Coding Bootcamp

Core Java

TECHNOLOGY

Data Handling and Function

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
    cout << "Hello, world!" << endl;
    return 0;
}
```

Learning Objectives

By the end of this lesson, you will be able to:

- Analyze the features of arrays in Java and demonstrate how to declare and initialize them
- Evaluate the various types of functions and differentiate the components that make up a function
- Categorize the different parameters of functions and differentiate function arguments from overloading
- Implement static polymorphism in method overloading and evaluate the use of string handling and string buffers
- List and justify the constructors of the StringBuffer class, and describe the supported functions for strings

TECHNOLOGY

Arrays

simplilearn

Arrays

An array is a data structure that allows users to store a fixed-size sequence of values of the same data type.

It is an information structure to store comparable elements.

It contains elements of similar data types.

Illustration

```
public class Main
{
public static void main(String[] args)
{
// Declare and initialize an integer array with
10 elements
int[] myArray = { 10, 20, 30, 40, 50, 60, 70, 80
90, 100 };
// Access and print the second element of the
array
System.out.println(myArray[1]);
}
}
```

Arrays

To access a value in an array, users need to use the index of the element to which they want to access it.

Here, an array of integers called Student_ID has been declared and initialized.

To access the value at index 0, [0] notation is used.

```
public class Main {
    public static void main(String[] args) {
        // Declare and initialize an array of
integers
        int[] Student_ID = {111, 112, 113, 114,
115};
        // Access the value at index 0 (first
element)
        int value = Student_ID[0];
        // Print the value
        System.out.println(value); // Output: 111
    }
}
```

Note

The index of an array starts from zero, which means the first element of the array has an index of 0, the second has an index of 1, and so on.

Arrays

The size of an array is determined at the time of creation and cannot be changed during the program's execution.

```
// declare an array of integers with
a size of 5
int[] myArray = new int[5];
```

Once an array is initialized to hold five elements, it cannot accommodate any additional elements beyond that limit.

TECHNOLOGY

Types of Arrays

Single-Dimensional Array

It is a data structure that holds a fixed-size sequential collection of elements of the same type. It is also called a one-dimensional array.

Example:

```
int[] array1
```

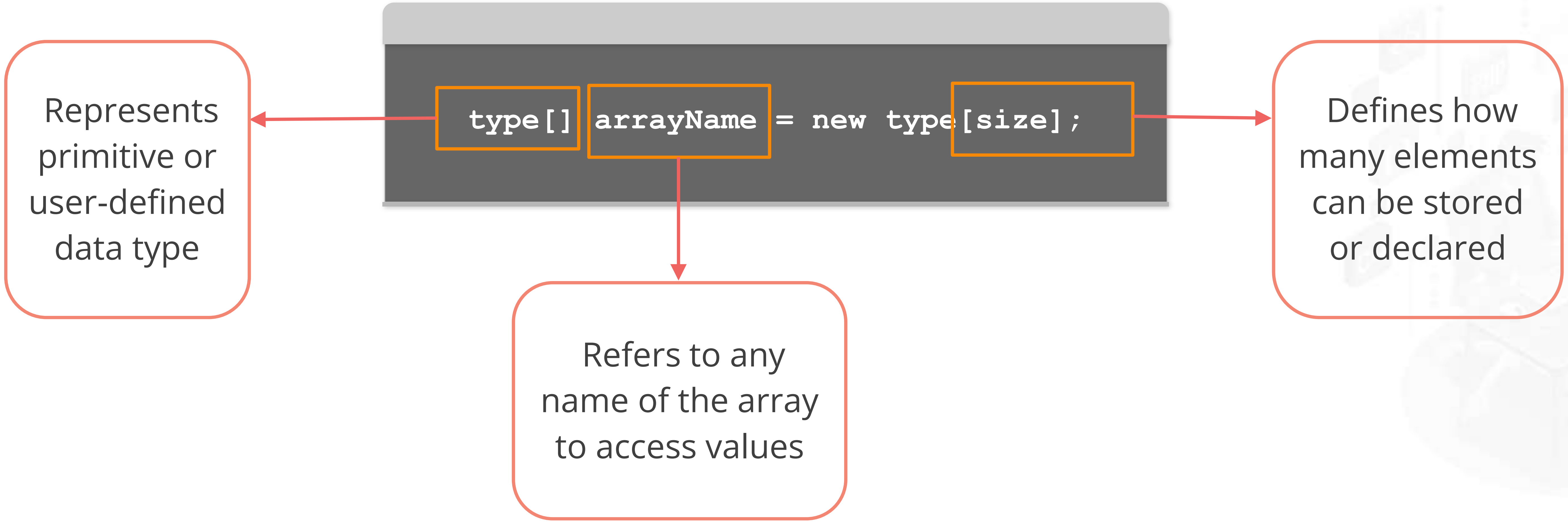
Here, the declaration "int[] array1" creates an array variable called "array1" that can hold a sequence of integers.

Note

To declare a single-dimensional array, users must first specify the data type and the number of elements in the array.

Single-Dimensional Array

To create a single-dimensional array, use this syntax:



```
type[] arrayName = new type[size];
```

Represents primitive or user-defined data type
Refers to any name of the array to access values
Defines how many elements can be stored or declared

Single-Dimensional Array

To create a **string array**, specify the length of the array within the square brackets [ ]

Syntax to create a String Array:

```
String[] names= new String[length];
```

Here, **length** is the number of elements users want to create in the array.

To create an **integer array**, specify the size of the array within the square brackets [ ]

Syntax to create an Integer Array:

```
int[] numbers= new int[size];
```

Here, **size** is the number of elements users want to store in the array.

Single-Dimensional Array

To create a user-defined type, define a class

Syntax to create a user defined type:

```
    class Users {
     String name;
     String email;
     int age;
  }
```

Here, **Users** is a defined class name with three instance variables: **name**, **email** of type String, and **age** of type int.

Single-Dimensional Array

The syntax for declaring arrays is shown below:

```
Users[] users = new Users[10];
```

Here, variable **users** is assigned an array of **Users** objects with a capacity of ten elements.

Note

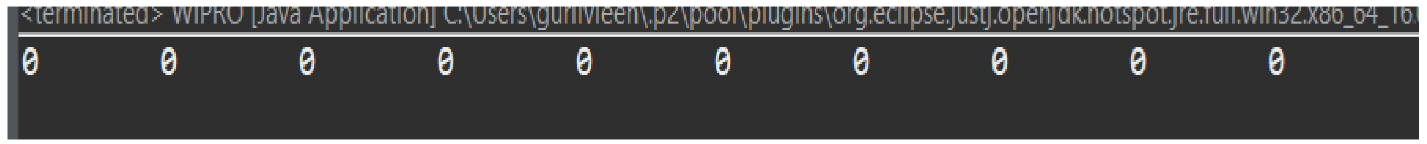
A **new** keyword is used to create a new instance of the array, the size of which is specified in square brackets after the class name.

Single-Dimensional Array

Syntax to read elements:

```java
int[] numbers = new int[10];
for (int i = 0; i < numbers.length; i++)
{
  System.out.print(numbers[i] + "\t");
}
```

Output:



```
<terminated> WIPRO [Java Application] C:\Users\gurlvleen\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16
0       0       0       0       0       0       0       0       0       0
```

Syntax to create Array without using new operator:

```java
int[] users = { 1, 2, 3, 4, 5, 6, 7, 8,
9, 10 };
```

Here, an integer array named **users** is created with ten elements.

Multi-Dimensional Array

Users can define an array of arrays by using a set of two or more square brackets to specify the size of each array.

Example:

```
String[][] names;
```

In this example, the variable **names** is defined as an array that can contain multiple arrays of strings. The first set of brackets [ ] represents the size of the primary array, while the second set of brackets [ ] specifies the size of each nested array.

Multi-Dimensional Array

Syntax to create a String Array:

```
String[][] names= new
String[rowCount][columnCount];
```

Here, **rowCount** is the number of rows, and **columnCount** is the number of columns in the array.

Syntax to create an Integer Array:

```
int[][] numbers= new
int[row][column];
```

Here, **row** represents the number of rows, and **column** represents the number of columns in the array.

Multi-Dimensional Array

Syntax to create a multi-dimensional array without using a new operator:

```
int[][] numbers = {
      {1, 2, 3, 4, 5},
      {1, 2, 3, 4, 5}
};
```

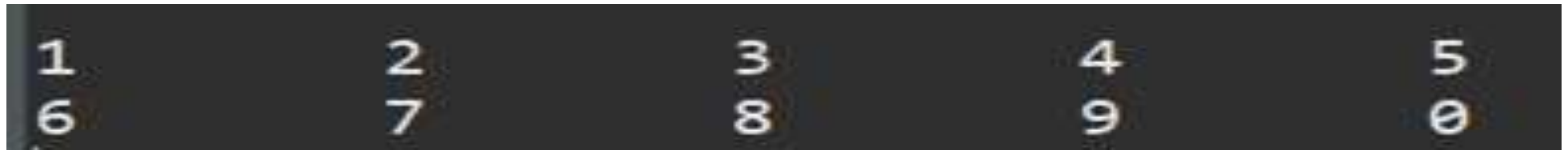
Here, the array **numbers** has two rows and five columns with integer values specified in curly braces. Both rows contain the values 1 through 5.

Multi-Dimensional Array

Syntax to read elements in a Multi-Dimensional Array:

```
int[][] numbers = {
     { 1, 2, 3, 4, 5 },
     { 6, 7, 8, 9, 0 }
};
for (int i = 0; i < numbers.length; i++) {
     for(int j = 0; j < numbers[i].length; j++) {
System.out.print(numbers[i][j] + "\t");
}
System.out.println("");
}
```

Output:



```
1       2       3       4       5
6       7       8       9       0
```

Working with Arrays

Problem Statement:

You have been asked to create and use one-dimensional and two-dimensional arrays in Java.

Outcome:

By creating and using one-dimensional and two-dimensional arrays in Java, you will learn how to store and manipulate groups of related data efficiently. This knowledge will enhance your ability to handle complex data structures and improve the functionality of your Java applications.

Note: Refer to the demo document for detailed steps: 01_Working_with_Arrays

Assisted Practice: Guidelines

Steps to be followed are:

1. Use a one-dimensional array citing suitable examples
2. Create a one-dimensional array in different ways
3. Use the new operator to create an array inside a heap
4. Access elements in an array
5. Create an array with a specific size
6. Implement a two-dimensional array with suitable examples

Using Arrays for Managing Covid Case Data

Problem Statement:

You have been asked to depict the various use cases of Covid with two dimensional arrays in Java.

Outcome:

By using two-dimensional arrays in Java to handle COVID data, you will learn to manage and analyze complex datasets like infection rates and vaccine distribution. This skill is vital for real-world data handling in Java applications.

Note: Refer to the demo document for detailed steps: 02_Using_Arrays_for_Managing_Covid_Case_Data

Assisted Practice: Guidelines

Steps to be followed are:

1. Represent indexes or column names
2. Run the index loop
3. Run the code and filtering data
4. Use the switch case statements
5. Implement filtration of data and execute it with example data

TECHNOLOGY

Functions

Functions

A **function** is a code block that performs a specific task and can be called by other parts of a program.

Functions provide several benefits. Some are:

Reusability of code

By writing a function once, one can avoid duplicating code and use it repeatedly whenever that functionality is required.

Update the code

It helps to reduce the amount of code duplication in your program, which can make it easier to maintain and update.

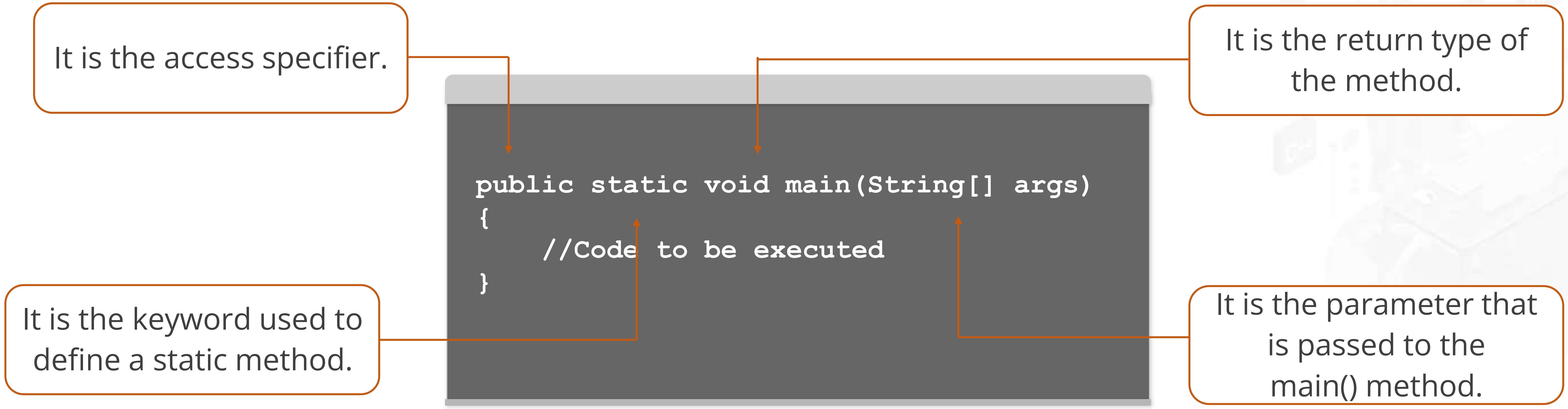
Note

A function is also referred as a method in Java.

Functions

The **main() function** is the entry point of the program. It is the first method executed when a program starts running.

The syntax of the **main()** function is:



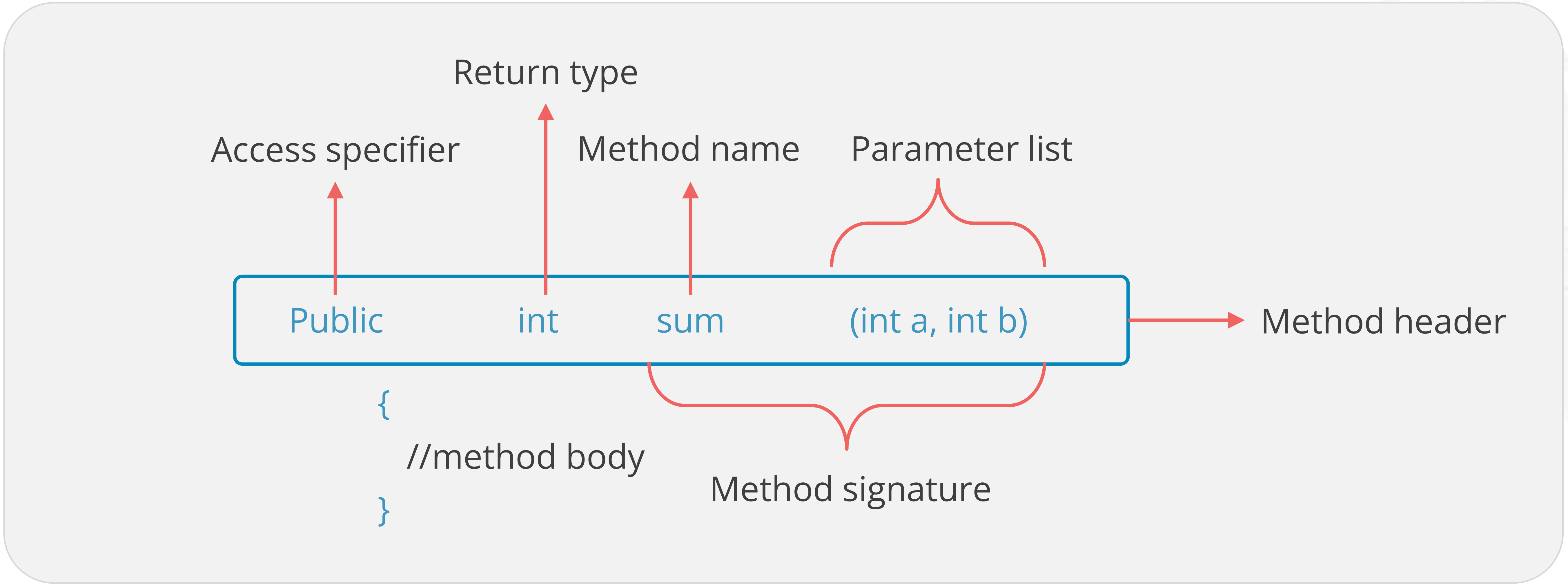
```
public static void main(String[] args)
{
    //Code to be executed
}
```

It is the access specifier.
It is the return type of the method.
It is the keyword used to define a static method.
It is the parameter that is passed to the main() method.

Function Declaration

A function declaration is a statement that defines a new function within a class.

It typically includes the following elements:



Function Signature

It is a unique identifier of a method that helps the compiler differentiate it from other methods with the same name in the same class.

```
public int add(int num1, int num2)
{
    return num1 + num2;
}
```

Here, the function signature for this method is add(int, int), which includes the function name (add) and the parameter types (int and int).

Note

It is part of a method declaration and includes the method name, the number of parameters it takes, and their data types.

Access Specifier

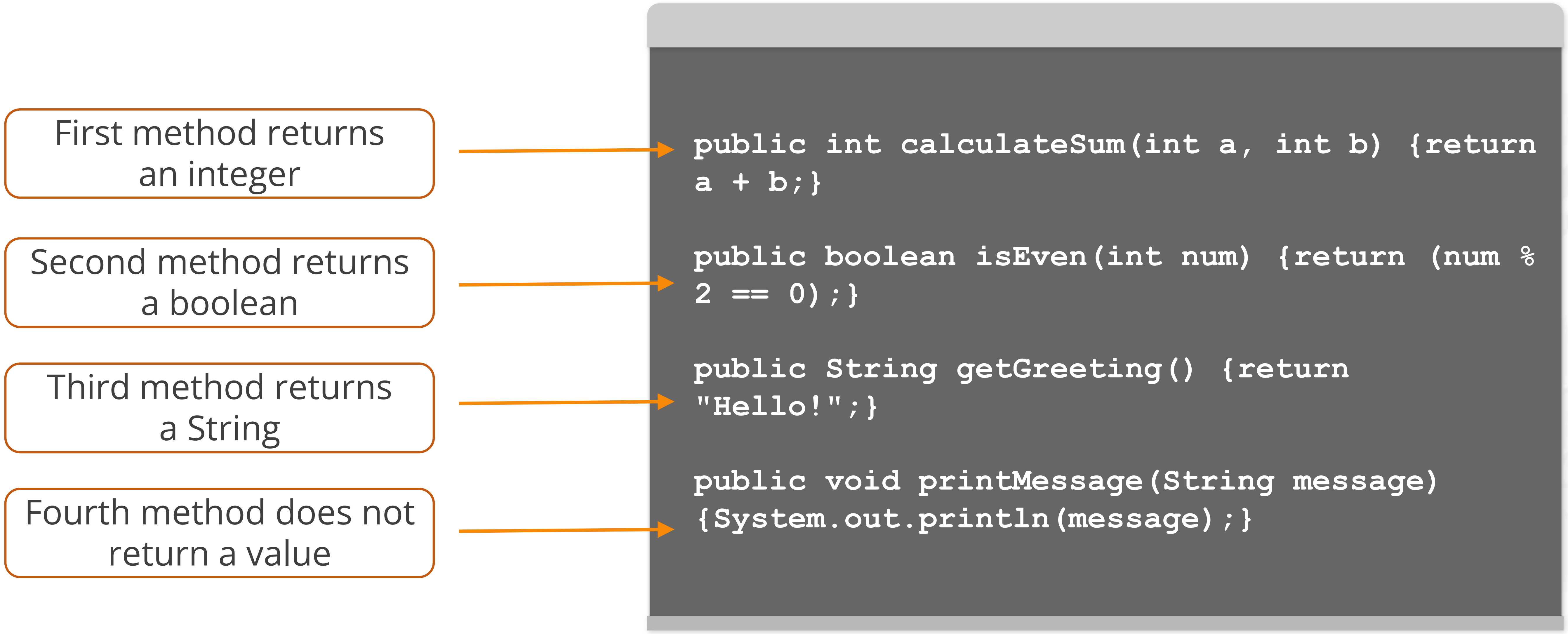
Access specifiers control the visibility and accessibility of class members from other classes and objects.

The four types of access specifiers are:

Public	Accessible from anywhere, including outside class and packages
Private	Only accessible within the declaring class
Protected	Accessible within class, subclasses, and same package, but not outside
Default	Accessible within the same package, but not outside

Return Type

The return type is the data type of the value a method returns when called.


First method returns an integer
Second method returns a boolean
Third method returns a String
Fourth method does not return a value

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

public boolean isEven(int num) {return (num %
2 == 0);}

public String getGreeting() {return
"Hello!";}

public void printMessage(String message)
{System.out.println(message);}
```

The return type can be any of the primitive or reference data types. If a method does not return a value, its return type is void.

Function Name

The function should be given a name that distinguishes it from others in the same class and accurately describes its intended purpose and behavior.

Here, **method_name** is the name of the function,

```
[access _modifier] return_type
method _name(parameter list){//
method body}
```

Note

While defining a function, the name should be in lowercase and a verb.

Function Name

Example:

```
Single-word function name: perimeter(), ratio()
Multi-word function name: perimeterOfSquare()
```

Note

Function overloading is a condition when the function has the same name in the same class.

Parameter List

It specifies the type and name of each parameter, separated by commas and enclosed in parentheses.

```
public void doSomething(int x,
String y, boolean z)
{
  // method body
}
```

A parameter list includes three parameters, which are type int, string, and boolean, with the names x, y, and z, respectively.

```
public void doSomething()
{
  // method body
}
```

The parameter list can be empty if the method doesn't require any parameters.

Function Body

It is a block of code that defines the behavior of a method. It consists of statements and expressions executed when the method is called.

```
public int add(int a, int b)
{
int sum = a + b; // defining a new
variableto store the sum of a and b
return sum; // returning the sum tothe
caller
}
```

Note

The function body is enclosed in curly braces { }, which defines the scope of the function.

Predefined Function

These functions are provided by Java's standard libraries and can be used directly in a Java program without needing to be defined.

The syntax for calling a predefined function:

```
functionName(argument1, argument2,
..., argumentN);
```

Here, **functionName** is the name of the predefined function, and argument1, argument2, ..., argumentN are the values passed to the function as input.

Predefined Function

Predefined functions are often called **methods**, and each method is defined within a class.

length() function

It is defined within the **java.lang.String class**. This method is used to get the length of a string.

equals() function

It is also defined within the **java.lang.Object class**. This method is used to compare two objects for equality.

Example:

The print() method is defined within the **java.io.PrintStream** class, which is part of the java.io package.

Predefined Function

Example:

```
public class Example
{
public static void main(String[] args)
{
// using the min() function of Math class
System.out.print("The maximum number is: " +
Math.min(4,5));
}
}
```

User-Defined Function

These functions are created by the programmer to perform a specific task or set of tasks. They are also known as custom methods or user-created methods.

The syntax for user-defined function:

```
[access-modifier] [static] [return-
type] methodName(parameter-list) { //
method body }
```

Here, **access-modifier** is an optional keyword that determines the accessibility of the method. It can be public, private, protected, or left out entirely.

User-Defined Function

Example:

```
public static void findOddEven(int num)
{
//function body
if(num%2!=0)
System.out.println(num+" is odd");
else
System.out.println(num+" is even");
}
```

Note

The static function is the most used user-defined function.

Static Function

A static function is a method that belongs to a class rather than an instance of that class.

```
public class MyClass { public static
void myStaticFunction() { // code goes
here } }
```

To declare a static function, the keyword **static** is placed before the method declaration.

Note

There is no need to create an object to call a static function.

Static Function

Static functions can access static data members and manipulate the values.

Example:

```
public class example
{
public static void main(String[] args)
{
show();
}
static void show()
{
System.out.println("Example for static function.");
}
}
```

Function Argument

Function Argument

When a function or method is called or invoked, the values passed as input to the function are called arguments.

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

In this case, a and b are parameters that the function expects to receive. When invoking this function, two values can be passed in as arguments.

Function Argument

When a function is defined with certain parameters, those parameters serve as placeholders for the actual values passed in when the function is called.

Example:

```
class Calculator{
int product(int a, int b) {
return a*b;
}
}

public class Example {

public static void main(String[] args) {
Calculator cal = new Calculator();
System.out.println(cal.product(5, 2));

}
}
```

The **product()** function accepts two integer arguments, a and b, and returns their product. When called with arguments 5 and 2, the function replaces a and b with the corresponding values and returns their product, which is 10.

Function Parameters

Actual parameters

These are the values passed to a function when it is called. They are assigned to the formal parameters within the function.

Formal parameters

These parameters are defined in the function header and act as placeholders or signatures for the values that will be passed as arguments when the function is called.

Function Argument

Arguments in a function are passed in two ways:

Pass by value

In this method, a copy of the actual value of the argument is passed to the function parameter.

Example:

```
void incrementCount(int count)//pass by value{
count=count+1;//increments the value of
countinside the function}
int main()
{
int count=0;// initialze the variable count
int result=0;// initialze the variable result
incrementCount(count);//call increment
functioncout<<"Pass by value\n";
cout<<"Count:";
cout<<count;//prints the value of count after
the function call
return 0;
}
```

Function Argument

Arguments in a function are passed in two ways:

Pass by reference

In this method, a reference to the actual value of the argument is passed to the function parameter.

Example:

```
void incrementCount(int & count)//& to pass
byreference{count=count+1;//increments the
value of count}
int main()
{
int count=0;//initialize the variable countint
result=0;// initialize the variable
resultincrementCount(count);//increment value
of countcout<<"Pass by Reference\n";
cout<<"Count:";
cout<<count;//prints count after the
functioncall
return 0;
}
```

Function Overloading

Function Overloading

It allows a class to have multiple methods with the same name but different parameters.

In this case, there are two methods with the same name, **overloadedfunction** but different parameter lists.

Created an object for the **OverloadingExample** class and called the two overloaded methods with different arguments

Example:

```
public class OverloadingExample {
 // overloaded function
 void overloadedfunction(int i){
  System.out.println("In overloaded function with int
parameter- " + i);
 }
 // overloaded function
 void overloadedfunction(int x, String str){
  System.out.println("In overloaded function with int
and string parameters- " + integer + " " + string);
 }
 public static void main(String args[]){
  OverloadingExample obj = new OverloadingExample();
  obj.overloadedfunction(8);
  obj.overloadedfunction(8, "Hello");
 }
}
```

Function Overloading

It allows a class to have multiple methods with the same name but different parameters.

Example:

```
public class OverloadingExample {
 // overloaded function
 void overloadedfunction(int i){
  System.out.println("In overloaded function with int
parameter- " + i);
 }
 // overloaded function
 void overloadedfunction(int x, String str){
  System.out.println("In overloaded function with int
and string parameters- " + integer + " " + string);
 }
 public static void main(String args[]){
  OverloadingExample obj = new OverloadingExample();
  obj.overloadedfunction(8);
  obj.overloadedfunction(8, "Hello");
 }
}
```

Takes an integer parameter

Takes an integer and a string parameter

Function Overloading

Output:

```
In overloaded function with int parameter- 8
In overloaded function with int and string parameters- 8
Hello
```

Using Methods in Java

Problem Statement:

You have been given a task to depict how methods are implemented in Java.

Outcome:

By depicting how methods are implemented in Java, you will learn to define reusable blocks of code that perform specific tasks, enhancing modularity and readability in your applications. This knowledge is essential for writing efficient and maintainable Java code.

Note: Refer to the demo document for detailed steps: 03_Using_Methods_in_Java

Assisted Practice: Guidelines

Steps to be followed:

1. Write an algorithm and implement the same
2. Write arrays with cited examples
3. Run the code and get the output
4. Create and use a non-static method
5. Create an object with an object construction statement
6. Differentiate between running a static and a non-static method

Overloading Methods in Java

Problem Statement:

You have been given a task to depict how to overload methods in Java.

Outcome:

By depicting how to overload methods in Java, you will learn to define multiple methods with the same name but different parameters. This enables you to handle varying inputs more efficiently, enhancing the flexibility and readability of your Java applications.

Note: Refer to the demo document for detailed steps: 04_Overloading_Methods_in_Java

Assisted Practice: Guidelines

Steps to be followed:

1. Write and implement method overloading rules in Java
2. Implement compiler time polymorphism
3. Test the overloaded methods
4. Run the code and execute the result
5. Write methods for authentication

Sorting Arrays with Methods

Problem Statement:

You have been given a task to sort an array using methods and a bubble sort algorithm.

Outcome:

By sorting an array using methods and the bubble sort algorithm in Java, you will learn how to implement basic sorting techniques within your programs. This task will enhance your understanding of algorithm efficiency and data manipulation, crucial for developing effective and optimized code.

Note: Refer to the demo document for detailed steps: 05_Sorting_Arrays_with_Methods

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Create a class and a main method
2. Create a method of static void print to print the array
3. Execute the bubble sort method and reprint the array
4. Calculate the length of the array and swap values
5. Dry run the bubble sort algorithm and add a debugger view

TECHNOLOGY

Static Polymorphism

simplilearn

Static Polymorphism

It is used with the help of method overloading. It allows you to define multiple methods in a class with the same name but different parameters.

Example:

```
class Calculator{
int product(int a, int b) {
return a*b;
}
int product(int a, int b, int c) {
return a*b*c;
}
}

public class Example {

public static void main(String[] args) {
Calculator cal = new Calculator();
System.out.println(cal.product(5, 2));
System.out.println(cal.product(2, 3, 4));
}
}
```

Static Polymorphism

Output:

```
10
24
```

The first version takes two parameters, and the second one takes three parameters.

The compiler looks at the method signature during compilation.

TECHNOLOGY

String Handling

String

Strings are represented as objects of the String class. When a string is created using double quotes, it is a string constant or a string literal.

Example:

```
 System.out.println("this is also a
String");
```

Here, **this is also a String** is a string literal, which is automatically converted to a string object by the compiler.

Note

Objects of type string are immutable.

String Handling

Syntax to create string objects:

```
class StringDemo {
public static void main(String
args[]) {
String strObl = "Hi"; String strOb2
="Hello";
 String str0b3 = strObl + " and "+
strOb2;
System Out.println(strObl);
System Out.println(strOb2);
System.Out.println(strOb3);
}
}
```

Creates three string objects using string literals, while strobj3 is created by concatenating strobj1 and strobj2 using the + operator

Syntax to create and initialize string objects:

```
Public String Class ()
Public String Constructor: ()
public String (String)
public String (char []) public
String (byte [I) public String (char
[], int offset, int no_of_chars)
public String (byte [I, int offset,
int no_of _bytes)
```

These are different constructors of the String class in Java that can be used to create new String objects with different initial values based on various input parameters.

String Handling: Functions

Syntax

```
class StringHandling {
  public static void main(String arg[]) {
    int length;
    char c;
    String s = new String("Championship");
    length = s.length();
    System.out.println("Length: " + length);
    c = s.charAt(2);
    System.out.println("Character: " + c);
    System.out.println("String: "+s.toUpperCase());
    System.out.println("String: "+s.toLowerCase());
  }
```

length(): Returns the number of characters in a string

charAt(index): Returns the character at the specified index in a string

toUpperCase(): Converts all the characters in a string to uppercase

toLowerCase(): Converts all the characters in a string to lowercase

String Handling

Syntax of the concat() function:

```
class StringHandling
{
public static void main(String arg[])
{
String s1="John";
String s2="Mathew";
System.out.println("Combined String:
"+s1.concat(s2));
}
}
```

This function concatenates two strings. It returns a new string that is the concatenation of the two input strings.

String Handling

Syntax of the equals() function:

```
class StringHandling
{
public static void main(String arg[])
{
String s1="John";
String s2="Mathew";
String s3="John";
System.out.println("Compare String:
"+s1.equals(s2));
System.out.println("Compare String:
"+s1.equals(s3));
}}
```

It is used to check if two strings are equal in content. It returns a boolean value of true if the two strings are equal in content and false otherwise.

Syntax of the equalsIgnoreCase() function:

```
class StringHandling
{
public static void main(String arg[])
{
String s1="John";
String s2="JOHN";
String s3="Mathew";
System.out.println("Compare String:
"+s1.equalsIgnoreCase(s2));
System.out.println("Compare String:
"+s1.equalsIgnoreCase(s3));
}}
```

This is similar to equals(), but it ignores the case of the letters in the two strings being compared.

String Handling

Syntax of the compareTo() function:

```
class StringHandling
{
public static void main(String arg[])
{
String s1="John";
String s2="Mathew";
int i;
i=s1.compareTo(s2);
if(i==0)
{
System.out.println("Strings are same");
}
else
{
System.out.println("Strings are not same");
}}}
```

It is used to compare two strings lexicographically. It returns an integer value that represents the difference between the two strings.

Syntax of the compareToIgnoreCase() function:

```
class StringHandling
{
public static void main(String arg[])
{
String s1="John";
String s2="JOHN";
int i;
i=s1.compareToIgnoreCase(s2);
if(i==0)
{
System.out.println("Strings are same");
}
else
{
System.out.println("Strings are not same");
}}}
```

It is similar to compareTo(), but it ignores the case of the letters in the two strings being compared.

String Handling

Syntax

```
class StringHandling
{
public static void main(String arg[])
{
String s="Java is programming language";
System.out.println(s.startsWith("Java"));
System.out.println(s.endsWith("language"));
System.out.println(s.substring(10)); // 10 is
starting index
System.out.println(s.indexOf("programming"));
System.out.println(s.trim());
}
}
```

startsWith(): Checks whether the given string starts with a specified prefix

endsWith(): Checks whether the given string ends with a specified suffix

substring(): Returns a substring of the given string, starting from the specified index

indexOf(): Returns the index of the first occurrence of the specified substring in the given string

trim(): Removes any leading and trailing whitespace characters from the given string

String Handling

Syntax of the split() function:

```
class StringHandling
{
public static void main(String arg[])
{
String s="abc@test.com";
String[] s1=s.split("@");  // divide string
based on @
for(String c:s1) //  foreach loop
{
System.out.println(c);
}}}
```

Syntax of the replace() function:

```
class StringHandling
{
public static void main(String arg[])
{
String s1="Hello";
String s2=s1.replace('e', 'k');
System.out.println(s2);
}
}
```

replace() is used to replace occurrences of a character or substring in a given string with a new character or substring.

split() is used to split a string into an array of substrings based on a given delimiter or regular expression.

TECHNOLOGY

StringBuffer

StringBuffer

It is the companion of the Java string class that has a mutable sequence of characters.

✓ Length and content can be changed with certain methods.

✓ It helps to make changes to the string class object by inserting or appending to the string.

Every time the StringBuffer is modified, a new string object is not created.

Constructors in StringBuffer Class

Public StringBuffer()	Constructs a StringBuffer having no character with an initial capacity of 16 characters
Public StringBuffer(int capacity)	Constructs a StringBuffer having no character with a particular initial capacity
Public StringBuffer(String str)	Constructs a StringBuffer with a particular String
Public String Buffer(CharSequence seq)	Constructs a StringBuffer with the same characters

Constructors in StringBuffer Class

StringBuffer is a thread-safe class with a capacity that synchronizes access to its internal array. If the character sequence length stays within the capacity, it avoids unnecessary array allocations.

Public int capacity()

It returns the capacity of an object or data structure.

Public int length()

It returns the length of an array or the number of characters in a string.

Constructors in StringBuffer Class

Example:

```
public class StringBufferCapacity {
 public static void main(String[] args)
  StringBuffer sb = new StringBuffer();
  System.out.println("length " + sb.length());
  System.out.println("capacity " + sb.capacity());
 }
}
```

Output

```
length 0
capacity 16
```

StringBuffer Methods

Append and **Insert** are the main operations in StringBuffer.

append() method

The StringBuffer object appends characters or strings to its current string. Its append() method accepts a single argument, which can be any data type that can be converted to a string, including characters and strings.

insert() method

It adds characters or other strings at a specified position in the current string. It takes two arguments: the index at which to insert the new string and the string or character to be inserted.

StringBuffer Append Method

Example for StringBuffer Append Method

```
public class StringBufferDemo {
 public static void main(String[] args) {
  StringBuffer sb = new StringBuffer();
  StringBuffer sb1 = sb.append("Java").append("is").append("powerful
").append("programming").append(" language ");
  System.out.println("After append -- " + sb.toString());

  if(sb == sb1){
   System.out.println("True");
  }else{
   System.out.println("false");
  }
  String str = new String();
  String str1 = str.concat("This").concat(" is");
  if(str == str1){
   System.out.println("True");
  }else{
   System.out.println("false");
  }
 }
}
```

Output:

```
After append –
Javaispowerfulprogrammingl
anguage
True
false
```

StringBuffer Insert Method

```
public class SBDemo {
 public static void main(String[] args) {
  StringBuffer sb = new StringBuffer("let");
  sb.insert(2, "n");
  System.out.println("After insert -- " +
sb.toString());
 }
}
```

Output

```
After insert -- lent
```

StringBuffer Methods

Java StringBuffer offers certain methods.

toString()

It is a built-in method used to return an object's string representation.

Reverse()

It reverses the sequence of characters and returns a new string.

StringBuffer Method

Example:

```
public class SBRevDemo {
 public static void main(String[] args) {
  StringBuffer sb = new StringBuffer("Test
String");
  sb.reverse();
  System.out.println("reversed - " +
sb.toString());
 }
}
```

Output

```
reversed - gnirtS
tseT
```

Using String Methods

Problem Statement:

You have been asked to explore methods available with string data type.

Outcome:

By exploring methods available with the string data type in Java, you will learn how to manipulate and process text effectively. This will enhance your ability to handle common string operations like concatenation, substring extraction, and pattern matching, vital for any Java developer.

Note: Refer to the demo document for detailed steps: 06_Using_String_Methods

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Write a string and calculate the number of characters in the string
2. Get the index of characters
3. Use the uppercase method to convert the string to uppercase
4. Implement string slicing and extracting sub-strings
5. Use the trim function on strings
6. Implement the concat method for concatenation of strings
7. Validate the strings using the methods ends, contains ends with, and starts with
8. Implement an algorithm to know the number of times a character appears
9. Depict the concept of string comparison and intern strings
10. Use content comparison with the method dot equals

Comparing Mutability and Immutability of Strings

Problem Statement:

You have been given a task to differentiate and use mutable and immutable strings.

Outcome:

By differentiating mutable and immutable strings in Java, you will understand string management and performance optimization in your applications.

Note: Refer to the demo document for detailed steps: 07_Comparing_Mutability_and_Immutability_of_Strings

ASSISTED PRACTICE

simplilearn

Assisted Practice: Guidelines

Steps to be followed:

1. Create a class and write the main method
2. Create and concatenate the strings
3. Differentiate the string buffer and string builder
4. Write a method to accept strings
5. Define classes to implement the char sequence
6. Implement the runtime polymorphic behavior for the interface
7. Pass a regular string, buffer, and builder
8. Use the common methods with strings

Key Takeaways

- Arrays store multiple values of the same type in a single variable for efficient data management.
- Functions, also known as methods, encapsulate reusable blocks of code that perform a specific task.
- Access specifiers control the visibility of classes, methods, and variables: public, private, protected, and default (package-private).
- StringBuffer is used to create mutable string objects, meaning the content can be changed after creation.
- Strings in Java are immutable; methods that modify a string return a new string.
- Static polymorphism, also known as compile-time polymorphism, is achieved through method overloading.

TECHNOLOGY

Thank You

simplilearn