**Defining Methods**

A method definition comprises two components:

1. Header that includes modifier, type, identifier, or name of method and a list of parameters. The parameter list is placed in a pair of parentheses.

2. Body that is placed in braces ({}) and consists of declarations and executable statement and other expressions.

A method may or may not throw an exceptional condition. A definition of method that does not throw an exception is illustrated in the following statements.

Modifier return\_type method\_indentifier(type Parameter\_Name,…) {/\*Statements\*/}

A definition of method that throws an exception is as follows:

Modifier return\_type method\_indentifier(type Parameter\_Name,…) throws exception-list

{/\*Statements\*/}

**Modifiers for Method**

A close-up of a document

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A diagram of a program

Description automatically generated

illustrates the declaration and working with overloaded methods.

class Overload

{

void myMethod(){// method without a parameter

System.out.println(“No value to return”);

}

void myMethod(int n)// one int parameter method

{

System.out.println (“n = ” + n);

}

void myMethod(int n, int m)// two int parameters method

{

System.out.println (“n\*m = ” + n\*m);

}

void myMethod(int n, int m, int p)//3 int parameter method

{

System.out.println(“n\*m\*p = ” + n\*m\*p);

}

void myMethod (double x, double y)//two double parameters

{

System.out.println (“x/y = ” + x/y);

}

}

class Ex {// class with main method

public static void main (String args [])

{

Overload Object = new Overload ();

Object. myMethod ();

Object. myMethod (5);

Object. myMethod (21, 2);

Object. myMethod (5, 3, 2);

Object. myMethod (12.6, 2.0);

}

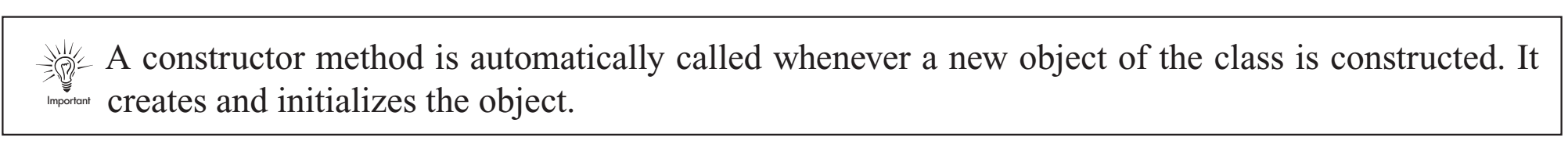
}

A text on a white background

Description automatically generated

**Overloaded Constructor Methods**

A constructor method has the same name as the name of class to which it belongs. It has no type and it does not return any value. It only initializes the object. The constructor method may also be overloaded by changing the number of default values. Therefore, constructors with different parameters may be declared; for the remaining parameters, it will pick up default values when these are not specified in the object definition.



class Farm

{

double length;

double width;

Farm (double l, double w)// Parametric Constructor

{

length = l; width = w;

}

Farm(Farm farm)// copy constructor

{

length = farm.length; width = farm.width;

}

Farm (double m)// Constructor with one parameter Length

{

length = m; width = 40;

}// Width has default value 40

Farm ()// Default constructor

{

length = 60;// Both Length and Width have default values

width = 30;

}

double area()

{

return length \* width;

}

}// end of class Farm

class FarmExec2 {// class with main method

public static void main (String args[])

{

Farm farm1 = new Farm (30, 10);

Farm farm2 = new Farm(farm1);

Farm farm3 = new Farm();

Farm farm4 = new Farm(60);

System.out.println (“Area of farm1 = ” + farm1.area());

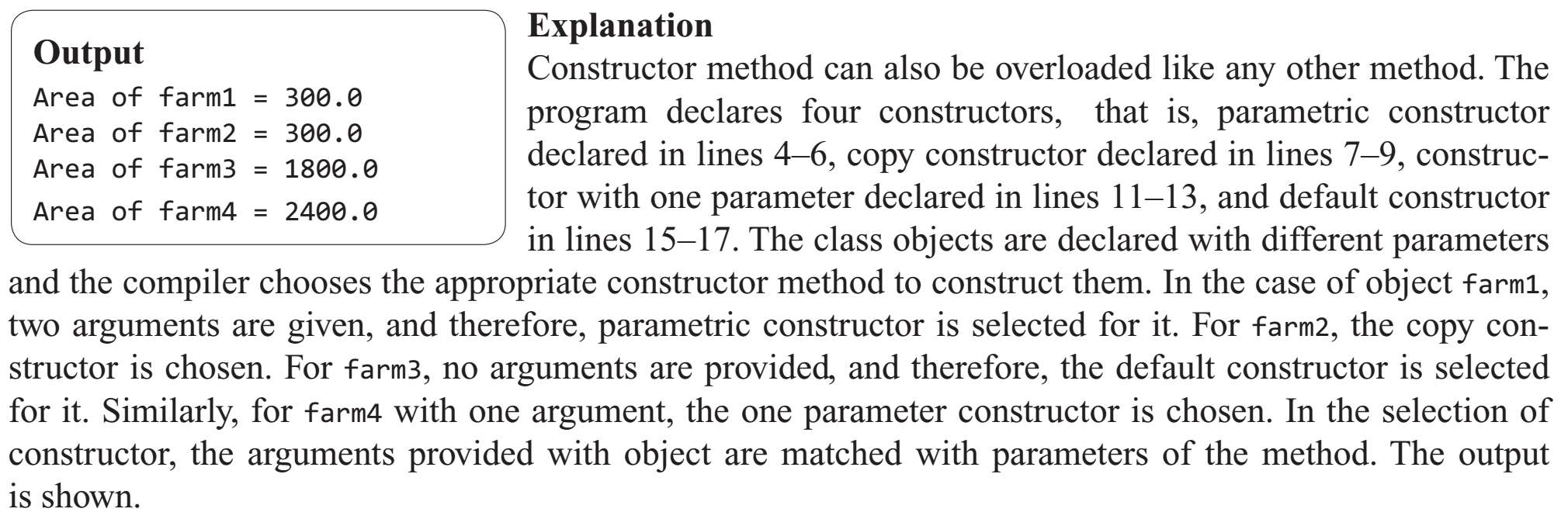
System.out.println (“Area of farm2 = ” + farm2.area());

System.out.println (“Area of farm3 = ” + farm3.area());

System.out.println (“Area of farm4 = ” + farm4.area());

}

}



**Class Objects as Parameters in Methods**

Similar to any other primitive parameter, an object of a class may also be a parameter of a method. The type of an object of a class is the name of its class. The operations that the class object may be subjected to are the ones derived from the class Object and those defined in its own class and the super class if it is the object of a derived class.

class Farm

{

double length;// declaration of variables

double width;

Farm (double l, double w)// method definition

{length = l; width = w;}

/\*Following is the method with parameter farm an object of Farm.\*/

void equals(Farm farm){// method definition

if (farm.length \* farm.width == length\*width)

{

System.out.println(“True”);

}

else

{

System.out.println(“False”);

}

}

}

class FarmExec4// class with main method

{

public static void main (String Str[])

{

Farm farm1 = new Farm (25, 10); //creating objects

Farm farm2 = new Farm(30, 20);

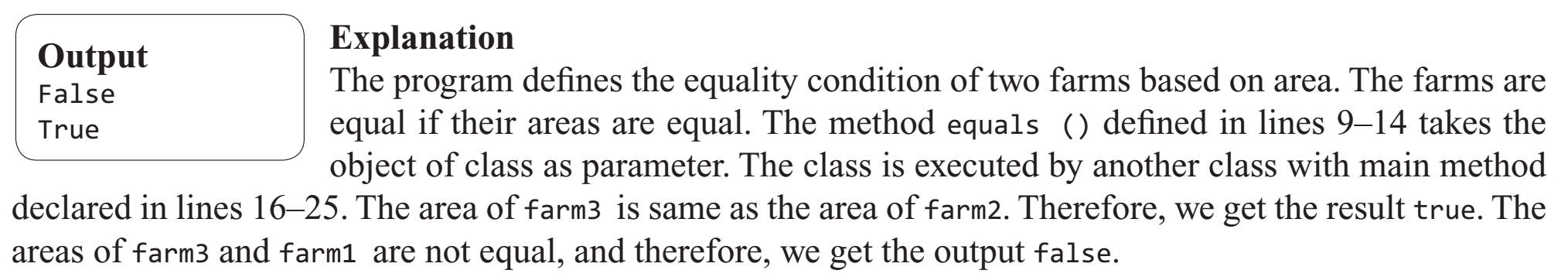
Farm farm3 = new Farm (15,40);

farm3. equals(farm1);// accessing methods

farm3. equals (farm2);

}

}



**Recursive Methods :**

Java recursion is a process where a function calls itself continuously. The corresponding function is known as the recursion function in Java.

By using the recursion strategy and algorithm, we can solve certain complex problems and make them more manageable and simpler.

Example :

**class** Factorial {

**// this is a recursive function**

**int** fact(**int** n) {

**int** result;// j a v a2s.c om

**if** (n == 1)

**return** 1;

result = fact(n - 1) \* n;

**return** result;

}

}

**public** **class** Main {

**public** **static** **void** main(String args[]) {

Factorial f = **new** Factorial();

System.out.println(**"Factorial of 5 is "** + f.fact(5));

}

}

**Nesting of Methods :**

In java, the methods and variables which we create in a class can only be called by using the object of that class or, in case of static methods, we can directly call it by using the name of the class. The methods and variables can be called with the help of the dot operator. But there is a special case that a method can also be called by another method directly, but the condition is they should be present in the same class.

**Syntax:**

**class Main**

**{**

method1()

**{**

**// statements**

**}**

method2()

**{**

**// statements**

**// calling method1() from method2()**

method1();

**}**

method3()

**{**

**// statements**

**// calling of method2() from method3()**

method2();

**}**

**}**

**Example :**

**import java.util.Scanner;**

**public class Nesting\_Methods**

**{**

int perimeter(int l, int b)

**{**

**int pr = 12 \* (l + b);**

**return pr;**

**}**

int area(int l, int b)

**{**

**int pr =** perimeter(l, b);

**System.out.println("Perimeter:"+pr);**

**int ar = 6 \* l \* b;**

**return ar;**

**}**

int volume(int l, int b, int h)

**{**

**int ar =** area(l, b);

**System.out.println("Area:"+ar);**

**int vol ;**

**vol = l \* b \* h;**

**return vol;**

**}**

**public static void main(String[] args)**

**{**

**Scanner s = new Scanner(System.in);**

**System.out.print("Enter length of cuboid:");**

**int l = s.nextInt();**

**System.out.print("Enter breadth of cuboid:");**

**int b = s.nextInt();**

**System.out.print("Enter height of cuboid:");**

**int h = s.nextInt();**

**Nesting\_Methods obj = new Nesting\_Methods();**

**int vol = obj.volume(l, b, h);**

**System.out.println("Volume:"+vol);**

**}**

**}**

