for
            int a = k + 1;            // Error! a already declared
            System.out.println(k);    // OK
        }
        System.out.println(k);        // Error! no k in main
    }
}
```



페이지 16

a는 ATest의 main에 local variable이므로
main method 내의 for문 등 다른 block에서도
모두 access가 가능합니다.
k는 for문 block의 local variable이므로
for문 밖에서는 보이지 않습니다.
그리고 for문 안에서도 main의 local인 a가 보이기 때문에
for문 block 내에 또다른 a를 선언하는 것은 중복이기 때문에
선언하는 순간 compile error가 발생됩니다.
한편, for문을 벗어나면 k는 소멸하고 더 이상 보이지 않기 때문에
k를 access하는 것은 compile error 입니다.
벤 다이어그램은 local variable들의 scope를 보여주고 있습니다.

Other Methods in Class Containing main()

```java
public class ATest2 {

    public static void main(String[] args) {
        int a = 1, b = 2, c = 3;
        double result = myMethod(a, b, c);
        System.out.println("OUTPUT: " + result); // 6.0
    }

    // Any other methods called from main must be static,
    // because the main method is static

    static double myMethod(int a, int b, double c) {
        return a + b + c;
    }
}
```


페이지 17

이 슬라이드에는 main method 이외에
같은 class 내에 다른 method들을 더 정의하여 사용하는 경우를
보여주고 있습니다.
결국 main method 내에서 다른 method를 call 하는 경우이며
이 예에서는 MyMethod를 a, b, c의 합을 구하기 위해
main 안에서 call 하고 있습니다.
그런데 main이 static method이기 때문에
main 안에서는 static method나 static variable만 사용할 수 있습니다.
즉, myMethod도 static method이어야만 한다는 것입니다.

Actual and Formal Parameters

```java
public class ATest2 {

    public static void main(String[] args) {
        int a = 1, b = 2, c = 3;
        double result = myMethod(a,b,c);  // actual parameters: a, b, c
    }

    // formal parameters: d, e, f
    static double myMethod(int d, int e, double f) {
        return d + e + f;
    }
}
```

People often use the terms **parameter** and **argument** interchangeably

페이지 18

Actual parameter는 method를 call할 때
pass하는 parameter를 지칭하는 용어입니다.
이 예에서는 result = myMethod(a,b,c) 에서
a, b, c가 actual parameter입니다.
반면에 myMethod(int d, int e, int f)의 definition에서
값을 전달받는 parameter인 d, e, f는
formal parameter 입니다.
actual parameter라는 이름은 실제 pass되는 값을 지칭하는 것이고
formal parameter라는 이름은 pass되어 method내에서 사용되는
형식적인 이름이라는 뜻으로 해석하면 되겠습니다.
한편, parameter와 argument라는 용어는 미묘한 차이가 있지만
실제로는 거의 같은 뜻의 용어로 사용되고 있습니다.

Automatic Type Conversion: Primitive Type Parameters

```java
public class ATest2 {

    public static void main(String[] args) {
        int a=1,b=2,c=3;
        double result = myMethod(a,b,c);  // c is auto-converted to double
    }

    // auto conversion rule
    // byte → short → int → long → float → double
    // char → int → long → float → double

    static double myMethod(int d, int e, double f) {
       return d + e + f;  // d, e automatically converted to double
    }
}
```


페이지 19

이 예제에서는 actual parameter인 c는 int type인데
그에 대응하는 formal parameter f는 double인 경우를 보여줍니다.
이 때 c의 int값이 자동적으로 double type으로 변환 됩니다.
이러한 자동 변환이 가능한 것은
double type이 int type보다 표현 범위가 넓기 때문입니다.
이러한 자동 변환을 위해 표현범위를 살펴보면
byte가 1 byte char가 unsigned 2 bytes로 가장 작고
short, int, long, float, double의 순서로 표현범위가 넓어집니다.

The Reference `this` (1/2)

```
class SomeClass {
    public void referenceThis() {
        System.out.println(this);
    }
}
```

- "this" is the reference of an object

```
public class ThisRefTest {
    public static void main(String[] args) {
        SomeClass o1 = new SomeClass();
        SomeClass o2 = new SomeClass();
        SomeClass o3 = o1;
        System.out.println("o1: " + o1); // o1: SomeClass@6b95977
        System.out.print("o1's this: "); // o1's this:
        o1.referenceThis();              // SomeClass@6b95977
        System.out.println("o2: " + o2); // o2: SomeClass@8bcc55f
        System.out.print("o2's this: "); // o2's this:
        o2.referenceThis();              // SomeClass@8bcc55f
    }
}
```


페이지 20

이 슬라이드에서는 특수한 reference value를 가진
"this" 에 대해 살펴봅니다.
SomeClass의 referenceThis method는
this 값을 그대로 프린트하는 method입니다.
"this" 는 현재 이 class에서 만들어진 object의 reference,
즉, object의 주소를 말합니다.

ThisRefTest class의 main에서 "this" reference가
어떤 value를 가지는지 직접 프린트해 보도록 하겠습니다.
먼저 o1, o2는 SomeClass의 object로 생성되었습니다.
o3는 o1을 그대로 assign했기 때문에 o1과 같은 reference를 가집니다.
먼저 o1을 프린트 해 보면 "SomeClass@6b95977" 이라고 나옵니다.
여기서 SomeClass는 o1이 어떤 class인지를 나타내며
6b95977이 o1의 reference 입니다.
그 아래에서 o1 dot referenceThis() method를 call하여
this를 print해 보았습니다.
this는 정확히 바로 위의 o1의 reference와 같은 value를 가지고 있습니다.
마찬가지로 o2와 o2의 this 는
정확히 같은 reference value를 가지고 있음을 알 수 있습니다.

The Reference `this` (2/2)

```
class MyDate {
    int year;
    String month;
    int day;
    void setMyDate(int newYear, String month, int day) {
        year = newYear;
        this.month = month;
        this.day = day;
    }
}
```

- In method setMyDate
  - this.month, this.day: instance variable of the class "MyDate"
  - month, day: formal parameter (local variable) of the method "setMyDate"

페이지 21

this reference는 instance variable을 지칭할 때
편리하게 사용할 수 있습니다.
setMyDate method의 formal parameter 세개 중에
month와 day는 instance variable들과 이름이 같습니다.
이 formal parameter들의 값은
instance variable들로 assign되게 됩니다.
따라서 formal parameter의 이름들을
instance variable들과 다르게 하면
예를 들면 newMonth, newDay 같이 하면
instance variable들과 구분할 수 있지만
의미상 통일성을 위해서는
instance variable의 이름과 같게 하는 것이 더 편리합니다.
그러나 assignment의 좌변과 우변을 같은 이름으로 할 수 없기 때문에
다른 방법이 있어야 하는데
이런 경우, reference this를 사용하면 이 문제를 쉽게 해결할 수 있습니다.
reference this는 object 자신의 reference를 나타내므로,
this dot month와 this dot day로 하면
이것들은 instance variable을 의미합니다.
그리고 month와 day를 그냥 쓰면 그것들은
formal parameter들을 의미합니다.

‘equals’ method

```
class SomeClass { }
```

```
public class SomeClassDemo {
    public static void main(String[] args) {
        SomeClass o1 = new SomeClass();
        SomeClass o2 = new SomeClass();
        SomeClass o3 = o1;

        System.out.println(o1.equals(o2)); // false
        System.out.println(o2.equals(o3)); // false
        System.out.println(o3.equals(o1)); // true
    }
}
```

```
class SomeClass {
    public boolean equals(SomeClass other) {
        return this == other;
    }
}
```


페이지 22

SomeClass가 완전히 비어있어서 아무런 method도 가지지 않는 상태일때
이 SomeClass의 object 두 개 o1, o2를 생성하였고
o3는 o1을 그대로 assign해서 o1과 같은 object를 가리키게 하였습니다.
이 때 o1.equals(o2)를 실행했더니 false가 나왔습니다.
o2.equals(o3)도 false가 나왔고
o3.equals(o1)은 true가 나왔습니다.
SomeClass에 equals method가 정의되지 않았는데
compile error가 나지 않습니다.
이것은 SomeClass에 우리가 명시적으로 equals method를 정의하지 않아도
우리 눈에는 보이지 않지만 equals method가 이미 존재한다는 뜻일 것입니다.
그런데 세 번의 print로부터 우리는 equals method가
equals를 call하는 object와 parameter로 주어지는 object의
reference를 비교한 결과를 boolean으로 return한다는 것을 예측할 수 있습니다.
즉, equals method의 code를 써 보면 이렇게 간단하게 구현되어 있을 것입니다.
사실 equals method는 모든 class에 보이지는 않지만
이미 존재하고 있습니다.
그 이유에 대해서 지금은 명확한 이유를 설명하기가 어렵습니다.
나중에 inheritance를 배울 때 정확한 이유를 공부하도록 하겠습니다.

Redefine ‘equals’

```
class SomeClass {
    String name;
    int x;
    public boolean equals(SomeClass other) {
        return name.equals(other.name) && x == other.x;
    }
}
```


페이지 23

그러나 equals method가 reference만을 비교하는 단순한 것이 아니라
두 object의 내용이 정말 같은지를 비교하려면
우리는 equals method를 다시 define하여야 합니다.
이 예에서 SomeClass에는 두 개의 instance variable들 name과 x가 있습니다.
equals를 다시 define했는데
name이 같고, x의 value가 같은지, 이 두 조건이 모두 만족해야만
equals가 true를 return하도록 다시 define 했습니다.
String type인 name이 other.name과 같은지를 알아보기 위해서
간단하게 String class의 equals method를 이용하는 방법을 사용하였습니다.
String class의 equals는 이미 두 개의 String 내용이 같을 때
true를 return하도록 잘 정의되어 있습니다.
이와 같이 우리가 정의하는 모든 class마다
equals method를 잘 redefine해 놓는다면
매우 편리하게 사용할 수 있을 것입니다.

‘toString’ method

- Similar to ‘equals’, the ‘toString’ method is already defined in every class.
- The purpose of ‘toString’ is to produce the contents of an object (mainly the values of instance variables) as a String.
- It's a good idea to redefine 'toString' properly as well.

```
class SomeClass {
    String name;
    int x;
    public String toString() {
        return "SomeClass name: " + name + " x: " + x;
    }
}
```


페이지 24

equals와 비슷하게 toString method도 모든 class에 이미 존재합니다.
toString의 목적은 object의 내용 (주로 instance variable들의 내용)을
String으로 만들어서 return하는 것입니다.
그래서 toString method를 class마다 제대로 redeine해 놓는 것은
좋은 아이디어 입니다.
이 예에서는 이 object가 일단 SomeClass의 object라는 것을 표시하였고
두 개의 instance variable들인 name과 x value를 더해
String을 완성하고 return하였습니다.

Example: Equals and ToString (1/2)

```
class Student {
    String name;
    int id;
    int age;

    public void setData(String name, int id, int age) {
        this.name = new String(name);
        this.id = id;
        this.age = age;
    }

    public boolean equals(Student other) {
        return name.equals(other.name) && id == other.id &&
                age == other.age;
    }

    public String toString() {
        return "STUDENT name(" + name + ") id(" + id + ") age(" + age + ")";
    }
}
```


페이지 25

이 예제는 equals와 toString을 실제로 사용하는 예를 보여주고 있습니다.
먼저 Student class는 name, id, age의 세 개의 instance variable을 가지고 있습니다.
setData는 parameter로 주어진 name, id, age를
세개의 instance variable에 assign하는 method 입니다.
equals method는 redefine 되어 있는데
name, id, age가 모두 같아야 true를 return하게 되어 있습니다.
toString method에서는
instance variable들의 value들을 잘 보여주도록 하였습니다.

Example: class EqualsToString (2/2)

```java
public class EqualsToString {

    public static void main(String[] args) {
        Student st1 = new Student();
        Student st2 = new Student();
        Student st3 = new Student();
        st1.setData("Tom", 3527832, 23);
        st2.setData("Jane", 3527214, 22);
        st3.setData("Tom", 3527832, 23);
        System.out.println("[st1] " + st1);
        System.out.println("[st2] " + st2);
        System.out.println("[st3] " + st3);
        System.out.println("[st1.equals(st2)?] " + st1.equals(st2));
        System.out.println("[st1.equals(st3)?] " + st1.equals(st3));
        System.out.println("[st2.equals(st3)?] " + st2.equals(st3));
    }
}
```

```
OUTPUT:
[st1] STUDENT name(Tom) id(3527832) age(23)
[st2] STUDENT name(Jane) id(3527214) age(22)
[st3] STUDENT name(Tom) id(3527832) age(23)
[st1.equals(st2)?] false
[st1.equals(st3)?] true
[st2.equals(st3)?] false
```


페이지 26

EqualsToString class에서는 먼저 세개의 Student object들인
st1, st2, st3를 생성하였고
name, id, age를 setData method를 이용하여
각 object들에 assign하였습니다.
println에 "[st1] " + st1 String을 프린트하도록 하였는데
이와 같이 object를 println에 parameter로 넣어주면
그 object의 toString method가 자동으로 실행되게 됩니다.
OUTPUT에서 toString이 return한 String이 프린트 되는 것을
확인할 수 있습니다.
st1과 st3의 내용이 같기 때문에
st1.equals(st3)는 true를 return하게 됩니다.