04_1 Class and Object

Object-Oriented Programming

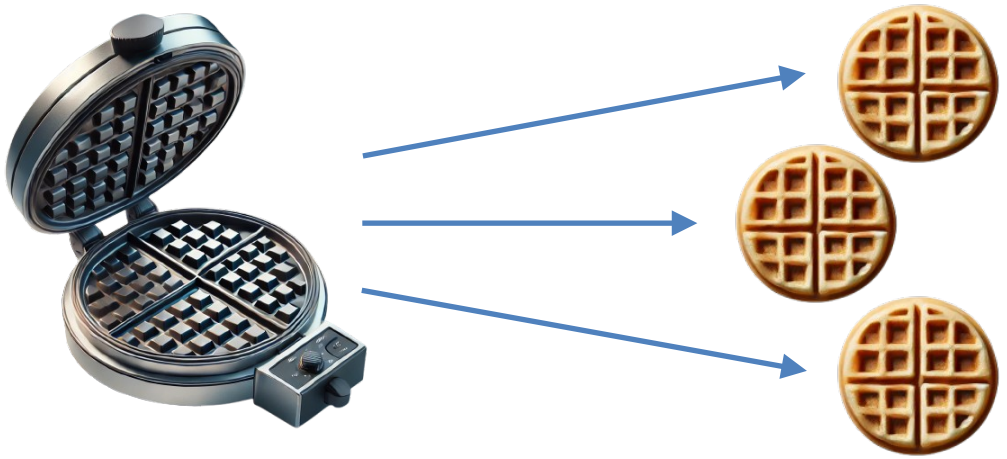
Class

- 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

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

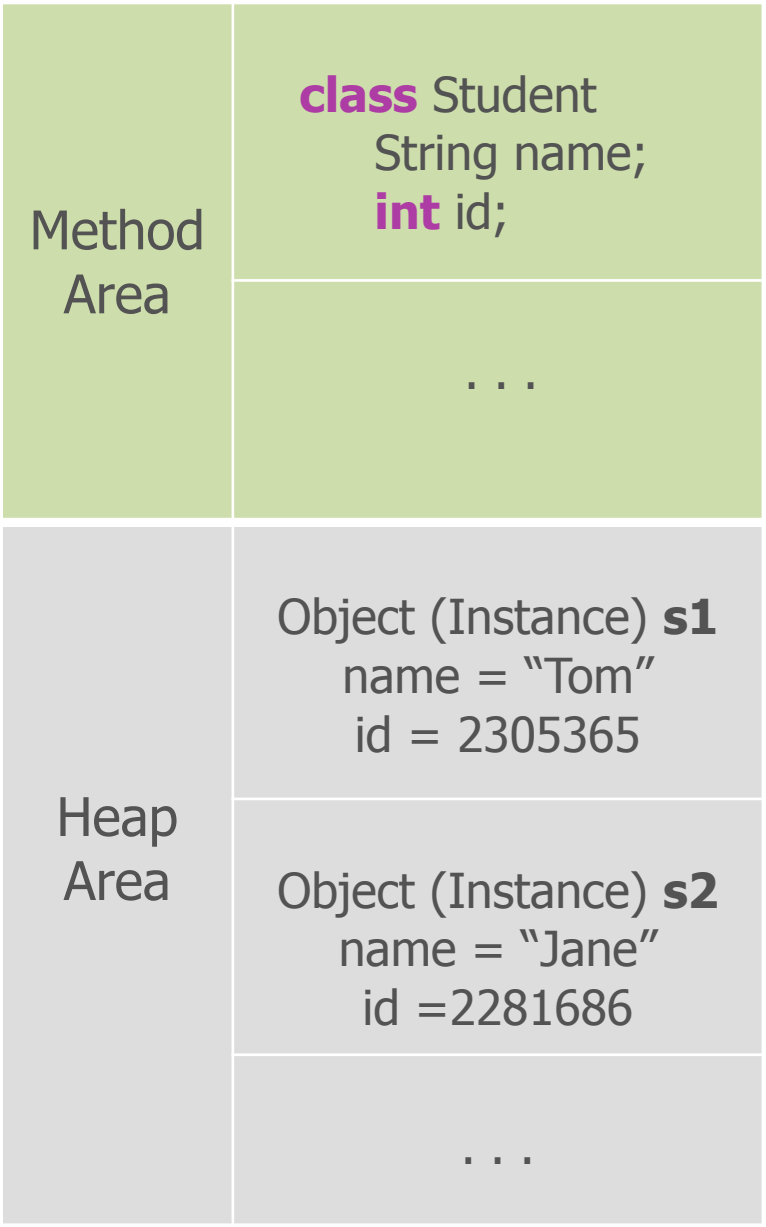
Creating Object

```
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

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;
```



Using Class Members (1/2)

SampleClass.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`

Using Class Members (2/2)

ClassDemo.java

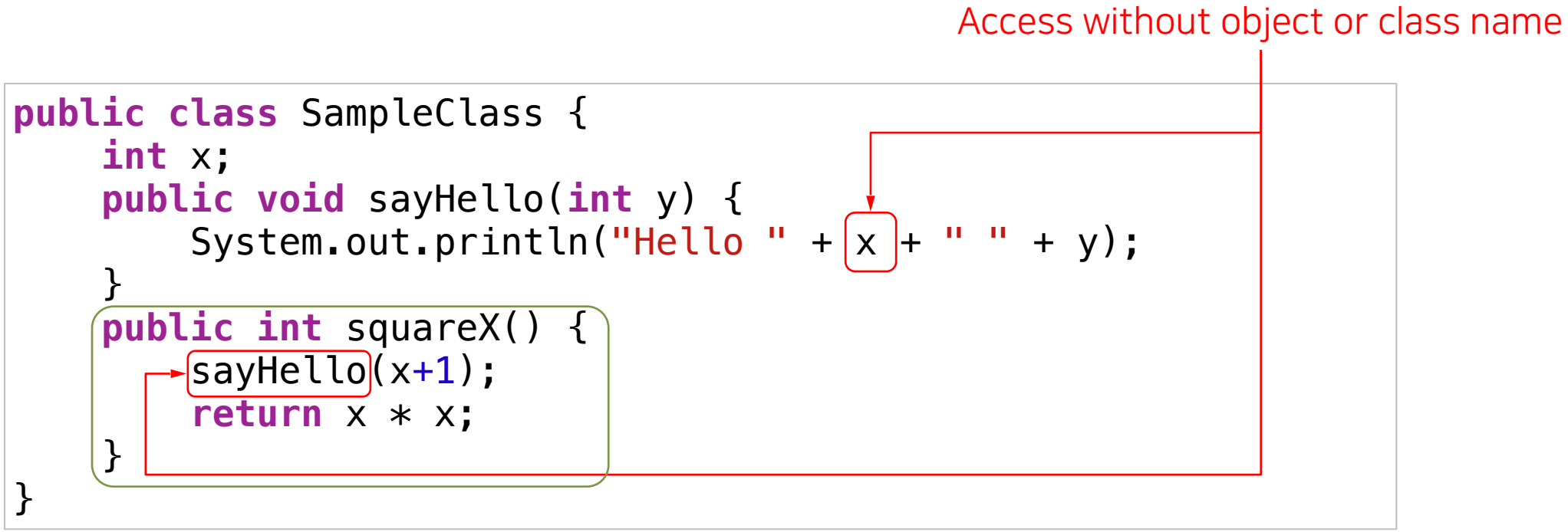
```
public class ClassDemo {
    public static void main(String[] args) {

        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()

Accessing Class Members Inside the Class



Example: Car Rental System (1/4)

```
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);
    }
}
```

Example: Car Rental System (2/4)

```
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);
    }
}
```

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
```

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
```

Multiple Classes in Multiple Java Files

Class1.java (public access)

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

Class2.java (package (default) access)

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

AATest.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
```

Multiple Classes in a Single Java File

AAATest.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
```

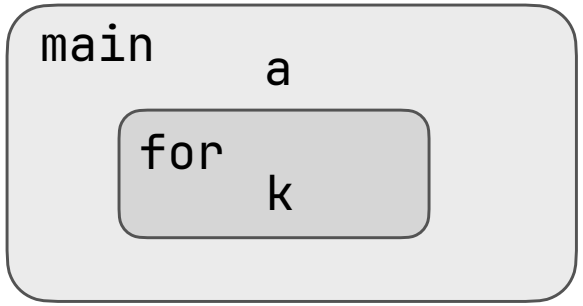
Local Variables

- 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

```
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; }
}
```

Scope of Local Variables

```
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
    }
}
```



Other Methods in Class Containing main()

```
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;
    }
}
```

Actual and Formal Parameters

```
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

Automatic Type Conversion: Primitive Type Parameters

```
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
    }
}
```

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
    }
}
```

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”

‘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;
    }
}
```

Redefine ‘equals’

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

‘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;
    }
}
```

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 + ")";
    }
}
```

Example: class EqualsToString (2/2)

```
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
```