

The logo for Oracle Academy is centered on a light gray background. It features the word "ORACLE" in a bold, orange, sans-serif font. Below it, the word "Academy" is written in a smaller, dark gray, sans-serif font. The entire logo is framed by two horizontal dark gray bars, one at the top and one at the bottom.

ORACLE

Academy

Java Fundamentals

7-2

Parameters and Overloading Methods

ORACLE
Academy



Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Objectives

- This lesson covers the following objectives:
 - Use access modifiers
 - Pass objects to methods
 - Return objects from methods
 - Use variable argument methods
 - Overload constructors
 - Overload methods
 - Write a class with specified arrays, constructors, and methods



Access Modifiers

- Access modifiers specify accessibility to changing variables, methods, and classes
- There are four access modifiers in Java:

Access Modifier	Description
public	Allows access from anywhere
protected	Allows access only from inside the same class, from a subclass, or from other classes of the same package as the modifier
private	Allows access only from inside the same class as the modifier
"default" (not specified/blank)	Allows access from only inside the same class, or from other classes of the same package as the modifier

public Access Modifier

- public access modifiers allow access from anywhere
- In Java, adding the keyword public as the variable, method, or class is declared, makes the variable, method, or class accessible from anywhere



Classes and methods are usually public.

Declaring as public

- The code below shows how to declare a variable, method, or class as public

–Variable:

```
public int milesRan = 2; //public access
int timePassed = 17; //access not specified
```

–Method:

```
public int addMiles(int a, int b){
    return a+b;
} //end method addMiles
```

–Class:

```
public class Jogging{
    //class code here
} //end class Jogging
```

Local variables never have an access modifier. Only instance variables use access modifiers.

protected and "default" Access Modifiers

- A protected access modifier allows access inside the class, subclass, or other classes of the same package as the modifier
- To declare a variable, method, or class as protected, write the keyword protected rather than public
- A "default" access modifier allows access from inside the same package only
- To declare a variable, method, or class as "default," do not include an access modifier

A common mistake new Java programmers make is to not specify an access modifier and thus get the default.

private Access Modifier

- A private access modifier:
 - Only allows access from inside the same class
 - Is the most restrictive access modifier
 - Is the opposite of the public access modifier

```
private int bankAccountNumber;
```



Fields are usually private.

When to Use public or private

Type	Definition	When to Use
public	Allows access from anywhere	When it does not matter that anyone can access your code or when you wish to share your code with others
private	Allows access only from inside the same class	When it is important that your code is secure and cannot be accessed from anywhere but inside the class itself

Objects as Parameters

- A parameter is a variable in a method declaration that is passed to the method

```
public int method(int parameter1, int parameter2)
```

- Parameter types are the type of parameters that can be passed to a method
- This includes:
 - Primitive types (such as int, double, char)
 - Objects
 - String
 - Array

```
public int method(int anInt, double aDouble,  
                  String aString, MyClassName anObjectOfMyClass)
```

It makes no difference if the object is from an API class or if it is from a user created class.

Objects as Parameters Example

- An employer has an opening for a promotion for one of his employees
- He wishes to create a method that will take in an employee as a parameter and calculate and return the employee's rating based on their qualifications for the new position

```
public int promotion(Employee E) {  
    int timeEmployed = E.getLengthOfEmployment();  
    //do some calculations to set a rating for E  
    return rating;  
} //end method promotion
```

Passing Objects as Parameters

- Passing objects as parameters allows for much easier access to the information that the object contains
- It also permits making changes to objects inside of the method, and even allows for comparing two objects that cannot use primitive comparing methods



Returning Objects

- Writing a method that returns an object is very similar to writing a method that returns a primitive type
- For example, the employer from the previous example just learned that methods can return an object
- To make it easier to find the employee to promote, he can write a method that takes in two employees
- The method returns the one that has a better rating
- This is easier than going through each employee, retrieving each of their ratings, and then comparing them

Returning Objects Example

- Employee identifies what is being returned

To return an object, simply write the object type here.

```
public Employee promotion(Employee A, Employee B){  
    //calculate to compare which employee is better  
  
    //if employee A is better  
    return A;  
    //if employee B is better  
    return B;  
} //end method promotion
```

Variable Argument Methods

- A variable argument method:
 - Is a method written to handle a variable number of arguments
 - Only works if you call the method with the same type of argument as the method requires
- A variable argument method looks like this:

```
public int total(int ... nums){  
    int sum = 0;  
    for(int i = 0; i < nums.length; i++)  
        sum += nums[i];  
    return sum;  
} //end method total
```

Variable Argument Methods Example

- For example, a method initialized with a variable argument of integers cannot be called with any number of Strings, but can only be called with any number of integers for the argument
- If another method is declared with a variable argument of Strings, they must call that method with String(s) to meet the arguments

Why not use Arrays in Variable Argument Methods?

- Why not just use an array?
 - In a program, you must know the number of elements in an array to create one
 - If the number of elements changes, you would need a different array for each different length
 - Using a variable argument method allows for use of the method without ever having to initialize an array
 - It also allows for multiple uses with a variable number of elements

Variable Argument Methods and Integers

- Does a variable argument method only work with integers?
 - No, the variable argument works with any primitive type, object, and even arrays
 - You can have a variable argument of arrays

Employee Example

- To determine employee promotions, the employer was coding a method that compared two employees and returned the best one
 - Now that the employer has the method to compare the employees, he needs a way to compare all the employees at once instead of only comparing two at a time
 - This is where variable arguments would help

Variable Argument Employee Example Code

- Code to compare all employees:

```
public Employee promotion(Employee ... employees){
    Employee bestCandidate = employees[0];
    /*go through the list of employees and calculate
       which one is the best candidate*/
    for(int i = 1; i < employees.length; i++){
        //if there is a candidate better than the current best
        if(employees[i].getRating > bestCandidate.getRating){
            //update the bestCandidate to the better one
            bestCandidate = employees[i];
        }//end if
    }//end for
    //return the best candidate found for the promotion
    return bestCandidate;
} //end method promotion
```

Calling a Method with Variable Arguments

- Calling a method with variable arguments is like calling any other method
- However, it can be called with a different number of arguments each time it is called



Calling a Method with Variable Arguments

- The code below demonstrates this idea
- Sam, Erica, Dominic, Sandy, and Jake are employees
- The employer is looking to promote either Sam, Erica, or Dominic to manager and Sandy or Jake to assistant manager

```
/*This compares Sam, Erica, and Dominic and assigns  
the best candidate of the 3 to newManager.*/  
Employee newManager = promotion(sam, erica, dominic);  
  
/*This compares Sandy and Jake and assigns the better  
of the 2 to newAssistantManager*/  
Employee newAssistantManager = promotion(sandy, jake);
```

Overloading Constructors

- Constructors assign initial values to instance variables of a class
- Constructors inside a class are declared like methods
- Overloading a constructor means having more than one constructor with the same name but different types and/or numbers of arguments



Overloading Constructors Example 1

- This example overloads the public constructor of a Dog class

```
public class Dog{  
    public Dog(){  
        ...implementation...  
    }//end constructor  
    public Dog(int weight){  
        ...implementation...  
    }//end constructor  
    public Dog(String barkNoise){  
        ...implementation...  
    }//end constructor  
    public Dog(int weight, int loudness, String barkNoise){  
        ...implementation...  
    }//end constructor  
}//end class Dog
```


How Overloading Constructors Works

- Overloading constructors works as follows:
 - Java reads the constructor based on what arguments are passed into it
 - Once it identifies the constructor name, it will compare the argument types
 - If the argument types do not match the first constructor of that name, it will proceed to the second, third, and so on until it identifies a constructor name and argument type match
 - If it does not find a match, then the program will not compile

Overloading Constructors Example 2

```
public class Dog{  
    private int weight;  
    private int age;  
    private String barkNoise;  
  
    public Dog(){  
        weight = 12;  
        loudness = 4;  
        barkNoise = "Woof";  
    }//end constructor  
  
    public Dog(int w, int l){  
        weight = w;  
        loudness = l;  
        barkNoise = "ARF!";  
    }//end constructor  
  
    public Dog(int w, int l, String bark){  
        weight = w;  
        loudness = l;  
        barkNoise = bark;  
    }//end constructor  
}//end class Dog
```

This is a constructor that specifies the dog's weight and loudness in the arguments.

This is a constructor that specifies the dog's weight, loudness, and bark noise in the arguments.

Overloading Constructors Example 2 Explained

- Dog() is the default constructor
- A default constructor has no arguments
- If you initialized a Dog object using this constructor, it would have a weight of 12, a loudness of 4, and a bark noise of "woof"
- The last two constructors in the Dog class allow the assignment of instance variables to differ according to specifications during initialization

Overloading Constructors Example 2 Explained

- Although the default Dog constructor has code to initialize the class variables, it is optional for a default constructor to have code
- If the default constructor does not have code, the class variables are initialized with:
 - null for objects
 - 0 (zero) for primitive numeric types
 - false for boolean

Overloading Constructors Example 2 Explained

- If a constructor is not written for a class, the default constructor (with no code) is supplied by the JVM
- If there is not a default constructor written, and there are one or more other constructors, the JVM will not supply a default constructor

Take note of the 2nd bullet point, as it is frequently overlooked

Overloading Methods

- Like overloading constructors, overloading a method occurs when the type and/or number of parameters differ
 - Below is an example of a situation where a method would need to be overloaded
 - Create the Dog class, then create an instance of Dog in a Driver Class
 - Call (use) both bark() methods

```
public class Dog{  
    private int weight;  
    private int loudness;  
    private String barkNoise;  
  
    public void bark(String b) {  
        System.out.println(b);  
    } //end method bark  
    public void bark() {  
        System.out.println("Woof");  
    } //end method bark  
} //end class Dog
```

A common mistake is to attempt to overload a method by only changing the return type. Overloading methods only involves changing the parameter types and the number of parameters.

Terminology

- Key terms used in this lesson included:
 - Access modifier
 - Constructor
 - Default constructor
 - Overloading
 - Private access modifier
 - Public access modifier
 - Variable argument method

Summary

- In this lesson, you should have learned how to:
 - Use access modifiers
 - Pass objects to methods
 - Return objects from methods
 - Use variable argument methods
 - Overload constructors
 - Overload methods
 - Write a class with specified arrays, constructors, and methods



The Oracle Academy logo is centered on a light gray background. It features the word "ORACLE" in a bold, orange, sans-serif font. Below it, the word "Academy" is written in a smaller, dark gray, sans-serif font. The entire logo is framed by two horizontal dark gray bars, one at the top and one at the bottom.

ORACLE

Academy