SmartCliff
CAREER MOBILITY SOLUTIONS

SDE Readiness Training

Empowering Tomorrow's Innovators

Module I

Java Software Development: Effective Problem Solving

SmartCliff
Career Mobility Solutions

Arrays and Functions

Learning Level: Basic and Easy

DATE: 14.02.2025

Contents

01

Arrays

02

Functions

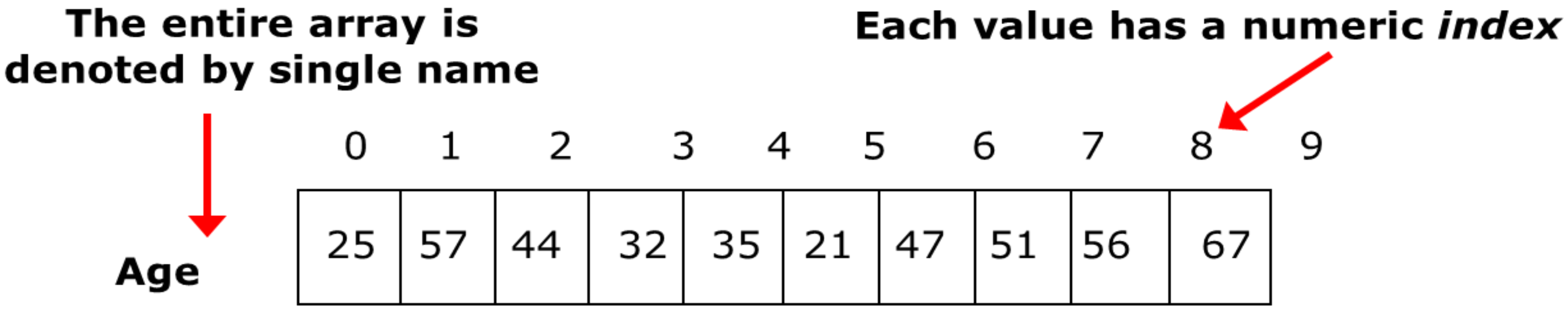
Arrays

Contents

- Introduction
- Single Dimensional Array
- Multi Dimensional Array
- Multi Dimensional Array: Jagged Array
- Quiz

Introduction

- An **array** is a **container** that holds a **fixed number of values** of a **same data type.**
- **Need of Array: Difficult to manage** large number of variable in normal way. The **idea of array** is to **represent many instances in one variable.**
- Length of the **array is fixed** and index **starts with 0. Can't access** the elements **beyond the array limit.**



The entire array is denoted by single name

Each value has a numeric *index*

	0	1	2	3	4	5	6	7	8	9
Age	25	57	44	32	35	21	47	51	56	67

An array of size N is indexed from 0 to N-1

Types

- There are **three types of arrays** in Java:
  - Single Dimensional Array
  - Multidimensional Array
  - Jagged Array

Single Dimensional Array

- **Single dimensional array** is a collection of items under **common name** in **linear array** fashion.
- **Declaration**
- **Syntax**

dataType[] arrayName; (or)

dataType []arrayName; (or)

dataType arrayName[];

Instantiation of an Array in Java

arrayReferenceVariable=**new** dataType[size]

Example

int []salary; ***//declaration***

salary=new int[10]; ***//Instantiation***

Declaration and Instantiation

int salary[]=**new** int[10];

Note:

- Array **declaration** only create **reference of array variable.**
- To **create or give memory** to **array only at** that **time of array instantiation**.

Single Dimensional Array

Create Array

- int [] marks=new int[10]; ***//declaration and instantiation***

(OR)

- int marks[]=new int[10];
- The **[] operator** is the **indication to the compiler** we are **naming an array object** and not an ordinary variable.
- Java Array is object, we use the **new operator** to create an array.

Single Dimensional Array

Define or initialize Array values:

- There are **two ways** to define the array values:

1. **During Declaration**: int [] marks={89, 90, 67,35, 99, 80};
2. **Using index :** int marks[]=new int[10]; **// declaration**

   marks[0]=89;

   marks[1]=90;

   ………..

- **Array Length**: To find the array length java provides **inbuilt length variable**

  int len=marks.length; //**assigns the size of marks array in to the variable len**

Single Dimensional Array

Retrieving and Processing Array Elements

- Array elements are accessed using index value. The index starts with 0.

  element=marks[3]; //**retrieve 4th element of array**

  marks[4]= 100; // **update 100 into 5th position**

- Since more number of elements are there in an array, **loops are more convenient way** to process.
- In general, array elements are **accessed and processed** using **for loop** and **for….each loop**.

Single Dimensional Array

```
/**
  * The ArrayDemo1 class implements an application that
  * Illustrate the access array elements
*/
 public class ArrayDemo1 {
     public static void main(String[] args) {
                 int[] marks = new int[3]; ; // create array
                 marks[0] = 89;  //assign values
                 marks[1] = 90;
                 System.out.println("Element at index 0: " + marks[0]); //access elements
                 System.out.println("Element at index 1: " + marks[1]);
                 System.out.println("Element at index 2: " + marks[2]);
     }
}
```

Output:

```
Element at index 0: 89
Element at index 1: 90
Element at index 2: 0
```

Single Dimensional Array

```
/**
  * The CustomerDetail class implements an application that
  * Illustrate the access array elements
*/
 public class CustomerDetail {
public static void main(String[] args) {
              String[] custName = new String[5]; ; // create array
              custName[0] = "Aaron";  //assign values
              custName[1] = "Kavin";
              custName[2] = "Jesicca";
              custName[3] = "Rishabh";
              custName[4] = "Vinitha";
```

Single Dimensional Array

```
                System.out.println("*********CUSTOMER DETAIL********");
                System.out.println(custName[0]); //access elements
                System.out.println(custName[1]);
                System.out.println(custName[2]);
                System.out.println(custName[3]);
                System.out.println(custName[4]);
        }
}
```

Output:

```
*********CUSTOMER DETAIL********
Aaron
Kavin
Jesicca
Rishabh
Vinitha
```

Single Dimensional Array

```
/**
  * The ArrayDemoForEach class implements an application that
  * Illustrate the access array elements using for-each statements
*/
Class ArrayDemoForEach {
     public static void main(String[] args)  {
                         int[] marks = {56, 84, 52, 90, 100};
                         System.out.println("Using for each Loop:");
                         for(int  i:marks) {
                                System.out.println(i);
                 }
        }
}
```

Output:

```
Using for each Loop:
56
84
52
90
100
```

Single Dimensional Array

```
/**
  * The CustomerDetailForEach class implements an application that
  * Illustrate the access array elements using for-each statements
*/
public class CustomerDetailForEach {
      public static void main(String[] args) {
            String[] custName = new String[5]; ; // create array
            custName[0] = "Aaron";  //assign values
            custName[1] = "Kavin";
            custName[2] = "Jesicca";
            custName[3] = "Rishabh";
            custName[4] = "Vinitha";
```

Single Dimensional Array

```
      System.out.println("************CUSTOMER DETAIL***********");
      for(String name : custName) {
          System.out.println(name);
      }
  }
}
```

Output:

```
************CUSTOMER DETAIL***********
Aaron
Kavin
Jesicca
Rishabh
Vinitha
```

Multi Dimensional Array

- In general, **more than one dimension** refer to the multi dimensional array. Its is **array of arrays**. The retrieve and processing like single dimensional array.
- **Declaration and Instantiation**
- **Two-Dimensional Array:** In the **form of matrix** which represents collection of **rows and columns.**

Syntax:

```
DataType[][] ArrayName = new int[RowSize][ColumnSize];

int bookCount[][] = new int[3][3];
```

- **Three Dimensional Array:** In the **form of table** and each table contains number of rows and columns.

Syntax:

```
DataType[][][] ArrayName = new int[TableSize][RowSize][ColumnSize];
int bookCount[][][]= new int[3][3][3];
```

Multi Dimensional Array

Need for Multi-dimensional Array:

- Data is stored in the form of **table or matrix form**. It is used to represents the data **in different dimension like row and column wise**.
- It is used to represents **Graph and database** like data structure.
- Specifically used to **find minimum spanning tree** and **connectivity checking between nodes.**

Multi Dimensional Array

```
/**
  * The ArrayDemo3 class implements an application that illustrate the access of multidimensional
    array elements */
Class ArrayDemo3{
    public static void main(String[] args){
                        int [][] x = new int[][] {{1,2},{3,4},{5,6}}; // initialize values
              for(int i=0; i < x.length; i++) {
                        // print array elements
                               for (int j=0; j < x[i].length; j++) {
                                          System.out.print(x[i][j]);
                              }
                            System.out.println();
                        }
      }
 }
```

Output:

```
1 2
3 4
5 6
```

Multi Dimensional Array

```
/**
  * The MovieSeat class implements an application that illustrate the access of multidimensional array elements */
public class MovieSeat {
public static void main(String[] args){
        String [][] seatType = new String[][] {{"B","B","A","A","A"},{"A","A","A","B","B"},{"A","B","B","B","B"},{"B","A","A","B","A"}};
        int vipcount = 0, premiumcount = 0, regularcount = 0, viptotal = 5, premiumtotal = 10, regulartotal = 5;
        System.out.println("--MOVIE SEAT ARRANGEMENT--");
        for(int i=0; i < seatType.length; i++) {
        if (i==0)
        System.out.println("--------VIP SEATS--------");
        else if(i==1)
        System.out.println("------PREMIUM SEATS------");
        else if(i==3)
        System.out.println("------REGULAR SEATS------");
```

Multi Dimensional Array

```
    for (int j=0; j < seatType[i].length; j++) {
      System.out.print(" "+seatType[i][j]+"   ");
      if(i==0 && seatType[i][j].equalsIgnoreCase("B"))
              vipcount++;
      else if(i>0 && i<3 && seatType[i][j].equalsIgnoreCase("B"))
               premiumcount++;
      else if(i==3 && seatType[i][j].equalsIgnoreCase("B"))
               regularcount++;
      }
      System.out.println();
}
System.out.println("----SEAT BOOKED DETAIL----");
System.out.println("--------VIP SEATS--------");
System.out.println("BOOKED : "+vipcount+" AVAILABLE : "+(viptotal-vipcount)+" TOTAL : "+viptotal);
```

Multi Dimensional Array

```
         System.out.println("-----PREMIUM SEATS-----");
        System.out.println("BOOKED : "+premiumcount+" AVAILABLE : "+(premiumtotal-premiumcount)+" TOTAL : "+premiumtotal);
        System.out.println("-----REGULAR SEATS-----");
        System.out.println("BOOKED : "+regularcount+" AVAILABLE : "+(regulartotal-regularcount)+" TOTAL : "+regulartotal);
    }
}
```

Output:

```
--MOVIE SEAT ARRANGEMENT--
--------VIP SEATS--------
 B   B   A   A   A
------PREMIUM SEATS------
 A   A   A   B   B
 A   B   B   B   B
------REGULAR SEATS------
 B   A   A   B   A
----SEAT BOOKED DETAIL----
--------VIP SEATS--------
BOOKED : 2 AVAILABLE : 3 TOTAL : 5
-----PREMIUM SEATS-----
BOOKED : 6 AVAILABLE : 4 TOTAL : 10
-----REGULAR SEATS-----
BOOKED : 2 AVAILABLE : 3 TOTAL : 5
```

Multi Dimensional Array : Jagged Array

- Java supports jagged array. It is **array of arrays** with **different number of columns**.
- **Declaration and Instantiation**
- **Jagged Array:** In the **form of matrix** which represents **fixed number of rows and different size columns**.

Syntax:

```
DataType[][] Array_Name = new int[Row_Size][];
int bookNo[][] = new int[3][]; //variable size column
//adding column
bookNo[0] = new int[3];  //0th row contains 3 columns
bookNo[1] = new int[5];  //1st  row contains 5 columns
bookNo[2] = new int[1];  //2nd  row contains 1 column
```

Multi Dimensional Array : Jagged Array

Need for Jagged Array:

- It improves the performance when working with multi-dimensional arrays.
- In a jagged array, which is an array of arrays, each inner array can be of a different size. It helps the efficient **memory management.**
- **For example**:
- Course registration count based on different category.
- Ticket reservation details based on different category. Etc.

Multi Dimensional Array : Jagged Array

```
/**
  * The JaggedArray class implements an application that
  * Illustrate the jagged array
*/
public class JaggedArray {
        public static void main(String[] args) {
                int bookNo[][] = new int[3][];
                bookNo[0] = new int[] {1,2,3};
                bookNo[1] = new int[] {4,5};
                bookNo[2] = new int[] {6,7,8,9,10};
                System.out.println("Two Dimensional Jagged Array");
```

Multi Dimensional Array : Jagged Array

```
            for(int i=0;i<bookNo.length;i++) {
                for(int j=0;j<bookNo[i].length;j++) {
                    System.out.println(bookNo[i][j]+" ");
                }
                System.out.println();
            }
    }
}
```

Output:

```
Two Dimensional Jagged Array
1
2
3

4
5

6
7
8
9
10
```

Multi Dimensional Array

```
/**
  * The MovieSeat class implements an application that illustrate the access of multidimensional array elements */
public class MovieSeatJaggedArray {
public static void main(String[] args){
        String [][] seatType = new String[][] {{"B","A","A"},{"A","A","A","B","B"},{"A","B","B","B","B"},{"B","A","A","A"}};
        int vipcount = 0, premiumcount = 0, regularcount = 0, viptotal = 5, premiumtotal = 10, regulartotal = 5;
        System.out.println("--MOVIE SEAT ARRANGEMENT--");
        for(int i=0; i < seatType.length; i++) {
        if (i==0)
                 System.out.println("--------VIP SEATS--------");
        else if(i==1)
                System.out.println("------PREMIUM SEATS------");
        else if(i==3)
                System.out.println("------REGULAR SEATS------");
```

Multi Dimensional Array

```
    for (int j=0; j < seatType[i].length; j++) {
       System.out.print(" "+seatType[i][j]+"   ");
       if(i==0 && seatType[i][j].equalsIgnoreCase("B"))
                vipcount++;
       else if(i>0 && i<3 && seatType[i][j].equalsIgnoreCase("B"))
                 premiumcount++;
       else if(i==3 && seatType[i][j].equalsIgnoreCase("B"))
                 regularcount++;
      }
      System.out.println();
}
System.out.println("----SEAT BOOKED DETAIL----");
System.out.println("--------VIP SEATS--------");
System.out.println("BOOKED : "+vipcount+" AVAILABLE : "+(viptotal-vipcount)+" TOTAL : "+viptotal);
```

Multi Dimensional Array

```
         System.out.println("-----PREMIUM SEATS-----");
        System.out.println("BOOKED : "+premiumcount+" AVAILABLE : "+(premiumtotal-
premiumcount)+" TOTAL : "+premiumtotal);
        System.out.println("-----REGULAR SEATS-----");
        System.out.println("BOOKED : "+regularcount+" AVAILABLE : "+(regulartotal-
regularcount)+" TOTAL : "+regulartotal);
    }
}
```

Output:

```
--MOVIE SEAT ARRANGEMENT--
--------VIP SEATS--------
 B   A   A
------PREMIUM SEATS------
 A   A   A   B   B
 A   B   B   B   B
------REGULAR SEATS------
 B   A   A   A
----SEAT BOOKED DETAIL----
--------VIP SEATS--------
BOOKED : 1 AVAILABLE : 2 TOTAL : 3
-----PREMIUM SEATS-----
BOOKED : 6 AVAILABLE : 4 TOTAL : 10
-----REGULAR SEATS-----
BOOKED : 1 AVAILABLE : 3 TOTAL : 4
```

Quiz

1. Which of the following is FALSE about Java array?

a) A java array is always an object

b) Length of array can be changed after the creation of array

c) Arrays in Java are always allocated on heap

d) Array is the example for Non-Primitive type

b) Length of array can be changed after the creation of array

Quiz

2. In java array supports,

a) Primitive type only

b) Object type only

c) Both

d) None of these above

c) Both

Quiz

3. Java supports variable size column in multidimensional array

a) Yes

b) No

a) Yes

Functions

Contents

- Introduction
- Function elements
- Recursive functions
- Quiz

Introduction

- A **function/Method** is a **block of code** which perform **specific task** and the task is executed when the function is invoked or called.
- Primary **uses of functions**:
  - It allows code r**eusability** (define once and use multiple times)
  - You can **break a complex program** into smaller chunks of code
  - **Reducing duplication** of code
  - Make program shorter and **increases code readability**

Types

- There are **two types** of functions:
  - **Built-in function:** Java has several functions that are **readily available for use**. In Java, every built-in function should be **part of some class**.
  - **User defined function:** Function created **by user** based on the **need of application**.

- With methods (functions), there are **2 major points** of view
  - **Builder of the method** - responsible for creating the *declaration* and the *definition* of the method (i.e. how it works)
  - **Caller** - somebody (i.e. some portion of code) that *uses* the method to perform a task

Function Elements

- There are **two** important **elements** of function**:**

1. **Function Definition**

   – It define the **operation of a function.** It consists of the **function signature followed by the function body**. The function prototype includes **function name, parameter list and return type**.

2. **Function Call**

   – It means **call or invoke** the specific function. When a function is invoked, the program **control jumps** to the **called function** to execute the statements that are in the part of that function. Once the called function is executed the program **control passes back to the calling function.**

Function Elements

- **Function Definition**

Syntax:

```
modifier returnType  methodName(parameterlists)  // Function signature
{
      //Function Body
}
```

  - **Modifiers / Specifiers:** It defines the **visibility of the method** i.e. from where it can be accessible in the application.

Function Elements

- In Java, there **4 type of the access specifiers**.
  - **public:** accessible in all class in your application.
  - **protected:** accessible within the class in which it is defined and, in its subclass,(es).
  - **private:** accessible only within the class in which it is defined.
  - **default (declared/defined without using any modifier) :** accessible within same class and package within which its class is defined.

Function Elements

- **Function Definition**
  - **The return type :** The data type of the value returned by the method or void if does not return a value.
  - **Method Name :** Name of the function should specify using valid identifier
  - **Java Naming Convention**: It is a **single word** that should be a verb in lowercase or **multi-word**, that begins with a verb in lowercase followed by adjective or noun. After the first word, **first letter of each word should be capitalized. Example**: computeAddition, setName
  - **Parameter list :** Comma separated list of the **input parameters** are defined, preceded with their data type, within the enclosed parenthesis. If there are no parameters, you must use **empty parentheses ().**
  - **Method body :** It is enclosed between braces. The code specifies the **task to be done**.

Function Elements

- **Function Call**
  - **Static method** is invoked by the **class name** or **using object**. (**Refer Example**)
  - **Example**: className.methodName(argumentList)
  - **Non static method** is invoked by the **instance/object** of the class (**We Will discuss later**)
  - **Example:** objectName.methodName(argumentList)

Return Values

- **Function return types**: Function return values of **any valid types**. It must **return data** that **matches their return type.**

- **Example:**

```
public int addTwoInt(int a, int b){
            return a+b;   // return integer
 }
```

```
 //void method return nothing
public void printName(String name){
   System.out.println("Hello World!!!"); // return
void
 }
```

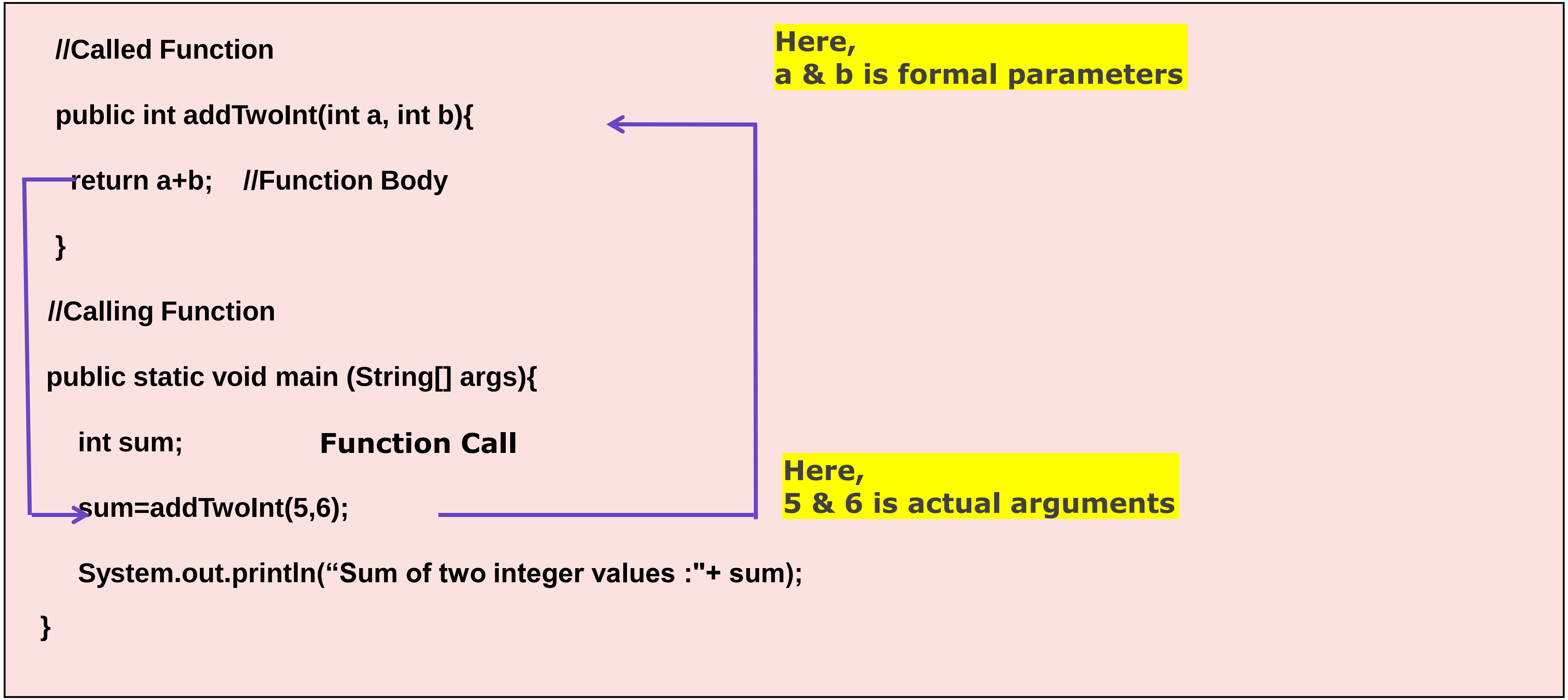
Arguments and Parameters

- **Arguments and Parameters:** The terms parameter and argument can be used for the same thing information that are passed into a function.
    - **Arguments** –An argument is a value that is passed during a **function call or calling function**.
    - **Parameters** – Parameters used in the function definition or called function. A parameter is a variable defined in the **function definition or calling function**.

Note:

- During program execution, the **values in the actual arguments** is **assigned to the formal parameter**.

Function Elements



```
 //Called Function

 public int addTwoInt(int a, int b){

  return a+b;    //Function Body

 }

//Calling Function

public static void main (String[] args){

   int sum;

   sum=addTwoInt(5,6);

   System.out.println(“Sum of two integer values :"+ sum);

}
```

Function Call

Here,
a & b is formal parameters

Here,
5 & 6 is actual arguments

Function Elements

```
/**
  * The MethodDeclareDemo class implements an application that
  * Illustrate the user defined methods(static) */
class MethodDeclareDemo{
                //User Defined Method
                public int static addTwoInt(int a, int b){  //a, b is parameters
                               return a+b;
                 }
                public static void main (String[] args){
                       int sum;
                       sum=addTwoInt(5,6);  //5, 6 is arguments
                        System.out.println(“Sum of two integer values :"+ sum);
                 }
        }
```

Output:

```
Sum of two integer
values :11
```

Function Elements

```
/**
  * The FunctionDeclare class implements an application that display the Movie details
  * using the user defined methods(static) */
public class FunctionDeclare {
     static void getMovieDetail(String moviename,String moviedescription,int movieduration,String movielanguage,
     String moviereleasedate,String moviecountry,String moviegenre) {
                    System.out.println("Movie Title : "+moviename);
                    System.out.println("Movie Description : "+moviedescription);
                    System.out.println("Movie Duration : "+movieduration);
                    System.out.println("Movie Language : "+movielanguage);
                    System.out.println("Movie Release Date : "+moviereleasedate);
                    System.out.println("Movie Country : "+moviecountry);
                    System.out.println("Movie Genre : "+moviegenre);
     }
```

Function Elements

```
    public static void main(String[] args) {
         String moviename = "AAA";
         String moviedescription = "Dramaof1945";
         int movieduration = 3;
         String movielanguage = "English";
         String moviereleasedate = "25/03/2022";
         String moviecountry = "XYZ";
         String moviegenre = "THRILLER";
        System.out.println("-------Movie Detail-------");
        getMovieDetail(moviename, moviedescription, movieduration, movielanguage, moviereleasedate,
moviecountry, moviegenre);
        System.out.println("---------------------------");
    }   }
```

Function Elements

Output:

```
-------Movie Detail-------
Movie Title : AAA
Movie Description : Dramaof1945
Movie Duration : 3
Movie Language : English
Movie Release Date : 25/03/2022
Movie Country : XYZ
Movie Genre : THRILLER
---------------------------
```

Recursive Functions

- Recursive functions are **functions that call themselves** to **solve smaller instances of the same problem**.
- They are composed of **two main parts**:
- **Base Case:**
  - This is the **condition under which the recursive function stops calling itself**. The base case prevents infinite recursion and typically handles the simplest, smallest instance of the problem.
- **Recursive Case:**
  - This part of the function calls itself with a smaller or simpler input. The **recursive calls continue until the base case is reached.**

Recursive Functions: Algorithm

- **Step1 - Define a base case:**
  - Identify the simplest case for which the solution is known or trivial.
  - This is the stopping condition for the recursion, as it prevents the function from infinitely calling itself.
- **Step2 - Define a recursive case:**
  - Define the problem in terms of smaller subproblems.
  - Break the problem down into smaller versions of itself, and call the function recursively to solve each subproblem.
- **Step3 - Ensure the recursion terminates:**
  - Make sure that the recursive function eventually reaches the base case, and does not enter an infinite loop.
- **Step4 - Combine the solutions:**
  - Combine the solutions of the subproblems to solve the original problem

Recursive Functions : Example - Factorial Calculation

- The factorial of a non-negative integer n (denoted as n!) is the **product of all positive integers** less than or equal to n.
- It can be **defined recursively** as:

  - Base Case: 0!=1

  - Recursive Case: $n! = n \times (n-1)!$

Recursive Functions : Example - Factorial Calculation

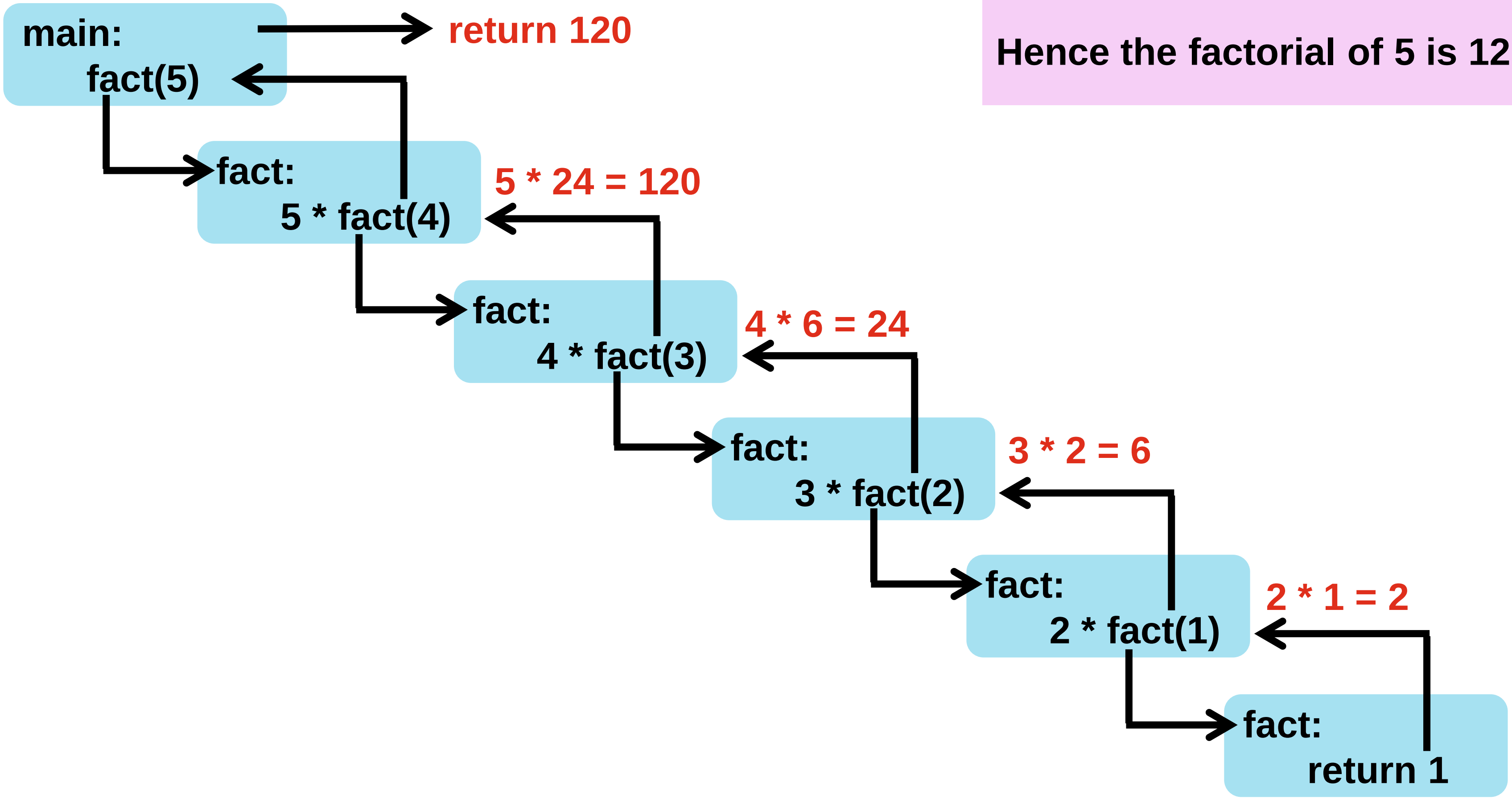
```
//To find factorial using recursion
// Function to calculate factorial
public class FactorialRecursive {
static int fact(int n) {
    if (n == 0 || n == 1) { // Base case
        return 1;
    } else {
        return n * fact(n - 1); // Recursive case
    }
}
```

Output:

```
Factorial of 5 is 120.
```

```
 public static void main(String[] args) {
    int number = 5;
   if (number < 0) {                   // Check for negative input
        System.out.println("Factorial is not defined for negative
numbers.\n");
    } else {
        // Calculate and display factorial
       System.out.println("Factorial of " + number + " is " +
       fact(number));
    }
}
}
```

Recursive Functions : Example - Factorial Calculation



Hence the factorial of 5 is 120

Recursive Functions : Example - Factorial Calculation

- If the base case is not reached or not defined, then the stack overflow problem may arise.

Example

```
int fact(int n) {
    if (n ==100) {                      // // Wrong Base case may cause Stack Overflow
        return 1;
    } else {
        return n * fact(n - 1);      // Recursive case
    }
}
```

- If fact(10) is called, it will call fact(9), fact(8), fact(7), and so on but the number will never reach 100.
- So, the base case is not reached.
- If the **memory is exhausted by these functions on the stack**, it will cause a stack overflow error.

Recursive Functions : Example - Fibonacci Series

- The Fibonacci sequence is a series of numbers where each number is the sum of the two preceding ones. It can be defined recursively as:
  - **Base Case:** Fib(0)=0, Fib(1)=1
  - **Recursive Case**: Fib(n)=Fib(n−1)+Fib(n−2)

Recursive Functions : Example - Fibonacci Series

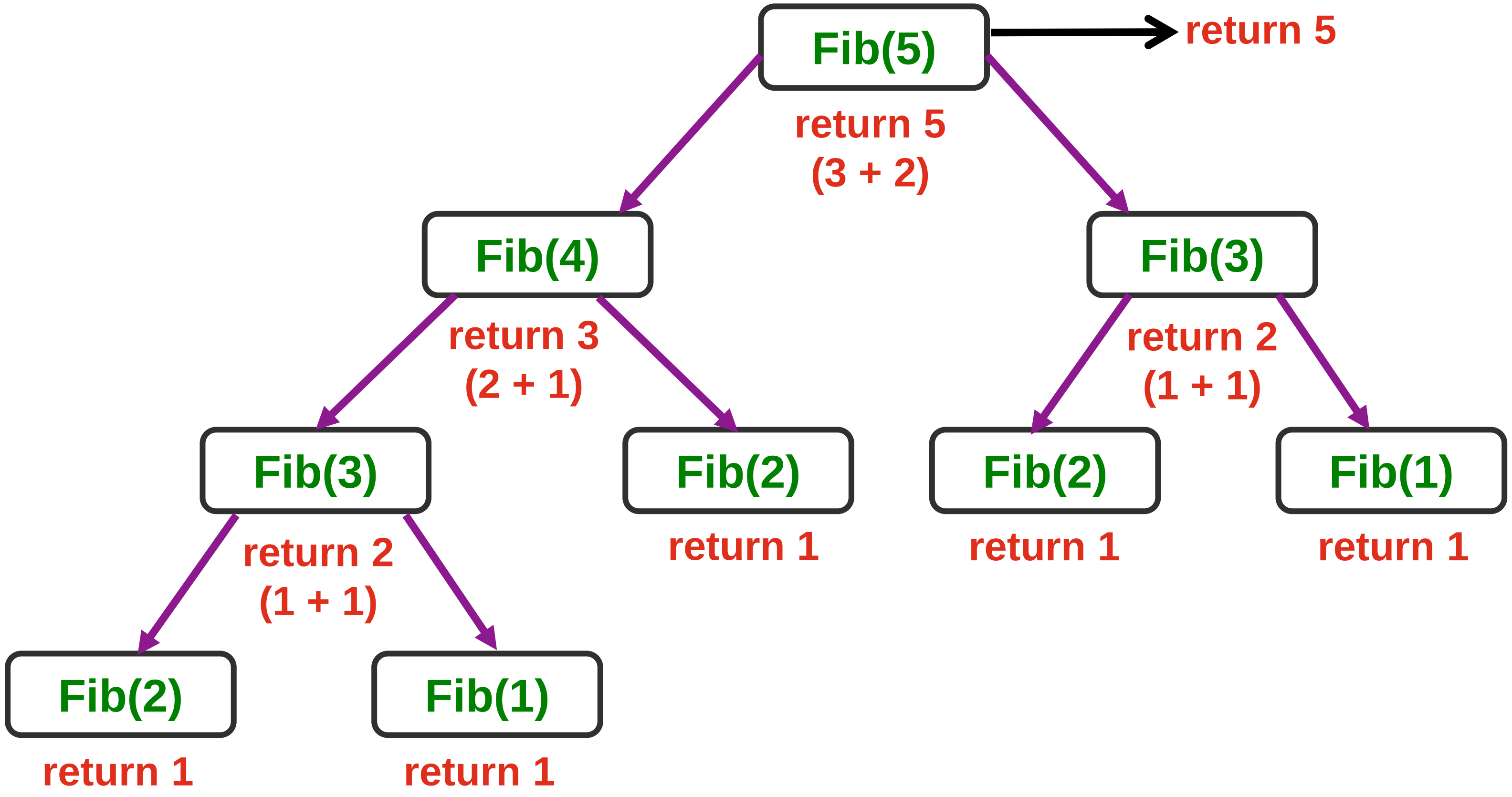
```
/*Recursive function to calculate the nth
Fibonacci number*/
public class FibonacciRecursive {
static int Fib(int n) {
    if (n == 0) // Base case
        return 0;
    else if (n == 1) // Base case
        return 1;
    else
        return Fib(n - 1) + Fib(n - 2);
        // Recursive case
}
```

```
public static void main(String[] args) {
    int terms, i;
    terms = 10;
   System.out.print("Fibonacci Series: ");
    for (i = 0; i < terms; i++) {
       // Call the Fib function for each term
       System.out.print(Fib(i)+ "  ");
     }
    System.out.println("\n");
}
}
```

Output:

```
Fibonacci Series: 0 1 1 2 3 5 8 13 21 34
```

Example: Fibonacci Series



Advantages and Disadvantages

Advantages of Recursion

- **Simplifies Code:** Recursion can make code more concise and easier to read.
- **Natural Fit:** Some problems are naturally recursive, such as tree traversal and the Tower of Hanoi.

Disadvantages of Recursion

- **Memory Usage:** Recursive calls use the call stack, which can lead to high memory usage and stack overflow if not managed properly.
- **Performance:** Recursion can be less efficient than iterative solutions due to the overhead of repeated function calls.

Recursion is a powerful tool in algorithm design, providing elegant solutions to complex problems, especially those that can be broken down into smaller, similar sub-problems.

Quiz

1. Java method signature is a combination of ___.

a) returnType

b) methodName

c) ParameterList

d) All the above

d) All the above

Quiz

2. What the naming convention should be for Methods in Java?

a) It should start with lowercase letter.

b) If name contains many words then first word's letter start with lowercase only and other words will start with uppercase

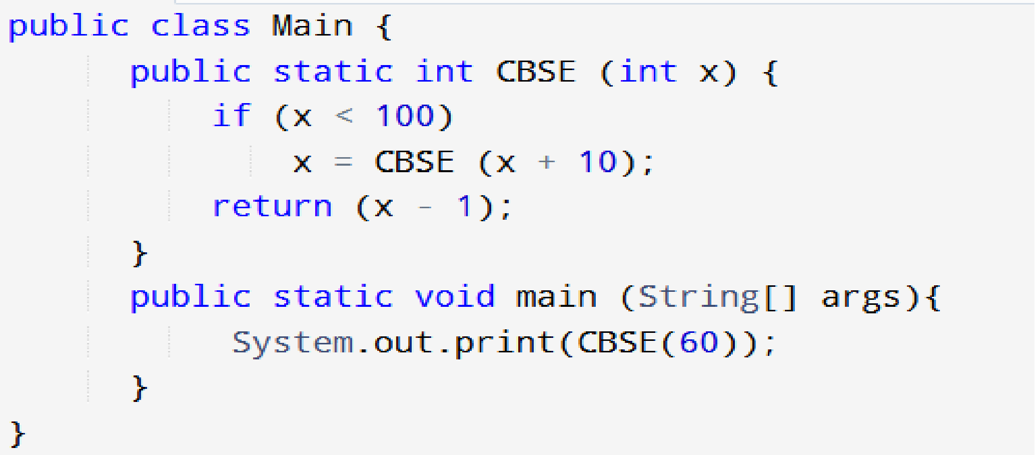
c) It should be a verb such as show(), count(), describe()

d) All the above

d) All the above

Quiz

3. Consider the following program :



```
public class Main {
        public static int CBSE (int x) {
            if (x < 100)
                x = CBSE (x + 10);
            return (x - 1);
        }
        public static void main (String[] args){
             System.out.print(CBSE(60));
        }
}
```

What does this program print ?

a) 59

b) 95

c) 69

d) 99

b) 95

Quiz

4. Which of the following main method signatures will cause a compilation error?

a) public static void main(String... args)

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

c) public void main(String[] args)

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

c) public void main(String[] args)

Quiz

5. A function which calls itself is called a ___ function.

a. Self Function

b. Auto Function

c. Recursive Function

d. Static Function

c. Recursive Function

> Proper Preparation
>
> Prevents
>
> Poor Performance

- Charlie Batch

THANK YOU

SmartCliff
Career Mobility Solutions