Programming Language
The third day

Inheritance (1)

This document first introduces “method”, “access control” and “if statement”. And the technology of “inheritance” will be introduced.

1. Method

A Java method is a collection of statements that are grouped together to perform an operation.

1. 1 Creating Method

Method definition consists of a method header and a method body. The following is syntax.

```
modifier returnType nameOfMethod (Parameter List) {
    // method body
}
```

modifier: It defines the access type of the method and it is optional to use. The access type will be introduced in Chapter 2.

returnType: Data type which the method returns. The “void” keyword allows us to create methods which do not return a value.

nameOfMethod: the method name.

Parameter List: A comma-delimited list of input parameters, preceded by their data types. If there is no parameter, we must use empty parentheses.

method body: The method body defines what the method does with statements.

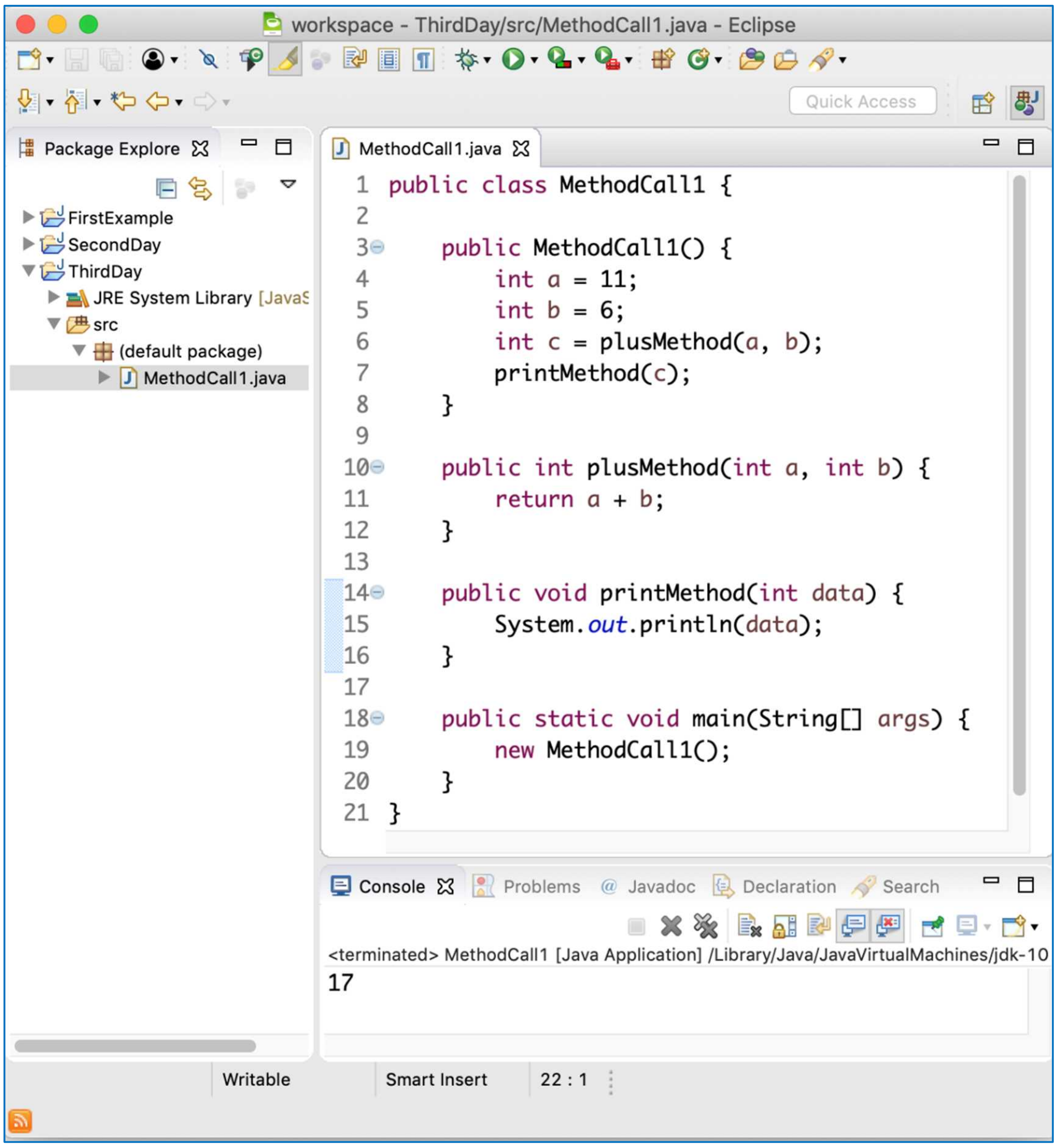
1. 2 Method Calling

For using a method, it should be called. There are two ways in which a method is called i.e., method returns a value or returning nothing (no return value).

The process of method calling is simple. When a program invokes a method, the program control gets transferred to the called method. This called method then returns control to the caller in two conditions:

- ✓ the return statement is executed.
- ✓ it reaches the method ending closing brace

The following is examples of two ways in which a method is called.



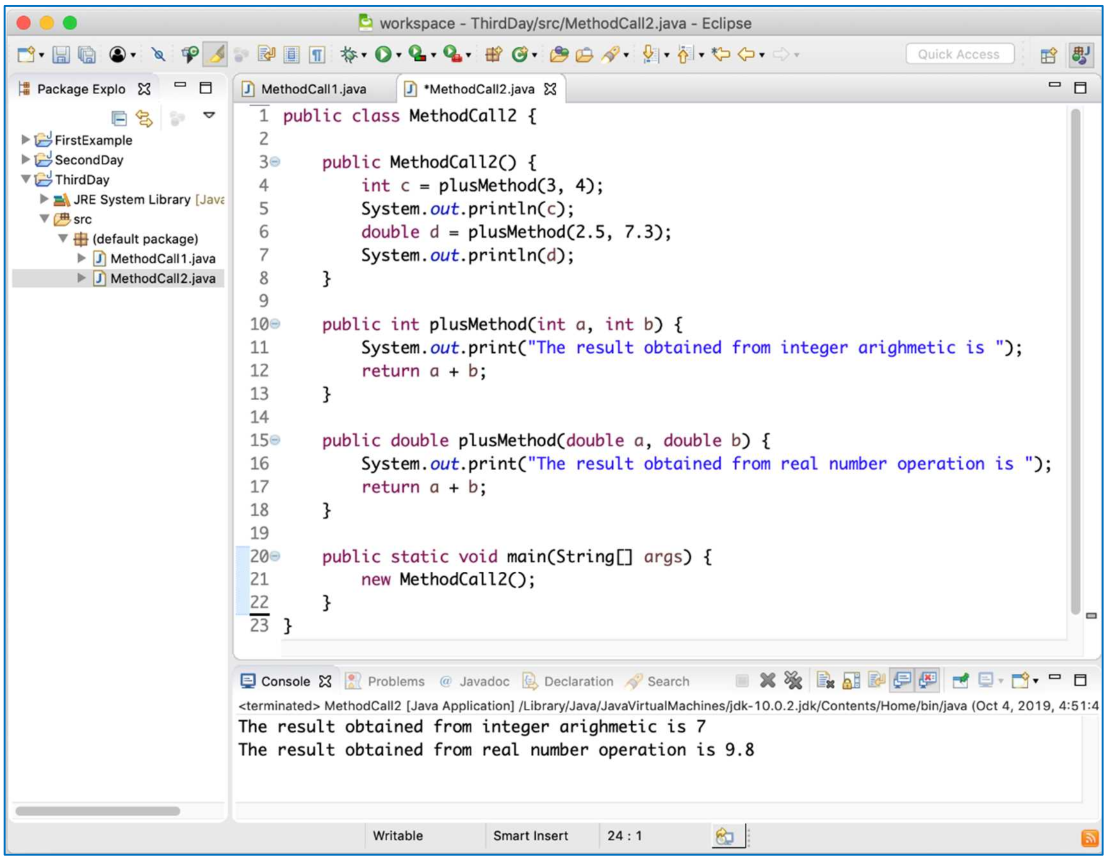
```
public class MethodCall1 {

    public MethodCall1() {
        int a = 11;
        int b = 6;
        int c = plusMethod(a, b);
        printMethod(c);
    }

    public int plusMethod(int a, int b) {
        return a + b;
    }

    public void printMethod(int data) {
        System.out.println(data);
    }

    public static void main(String[] args) {
        new MethodCall1();
    }
}
```

```
<terminated> MethodCall1 [Java Application] /Library/Java/JavaVirtualMachines/jdk-10
17
```

The method “plusMethod” takes two integers as arguments and returns the sum of them as a return value. The method “printMethod” gets a single integer and prints it to standard output. No value is returned.

1. 3 Method Overloading

When a class has two or more methods by the same name but different parameters, it is known as method overloading. It is different from overriding. The following is an example where different methods are called depending on the argument.



```java
public class MethodCall2 {

    public MethodCall2() {
        int c = plusMethod(3, 4);
        System.out.println(c);
        double d = plusMethod(2.5, 7.3);
        System.out.println(d);
    }

    public int plusMethod(int a, int b) {
        System.out.print("The result obtained from integer arighmetic is ");
        return a + b;
    }

    public double plusMethod(double a, double b) {
        System.out.print("The result obtained from real number operation is ");
        return a + b;
    }

    public static void main(String[] args) {
        new MethodCall2();
    }
}
```

```
<terminated> MethodCall2 [Java Application] /Library/Java/JavaVirtualMachines/jdk-10.0.2.jdk/Contents/Home/bin/java (Oct 4, 2019, 4:51:4
The result obtained from integer arighmetic is 7
The result obtained from real number operation is 9.8
```

2. Access Control

As the name suggests access modifiers in Java helps to restrict the scope of a class, constructor, variable, method or data member. There are four types of access modifiers available in Java as shown in Table 1.

A "Package" can be defined as a grouping of related types (classes, interface, etc.) providing access protection and namespace management. If a package is specified, the source file for the class must be stored in a folder with the same name as the specified package. If no package is specified, all class are classified as default packages.

Table 1. Access Modifiers in Java

Modifier	Same Class	Same Package	Different Package Subclass	World
public	Yes	Yes	Yes	Yes
protected	Yes	Yes	Yes	No
default (no modifier)	Yes	Yes	No	No
private	Yes	No	No	No

(A) public

The public access modifier is specified using the keyword “public”. The public access modifier has the widest scope among all other access modifiers. Classes, method or data members which are declared as public are accessible from every where in the program. There is no restriction on the scope of a public data members.

(B) protected

The protected access modifier is specified using the keyword “protected”. The method or data members declared as protected are accessible within same package or sub classes in different package.

(C) default

When no access modifier is specified for a class, method or data member, it is said to be having the default access modifier by default. The data members, class or methods which are not declared using any access modifiers are accessible only within the same package.

(D) private

The private access modifier is specified using the keyword “private”. The methods or data members declared as private are accessible only within the class in which they are declared.

3. Control flow Statement

When we need to execute a set of statements based on a condition then we need to use **control flow statements**. For example, if a number is greater than zero then we want to print “Positive Number” but if it is less than zero then we want to print “Negative Number”. In this case we have two print statements in the program, but only one print statement executes at a time based on the input value. We will see how to write such type of conditions in the java program using control statements.

There are four types of control statements that we can use in java program based on the requirement.

(A) If Statement

If statement consists a condition, followed by statement or a set of statements as shown below:

```
if(condition){
    statement
}
```

The statements are executed only when the *condition* is true. If the *condition* is false then the statements inside if statement body are completely ignored.

(B) Nested If Statement

When there is an if statement inside another, it is called the nested if statement. The structure of nested if statement looks like this:

```
if(condition_1){
    statement_1
    if(condition_2){
        statement_2
    }
}
```

Statement_1 would be executed if the *condition_1* is true. Statement_2 would be only executed if both the conditions (*condition_1* and *condition_2*) are true.

(C) If-else Statement

This is how an if-else statement looks:

```
if(condition){
    statement_1
} else {
    statement_2
}
```

The statements inside "if" (statement_1) would be executed if the *condition* is true, and the statements inside "else" (statement_2) would be executed if the *condition* is false.

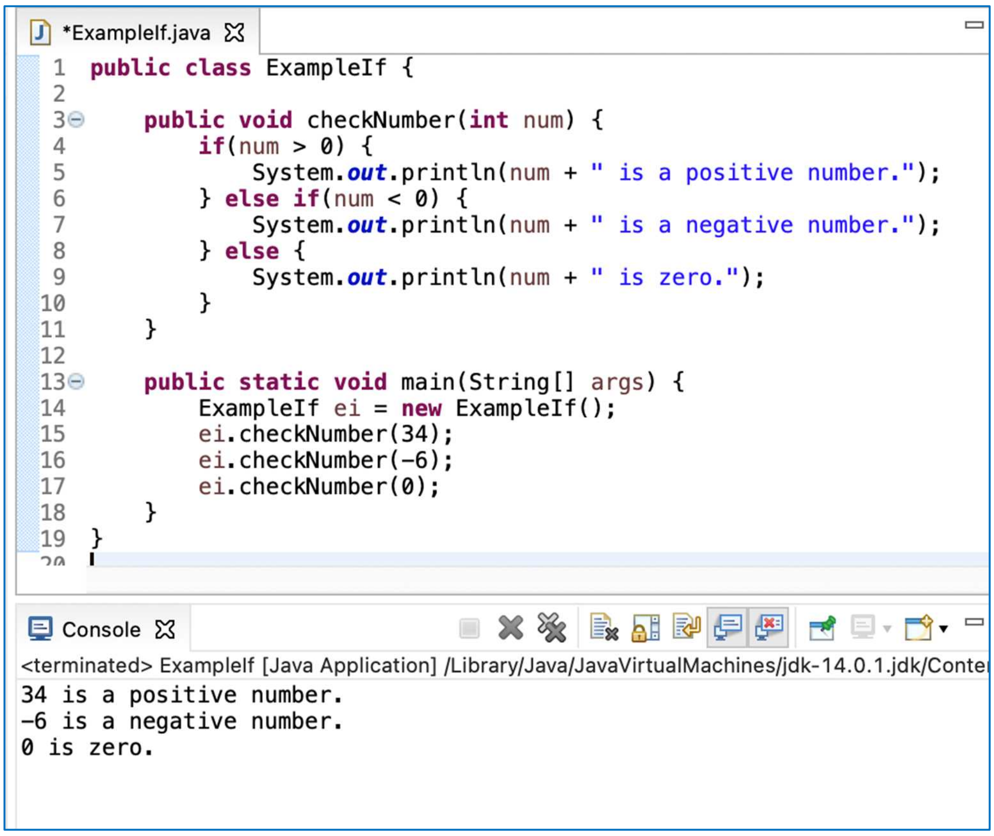
(D) If-else-if Statement

If-else-if statement is used when we need to check multiple conditions. We can have multiple "else if". This is how it looks:

```
if(condition_1) {
    statement_1
} else if(condition_2) {
    statement_2
} else if(condition_3) {
    statement_3
}
.
.
} else {
    statement_n
}
```

If *condition_1* is true, statement_1 would be executed. The statement_2 would be executed if *condition_1* is false and *condition_2* is true. The statement_3 would be executed if *condition_1* and *condition_2* are false and *condition_3* is true. If none of the condition is true, then statement_n would be executed.

An example using an if statement is shown below.



*ExampleIf.java

```java
public class ExampleIf {

    public void checkNumber(int num) {
        if(num > 0) {
            System.out.println(num + " is a positive number.");
        } else if(num < 0) {
            System.out.println(num + " is a negative number.");
        } else {
            System.out.println(num + " is zero.");
        }
    }

    public static void main(String[] args) {
        ExampleIf ei = new ExampleIf();
        ei.checkNumber(34);
        ei.checkNumber(-6);
        ei.checkNumber(0);
    }
}
```

Console

<terminated> ExampleIf [Java Application] /Library/Java/JavaVirtualMachines/jdk-14.0.1.jdk/Conte

```
34 is a positive number.
-6 is a negative number.
0 is zero.
```

The condition is described using the relational operators and the logical operators. The relational operators are introduced in Table 2. The logical operators are also introduced in Table 3.

Assume variable A holds 10, variable B holds 20, variable X holds true and variable Y holds false.

Table 2. Relational Operators

Operator	Description	Example
== (equal to)	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true
!= (not equal to)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true
> (greater than)	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true
< (less than)	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true
>= (greater than or equal to)	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true
<= (less than or equal to)	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true

Table 3. Logical Operators

Operator	Description	Example
&& (logical and)	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(X && Y) is false
|| (logical or)	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(X || Y) is true
! (logical not)	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(X && Y) is true

4. Inheritance

Inheritance is an OOPS concept in which one object acquires the properties (fields and methods) of the parent object. It's creating a parent-child relationship between two classes. It offers robust and natural mechanism for organizing and structure of any software. The class which inherits the properties of other is known as subclass (child class) and the class whose properties are inherited is known as superclass (parent class).

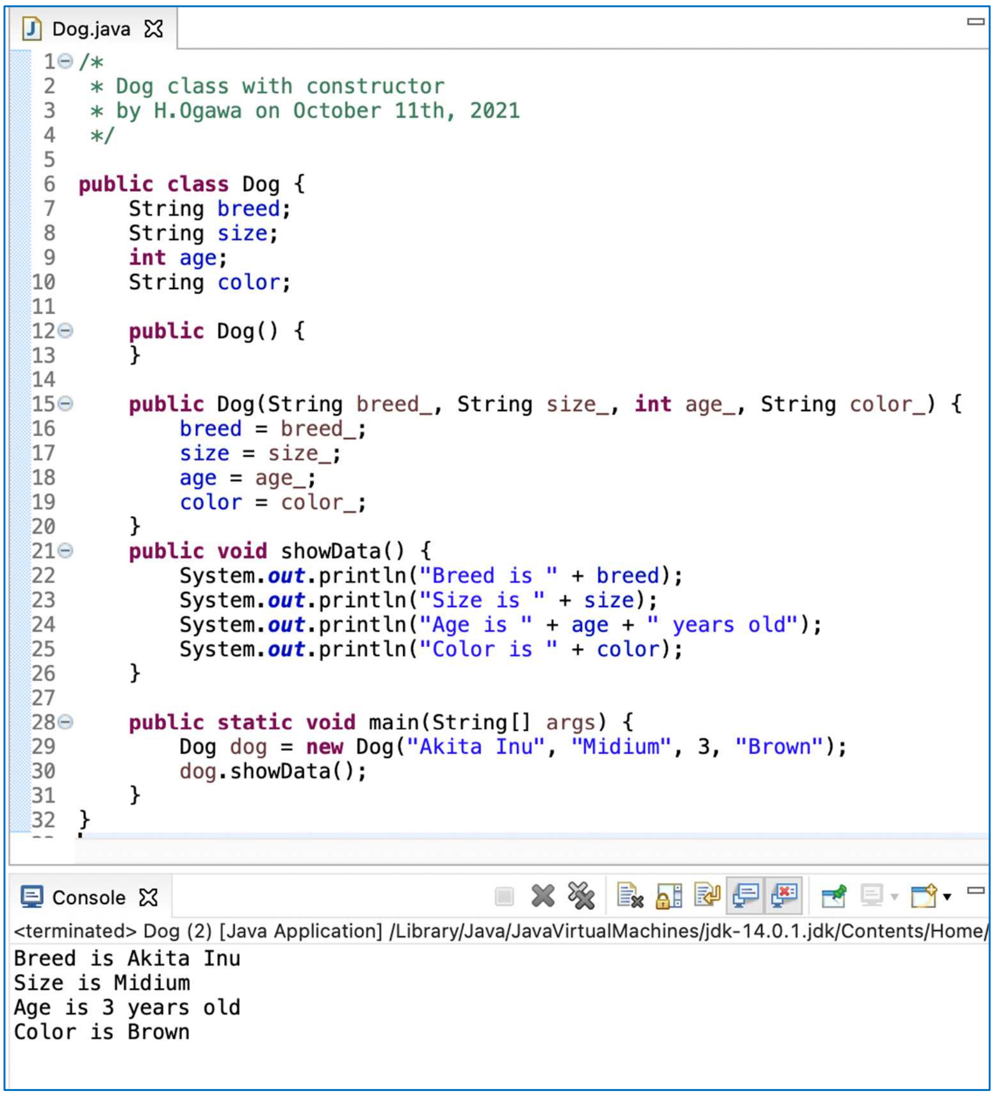
(A) Keyword extends

Keyword "extends" is used to inherit the properties of a class. Following is the syntax of extends keyword.

```
class SubClass extends SuperClass {
    ...
}
```

Since Akita Inu is a kind of dog, Akita_Inu class can inherit Dog class as well. To show various inheritance examples, no-argument constructor is added to the Dog class as shown in Figure 1. Figure 2 shows an example of Akita_Inu class which uses no argument constructor. When an instance of Akita_Inu class is created, a copy of the contents of the superclass (Dog class) is made within it. Variables and methods set in Dog class can be used in the same way as set in Akita_Inu class. So, we do not need to use "dog." to refer to instance variable and methods.

Figure 3 shows an example of Akita_Inu class which uses Akita_Inu constructor.



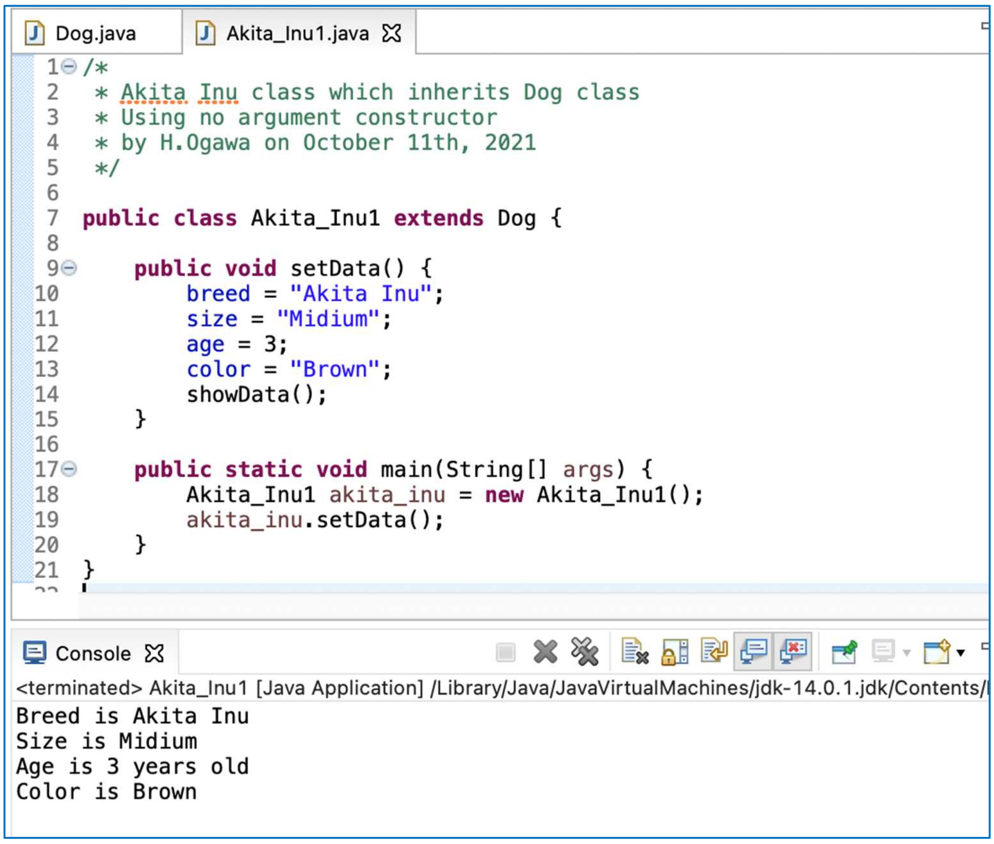
Dog.java

```java
/*
 * Dog class with constructor
 * by H.Ogawa on October 11th, 2021
 */

public class Dog {
    String breed;
    String size;
    int age;
    String color;

    public Dog() {
    }

    public Dog(String breed_, String size_, int age_, String color_) {
        breed = breed_;
        size = size_;
        age = age_;
        color = color_;
    }
    public void showData() {
        System.out.println("Breed is " + breed);
        System.out.println("Size is " + size);
        System.out.println("Age is " + age + " years old");
        System.out.println("Color is " + color);
    }

    public static void main(String[] args) {
        Dog dog = new Dog("Akita Inu", "Midium", 3, "Brown");
        dog.showData();
    }
}
```

Console

```
<terminated> Dog (2) [Java Application] /Library/Java/JavaVirtualMachines/jdk-14.0.1.jdk/Contents/Home/
Breed is Akita Inu
Size is Midium
Age is 3 years old
Color is Brown
```

Figure 1. Dog class with no-argument constructor.



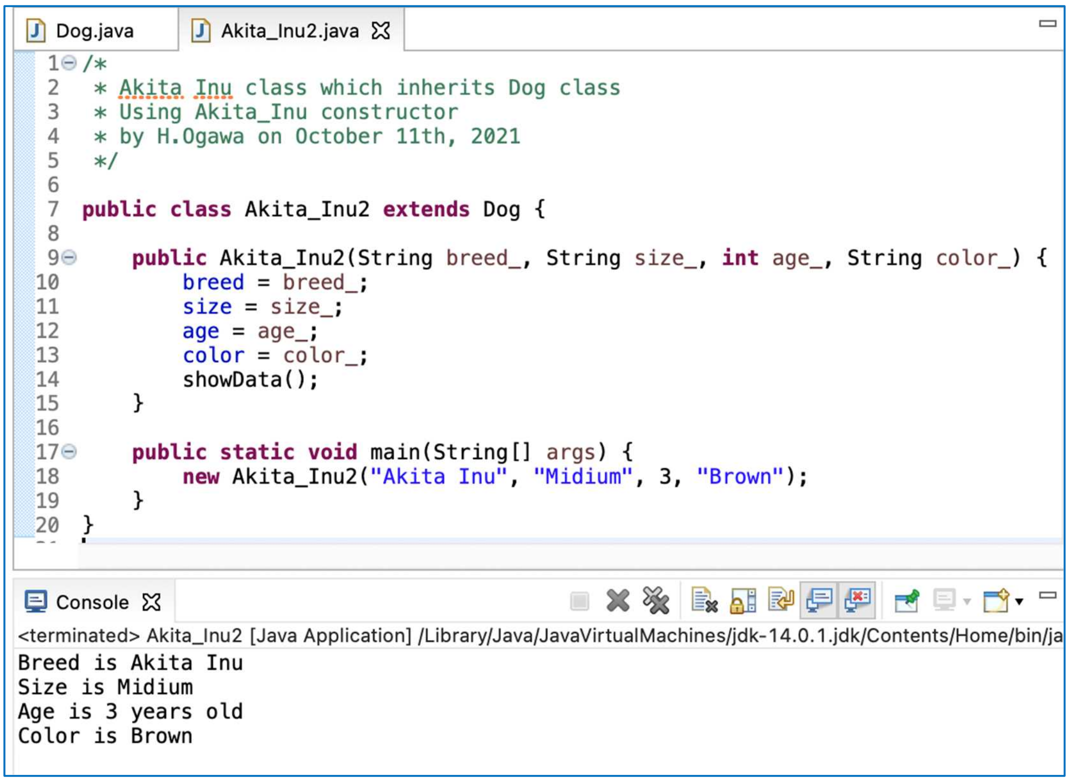
Dog.java Akita_Inu1.java

```java
/*
 * Akita Inu class which inherits Dog class
 * Using no argument constructor
 * by H.Ogawa on October 11th, 2021
 */

public class Akita_Inu1 extends Dog {

    public void setData() {
        breed = "Akita Inu";
        size = "Midium";
        age = 3;
        color = "Brown";
        showData();
    }

    public static void main(String[] args) {
        Akita_Inu1 akita_inu = new Akita_Inu1();
        akita_inu.setData();
    }
}
```

Console

```
<terminated> Akita_Inu1 [Java Application] /Library/Java/JavaVirtualMachines/jdk-14.0.1.jdk/Contents/
Breed is Akita Inu
Size is Midium
Age is 3 years old
Color is Brown
```

Figure 2. Akita_Inu class which uses no argument constructor.



Dog.java Akita_Inu2.java

```java
/*
 * Akita Inu class which inherits Dog class
 * Using Akita_Inu constructor
 * by H.Ogawa on October 11th, 2021
 */

public class Akita_Inu2 extends Dog {

    public Akita_Inu2(String breed_, String size_, int age_, String color_) {
        breed = breed_;
        size = size_;
        age = age_;
        color = color_;
        showData();
    }

    public static void main(String[] args) {
        new Akita_Inu2("Akita Inu", "Midium", 3, "Brown");
    }
}
```

Console

```
<terminated> Akita_Inu2 [Java Application] /Library/Java/JavaVirtualMachines/jdk-14.0.1.jdk/Contents/Home/bin/ja
Breed is Akita Inu
Size is Midium
Age is 3 years old
Color is Brown
```

Figure 3. An example using Akita_Inu constructor.

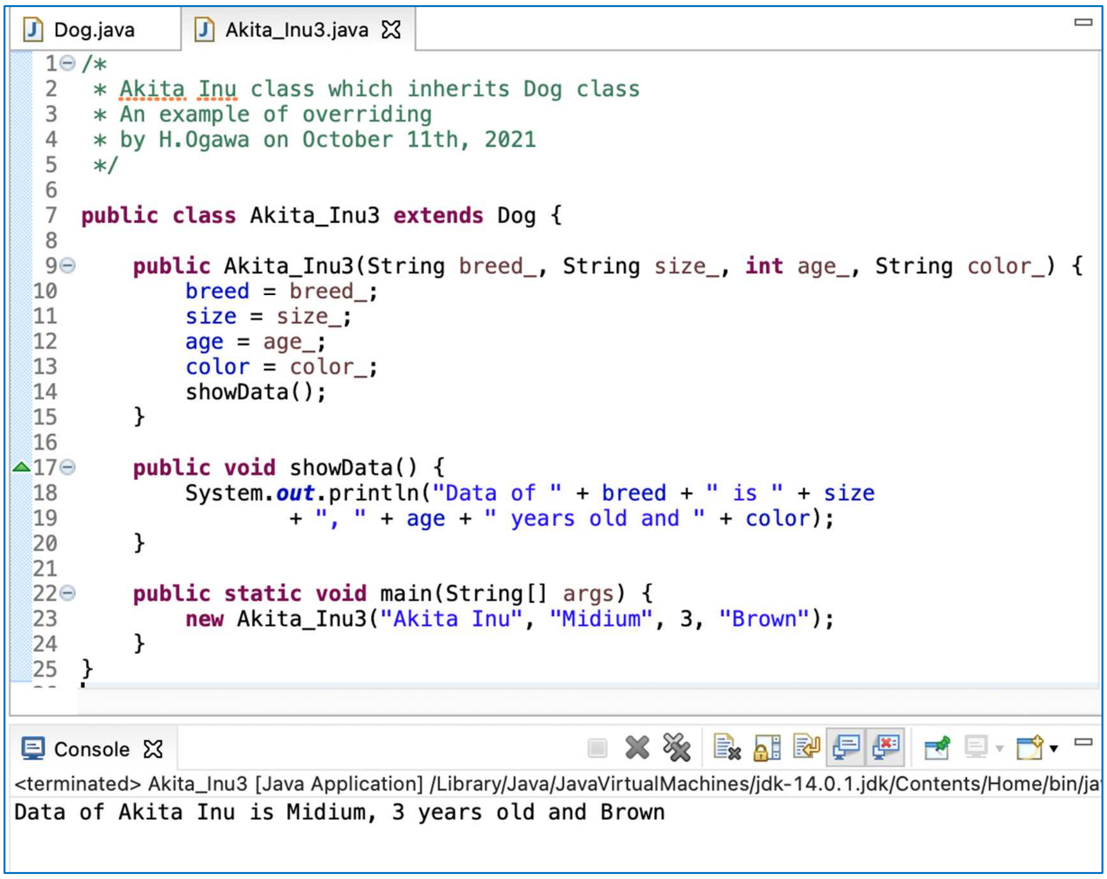
(B) Overriding

The benefit of overriding is an ability to define a behavior that's specific to the subclass type, which means a subclass can implement a parent class method based on its requirement.

In order to override, all of the following conditions must be satisfied.

- ✓ The argument list should be exactly the same as that of the overridden method.
- ✓ The return type should be the same or a subtype of the return type declared in the original overridden method in the superclass.

To change the information to display on a single line, just add the new showData method to the Akita_Inu class and override it, as shown in Figure 4. The method "showData" of Akita_Inu is used in this example.



```java
/*
 * Akita Inu class which inherits Dog class
 * An example of overriding
 * by H.Ogawa on October 11th, 2021
 */

public class Akita_Inu3 extends Dog {

    public Akita_Inu3(String breed_, String size_, int age_, String color_) {
        breed = breed_;
        size = size_;
        age = age_;
        color = color_;
        showData();
    }

    public void showData() {
        System.out.println("Data of " + breed + " is " + size
                + ", " + age + " years old and " + color);
    }

    public static void main(String[] args) {
        new Akita_Inu3("Akita Inu", "Midium", 3, "Brown");
    }
}
```

```
<terminated> Akita_Inu3 [Java Application] /Library/Java/JavaVirtualMachines/jdk-14.0.1.jdk/Contents/Home/bin/ja
Data of Akita Inu is Midium, 3 years old and Brown
```

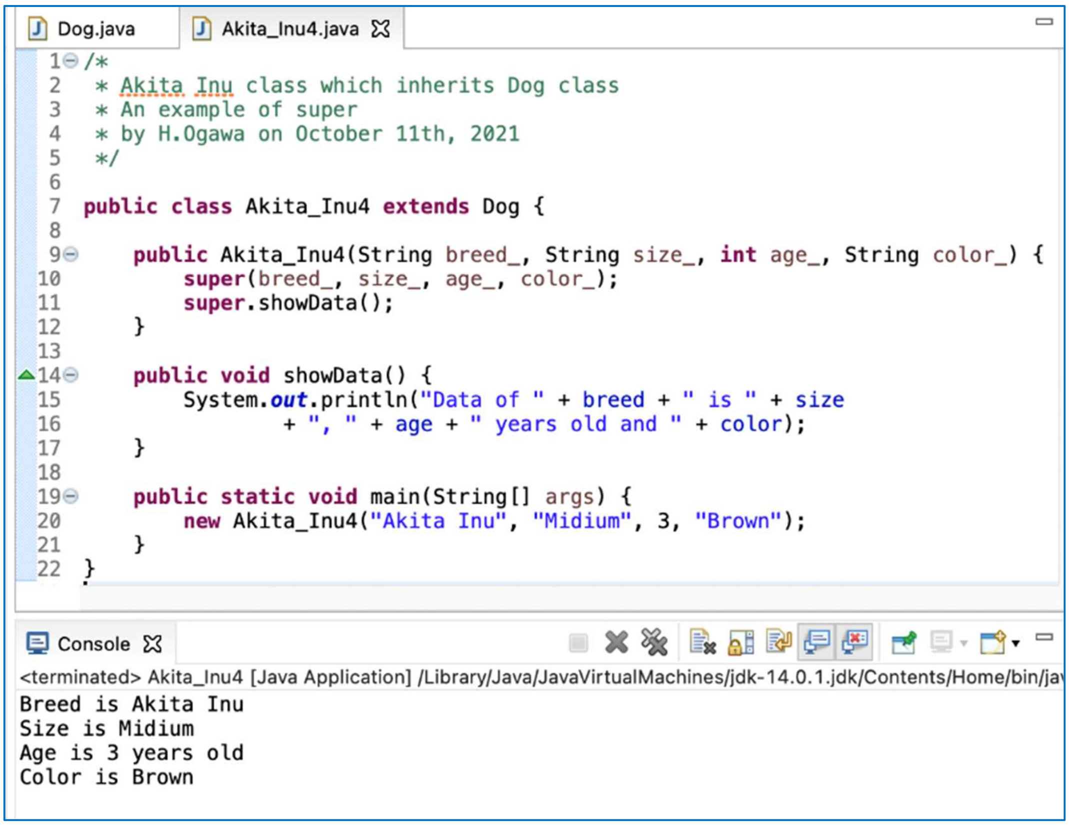
Figure 4. An example of override

(C) Keyword super

In the example shown in Figure 4, “showData()” method of Akita_Inu class is overriding “showData()” method on Dog class. In this case, the information of Akina Inu is displaied on a single line. In such a situation, in order to display four lines using “showData()” method on Dog class, use “super” as shown in Figure 5.

In the following cases, “super” is used.

- ✓ To access the data members of superclass when both superclass and subclass have member with same name.
- ✓ To explicitly call the no-argument and parameterized constructor of superclass. (for example, line 9)
- ✓ To access the method of superclass when subclass has overridden that method. (for example, line 10)



Dog.java | Akita_Inu4.java

```java
 1 /*
 2  * Akita Inu class which inherits Dog class
 3  * An example of super
 4  * by H.Ogawa on October 11th, 2021
 5  */
 6
 7 public class Akita_Inu4 extends Dog {
 8
 9     public Akita_Inu4(String breed_, String size_, int age_, String color_) {
10         super(breed_, size_, age_, color_);
11         super.showData();
12     }
13
14     public void showData() {
15         System.out.println("Data of " + breed + " is " + size
16                 + ", " + age + " years old and " + color);
17     }
18
19     public static void main(String[] args) {
20         new Akita_Inu4("Akita Inu", "Midium", 3, "Brown");
21     }
22 }
```

Console

```
<terminated> Akita_Inu4 [Java Application] /Library/Java/JavaVirtualMachines/jdk-14.0.1.jdk/Contents/Home/bin/ja
Breed is Akita Inu
Size is Midium
Age is 3 years old
Color is Brown
```

Figure 5. An example of super