



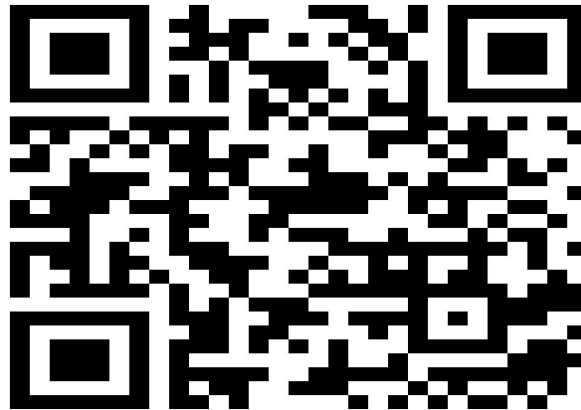
<Codemithra /><sup>TM</sup>

Explore | Expand | Enrich

# TEST TIME ON STRUCTURE

**URL:** <https://forms.gle/iHwKZdaoH2Smz6sP8>

**QR CODE:**





<Codemithra /><sup>TM</sup>



Explore | Expand | Enrich

# Data types





<Codemithra /><sup>TM</sup>



Explore | Expand | Enrich

## What you'll learn

- Need of data types
- Introduction to data types
- Classification of data types
- Size and range of primitive data types
- Non primitive data types





<Codemithra /><sup>TM</sup>



Explore | Expand | Enrich

## What you'll learn

- Difference between primitive and non primitive
- Type casting
- Variables, Methods
- Types of variables & methods
- Ways to access the variables & methods
- Code snippets
- Real life scenario





## Need of data types

- Java - strongly typed language
- Operations are type-checked by compiler
- To achieve type compatibility
- To allocate the respective memory space
- To be more robust

<Codemithra /><sup>TM</sup>



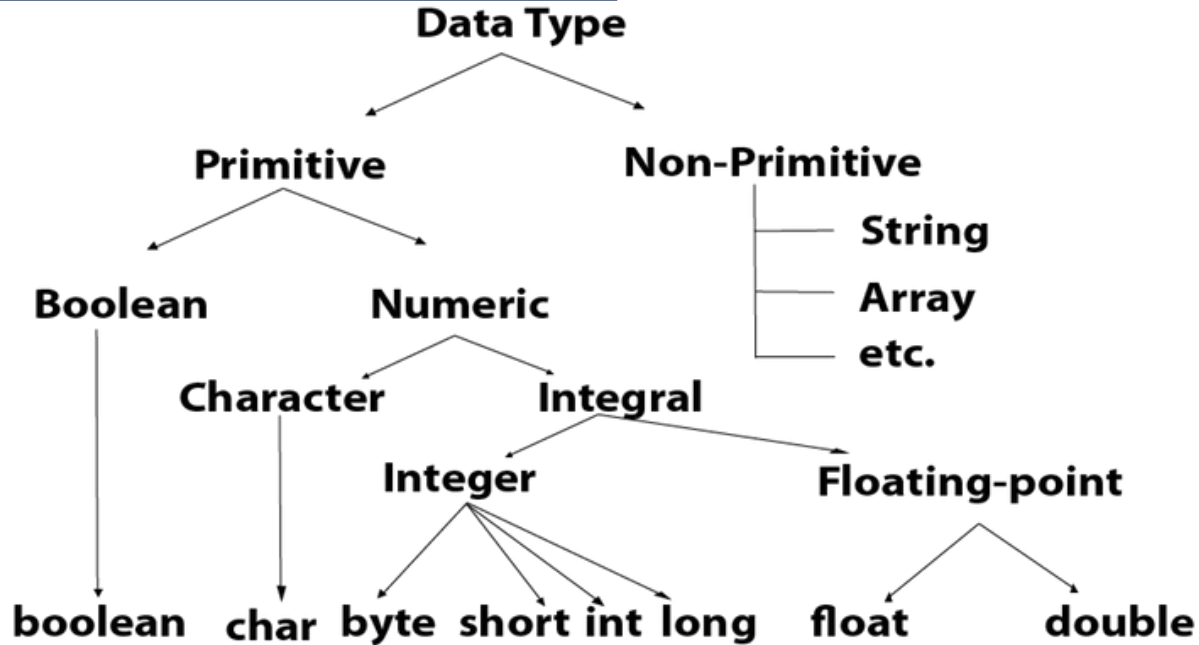
## Introduction to data types

Specifies the type and size of the values to be stored in a variable / Identifier

Two broader classification for datatypes

- **Primitive data types** - includes byte, short, int, long, float, double, boolean and char
- **Non-primitive data types** - such as String, Arrays and Classes

## Classification of data types





## Size and range of primitive data types

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter



<Codemithra /><sup>TM</sup>



Explore | Expand | Enrich

## Code snippet 1

```
public class Main {  
    public static void main(String[] args) {  
        byte Num = 127;  
        System.out.println(Num);  
    }  
}
```





<Codemithra /><sup>TM</sup>



Explore | Expand | Enrich

## Code snippet 2

```
public class Main {  
    public static void main(String[] args) {  
        byte Num = 128;  
        System.out.println(Num);  
    }  
}
```





<Codemithra /><sup>TM</sup>



Explore | Expand | Enrich

## Code snippet 3

```
public class Main {  
    public static void main(String[] args) {  
        short Num = 5000;  
        System.out.println(Num);  
    }  
}
```





## Code snippet 4

<Codemithra /><sup>TM</sup>




Explore | Expand | Enrich

```
public class Main {  
    public static void main(String[] args) {  
        short Num = 509864700;  
        System.out.println(Num);  
    }  
}
```



## Code snippet 5

```
public class Main {  
    public static void main(String[] args) {  
        int Num = 5986745;  
        System.out.println(Num);  
    }  
}
```





<Codemithra /><sup>TM</sup>



Explore | Expand | Enrich

## Code snippet 6

```
public class Main {  
    public static void main(String[] args) {  
        long Num = 500000001;  
        System.out.println(Num);  
    }  
}
```





## Code snippet 7

<Codemithra /><sup>TM</sup>



Explore | Expand | Enrich


```
public class Main {  
    public static void main(String[] args) {  
        long Num = 50000000;  
        System.out.println(Num);  
    }  
}
```






## Code snippet 8

```
public class Main {  
    public static void main(String[] args) {  
        float Num = 5.75f;  
        System.out.println(Num);  
    }  
}
```



## Code snippet 9

```
public class Main {  
    public static void main(String[] args) {  
        double Num = 19.99d;  
        System.out.println(Num);  
    }  
}
```





## Code snippet 10

A floating point number can also be a scientific number with an "e" to indicate the power of 10

```
public class Main {  
    public static void main(String[] args) {  
        float f1  = 35e3f;  
        double d1 = 12E4d;  
        System.out.println(f1);  
        System.out.println(d1);  
    }  
}
```





<Codemithra /><sup>TM</sup>



Explore | Expand | Enrich

## Code snippet 11

```
public class Main {  
    public static void main(String[] args) {  
        boolean isJavaFun = true;  
        boolean isFishTasty = false;  
        System.out.println(isJavaFun);  
        System.out.println(isFishTasty);  
    }  
}
```





## Code snippet 12

The char data type is used to store a single character. The character must be surrounded

```
public class Main {  
    public static void main(String[] args) {  
        char Letter = 'B';  
        System.out.println(Letter);  
    }  
}
```



## Code snippet 13

```
public class Main {  
    public static void main(String[] args) {  
        char a = 65, b = 66, c = 67;  
        System.out.println(a);  
        System.out.println(b);  
        System.out.println(c);  
        System.out.println(a+b+c);  
    }  
}
```

<Codemithra /><sup>TM</sup>




Explore | Expand | Enrich

## Code snippet 14

The String data type is used to store a sequence of characters (text). String values must be surrounded by double quotes.

```
public class MyClass {  
    public static void main(String[] args) {  
        String greeting = "Hello World";  
        System.out.println(greeting);  
    }  
}
```

## Non primitive data types

- Non-primitive data types are called **reference types** because they refer to objects.
  - Some of the reference types are pre defined such as String, Array.
  - `String s = "Hello world";`
  - String is a non primitive data type (pre defined class).
  - User defined non primitive data types can also be created with classes & Interfaces.
- 



## Differences

- Primitive types are predefined (already defined) in Java. Non-primitive types can be created by the programmers.
- Non-primitive types can be used to call methods to perform certain operations, while primitive types cannot.
- A primitive type has always a value, while non-primitive types can be null.
- A primitive type starts with a lowercase letter, while non-primitive types starts with an uppercase letter.
- The size of a primitive type depends on the data type, while non-primitive types have all the same size.

# Typecasting

- Changing one type of information to the other type.
- It can be implied for both primitive and as well as non primitive data types.

Two types:

- **Widening Casting (automatically)** - converting a smaller type to a larger type size  
byte -> short -> char -> int -> long -> float -> double
- **Narrowing Casting (manually)** - converting a larger type to a smaller size type  
double -> float -> long -> int -> char -> short -> byte

## When to go for typecasting?

Consider

```
int i = 10;           // type matching statement
String s = "Hello world"; //type matching statement
char c = 'a';         //type matching statement
double d = 87.97545;  //type matching statement
```

But,

```
double d1= 50;        //type mismatching statement
int i1 = 89.98;       //type mismatching statement
```

Thus, whenever there are type mismatching statements, perform typecasting to change the effective data type.



## WIDENING CASTING

Widening casting is done automatically when passing a smaller size type to a larger size type

### Example

```
public class Main{  
    public static void main(String[] args) {  
        int myInt = 9;  
        double myDouble = myInt;  
        System.out.println(myDouble);  
        System.out.println(myInt);  
    }  
}
```

## NARROWING CASTING

Narrowing casting is done automatically when passing a larger size type to a smaller size type

### Example

```
public class Main{  
    public static void main(String[] args) {  
        double myDouble = 9.78;  
        int myInt = (int) myDouble;  
        System.out.println(myDouble);  
        System.out.println(myInt);  
    }  
}
```

## Code snippet

```
public class Test {  
    public static void main(String[] args) {  
        int i = 98756;  
        int i1 = 9.0;  
        double d = i;  
        System.out.println(i+" "+d);  
        double d1 = 56.897d;  
        int i2 = (int) d1;  
        System.out.println(d1+" "+i2);  
        int a = 9;  
        float f = a/2; // 9/2 = 4.0  
        System.out.println(f);  
        char c = 'A';//65  
        int n = c+1;//66  
        char c1 = (char) n;  
        System.out.println(c1); }}
```

## Identifiers

Containers to store the values

Variable type and value must match

`int a = 100;` // int is the data type, a is the variable name, 100 is the value

1. begin your variable names with a letter, not "\$" or "\_".
2. the dollar sign character, by convention, is never used at all
3. variable's name with "\_", this practice is discouraged. White space is not permitted.
4. use full words instead of cryptic abbreviations
5. must not be a **keyword or reserved word**.
6. If the name you choose consists of only one word, spell that word in all lowercase letters. If it consists of more than one word, capitalize the first letter of each subsequent word

`String name = "John";`

`String companyName = "Ethnus";`




## Types

### 1. Instance Variables (Non-Static Fields) :

- objects store their individual states in "non-static fields", that is, fields declared without the static keyword
- Non-static fields are also known as *instance variables*

### 1. Class Variables (Static Fields) :

- A class variable is any field declared with the static modifier
  - The compiler that there is exactly one copy of this variable in existence, regardless of how many times the class has been instantiated.
- 

# Types

## 1. Local Variables :

- method will often store its temporary state in local variables.
- The syntax for declaring a local variable is similar to declaring a field (for example, `int count = 0;`).
- There is no special keyword designating a variable as local.

## 1. Parameters

- The signature for the main method is `public static void main(String[] args)`.
- The `args` variable is the parameter to this method.
- The parameters are always classified as "variables" not "fields".

## Declaring Many Variable

```
public class Main {  
    public static void main(String[] args) {  
        int x = 5, y = 10, z = 15;  
        System.out.println(x + y + z);  
    }  
}
```



<Codemithra /><sup>TM</sup>



Explore | Expand | Enrich

## Java Method

- A **method** is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a method.
- Methods are used to perform certain actions, and they are also known as **functions**.

Why use methods?

To reuse code: define the code once, and use it many times.



## Java Methods

- A method must be declared within a class.
- It is defined with the name of the method, followed by parentheses ().

### Example

```
public class MyClass {  
    void myMethod() {  
        // code to be executed  
    }  
}
```

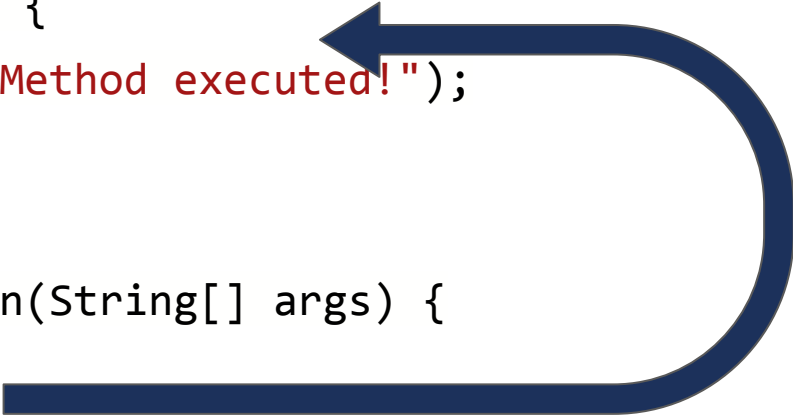
<Codemithra />™



Explore | Expand | Enrich

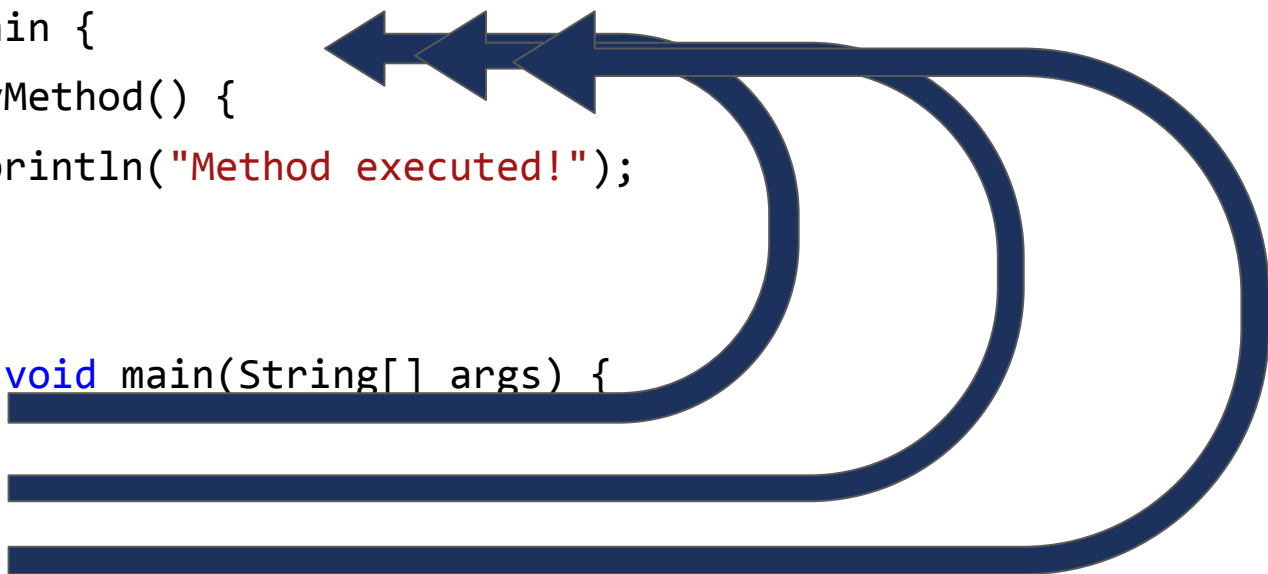
## Call a Method

```
public class Main {  
    static void myMethod() {  
        System.out.println("Method executed!");  
    }  
  
    public static void main(String[] args) {  
        myMethod();  
    }  
}
```



## Multiple Call

```
public class Main {  
    static void myMethod() {  
        System.out.println("Method executed!");  
    }  
  
    public static void main(String[] args) {  
        myMethod();  
        myMethod();  
        myMethod();  
    }  
}
```





<Codemithra />™



Explore | Expand | Enrich

## Java Method parameter

- Information can be passed to methods as parameter.
- Parameters act as variables inside the method.
- Parameters are specified after the method name, inside the parentheses.
- You can add as many parameters as you want, just separate them with a comma







## Types of methods

Pre-defined methods

Example → `System.out.println();`

User defined methods

Example → `void displayInformation(){ }`

<Codemithra /><sup>TM</sup>



Explore | Expand | Enrich





<Codemithra /><sup>TM</sup>



Explore | Expand | Enrich

## Ways to access static and non-static methods

Static methods → access using the class name

Non static methods → access using the object / reference variable





## Example

```
public class MyClass {  
    static void myMethod(String fname) {  
        System.out.println(fname + " Refsnes");  
    }  
    public static void main(String[] args) {  
        myMethod("Liam");  
        myMethod("Jenny");  
        myMethod("Anja");  
    }  
}
```





<Codemithra /><sup>TM</sup>



Explore | Expand | Enrich

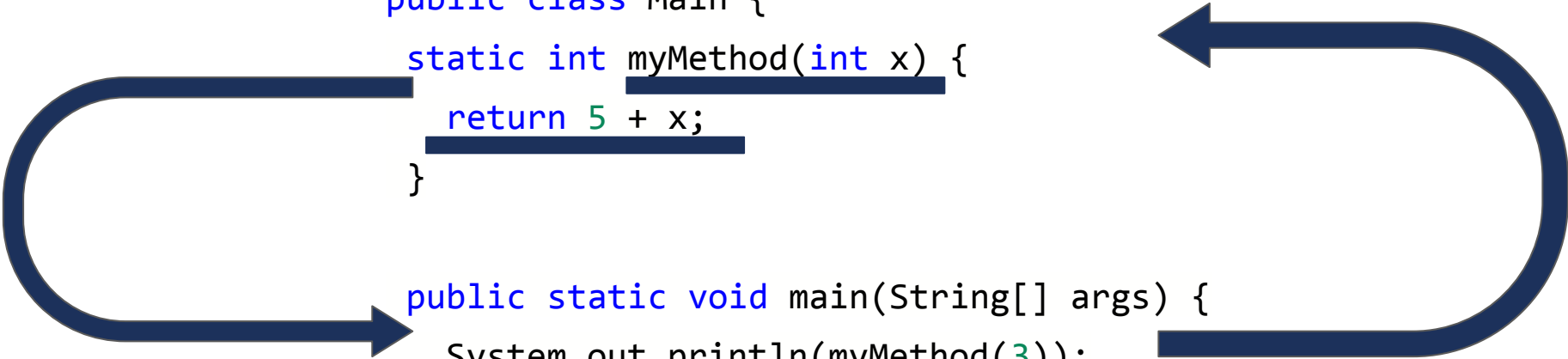
## Multiple Parameter

```
public class Main {  
    static void myMethod(String fname, int age) {  
        System.out.println(fname + " is " + age);  
    }  
    public static void main(String[] args) {  
        myMethod("Liam", 5);  
        myMethod("Jenny", 8);  
        myMethod("Anja", 31);  
    }  
}
```



## Example

```
public class Main {  
    static int myMethod(int x) {  
        return 5 + x;  
    }  
  
    public static void main(String[] args) {  
        System.out.println(myMethod(3));  
    }  
}
```



The diagram illustrates the execution flow of the code. A large blue arrow originates from the `main` method, specifically from the `myMethod(3)` call, and points to the `myMethod` method definition. Another large blue arrow originates from the `myMethod` method definition and points back to the `main` method, indicating the return path.

## Predict the Output

```
public class Main {  
    static int myMethod(int x, int y) {  
        return x + y;  
    }  
  
    public static void main(String[] args) {  
        System.out.println(myMethod(5, 3));  
    }  
}
```



<Codemithra /><sup>TM</sup>



Explore | Expand | Enrich

## Predict the Output

```
public class Main {  
    static int myMethod(int x, int y) {  
        return x + y;  
    }  
  
    public static void main(String[] args) {  
        int z = myMethod(5, 3);  
        System.out.println(z);  
    }  
}
```



## Example

```
public class Main {  
    static int plusMethodInt(int x, int y) {  
        return x + y;  
    }  
    static double plusMethodDouble(double x, double y) {  
        return x + y;  
    }  
    public static void main(String[] args) {  
        int myNum1 = plusMethodInt(8, 5);  
        double myNum2 = plusMethodDouble(4.3, 6.26);  
        System.out.println("int: " + myNum1);  
        System.out.println("double: " + myNum2);  
    }  
}
```



## Real life scenario!



Problem statement :

Create a simple Java project which will contain two source files, one for storing the aadhar card details, another for storing the PAN details. **Choose the appropriate data types** for storing the respective details. Use the **constructor** to initialize the **variables** and a non static **method** to display the information to the user. **Link both PAN and aadhar details and notify the same to the user.**

## Sample output

Hello John!


Please find your addhar details below

```
-----  
Aadhar num      549789761254  
VID              987634527668  
Gender           M  
DOB              20/10/2002  
Address          20A,RR nagar,Bangalore,Karnataka  
-----
```

Your PAN LWST8976 is successfully linked with aadhar 549789761254


## Question: 01

Which of the following is not a valid declaration of an array?

- A. `int [ ] a = new int [3];`
  - B. `int a [ ] [ ] = new int [3] [3]`
  - C. `int [ ] [ ] a = new int [3] [ ];`
  - D. `int [ ] [ ] a = new int [ ] [3];`
- 

## Question: 02

Which of the following line will not compile assuming b1, b2 and b3 are byte variables and J is Int variable?

- A. `b1 = 3;`
  - B. `b3 = b1 * b2;`
  - C. `b3 = 10 * b1;`
  - D. `b2 = (byte) j;`
- 

## Question: 03

```
public class Test {  
    public static void main(String[] args)  
    {  
        System.out.print("Y" + "O");  
        System.out.print('L' + 'O');  
    }  
}
```

- A. YOLO
- B. YO155

## Question: 04

```
public class Test {  
    public static void main(String[] args) {  
        System.out.print("Y" + "O");  
        System.out.print('L');  
        System.out.print('O');  
    }  
}
```

- A. YO7679
- B. YOLO

## Question: 05

```
public class Main
{
    public static void main(String[] args) {
        int[] initializedArrays = new int[]{50, 2, 44};
        System.out.println(initializedArrays[1]);
        initializedArrays[1] = 100;
        System.out.println(initializedArrays[1]);
    }
}
```

- A. 2  
100
- A. 2  
2
- A. Error

## Question: 06

```
public class Test {  
    public static void main(String[] argv){  
        char ch = 'c';  
        int num = 88;  
        ch = num;  
        System.out.println(ch);  
    }  
}
```

- A. Error
- B. 88
- C. C
- D. X





## Question: 07

```
class Main{  
    public static void main(String[] args) {  
        float f = 10.5f;  
        int a = (int) f;  
        System.out.println(f);  
        System.out.println(a);  
    }  
}
```

- A. 10.5  
10
- A. Error
- B. 10  
10

## Question: 08

Integer Data type does not include following primitive data type \_\_\_\_\_.

- A. long
  - B. byte
  - C. short
  - D. double
- 
- 

## Question: 09

```
class mainclass {  
    public static void main(String args[]) {  
        boolean var1 = true;  
        boolean var2 = false;  
        if (var1)  
            System.out.println(var1);  
        else  
            System.out.println(var2);  
    }  
}
```

- A. 0
- B. 1
- C. true
- D. false

## Question: 10

```
class booloperators {  
    public static void main(String args[]) {  
        boolean var1 = true;  
        boolean var2 = false;  
        System.out.println((var1 & var2));  
    }  
}
```

- A. 0
- B. 1
- C. true
- D. false

## Question: 11

```
class asciicodes {  
    public static void main(String args[]) {  
        char var1 = 'A';  
        char var2 = 'a';  
        System.out.println((int) var1 + " " + (int)  
var2);  
    }  
}
```

- A. 162
- B. 65 97
- C. 67 95
- D. 66 98

## Question: 12

```
class A {  
    public static void main(String args[]) {  
        byte b;  
        int i = 258;  
        double d = 325.59;  
        b = (byte) i;  
        System.out.print(b);  
        i = (int) d;  
        System.out.print(i);  
        b = (byte) d;  
        System.out.print(b);}}}
```

- A. 258 325 325
- B. 258 326 326
- C. 2 325 69
- D. Error



<Codemithra />™



Explore | Expand | Enrich

## Question: 13

```
class A {  
    public static void main(String args[]) {  
        int x;  
        x = 10;  
        if (x == 10) {  
            int y = 20;  
            System.out.print("x and y: " + x + " " + y);  
            y = x * 2;  
        }  
        y = 100;  
        System.out.print("x and y: " + x + " " + y);  
    }  
}
```

- A. 10 20 10 100
- B. 10 20 10 20
- C. 10 20 10 10
- D. Error



## Question: 14

```
public class Test {  
    static void test(float x) {  
        System.out.print("float");  
    }  
    static void test(double x) {  
        System.out.print("double");  
    }  
    public static void main(String[] args) {  
        test(99.9);  
    }  
}
```

- A. float
- B. double
- C. Compilation Error
- D. Exception is thrown at runtime





<Codemithra /><sup>TM</sup>



Explore | Expand | Enrich

Question: 15

```
public class Test {  
    public static void main(String[] args) {  
        int i = 010;  
        int j = 07;  
        System.out.println(i);  
        System.out.println(j);  
    }  
}
```

- A. 8 7
- B. 10 7
- C. Compilation fails with an error at line 3
- D. Compilation fails with an error at line 5
- E. None of these



## Question: 16

```
public class MyClass {  
    public static void main(String[] args) {  
        int a = 10;  
        System.out.println(++a++);  
    }  
}
```

- A. 10
- B. 11
- C. 12
- D. Compilation Error

## Question: 17

```
public class Main {  
    public static void main(String[] args) {  
        int a = 5 + 5 * 2 + 2 * 2 + (2 * 3);  
        System.out.println(a);  
    }  
}
```

- A. 138
- B. 264
- C. 41
- D. 25

## Question: 18

```
class char_increment {  
    public static void main(String args[])  
    {  
        char c1 = 'D';  
        char c2 = 84;  
        c2++;  
        c1++;  
        System.out.println(c1 + " " + c2);  
    }  
}
```

- A. E U
- B. U E
- C. V E
- D. U F

## Question: 19

```
class conversion {  
    public static void main(String args[]) {  
        double a = 295.04;  
        int b = 300;  
        byte c = (byte) a;  
        byte d = (byte) b;  
        System.out.println(c + " " + d);  
    }  
}
```

- A. 38 43
- B. 39 44
- C. 295 300
- D. 295.04 300


## Question: 20

```
class mainclass {  
    public static void main(String args[]) {  
        char a = 'A';  
        a++;  
        System.out.print((int) a);  
    }  
}
```

- A. 66
- B. 67
- C. 65
- D. 64

## Question: 21

What is the incorrect statement about non-primitive data types or user-defined data types?

- A. Non-primitive data types can be created using the primitive data types.
  - B. Non-primitive data types can contain only primitive data types. They can not contain other non-primitive/user-defined data types in them.
  - C. Non-primitive data types can contain both primitive data types and other non-primitive/user-defined data types.
  - D. None of the above
- 

## Question: 22

```
class variable_scope {  
    public static void main(String args[]) {  
        int x;  
        x = 5; {  
            int y = 6;  
            System.out.print(x + " " + y);  
        }  
        System.out.println(x + " " + y);  
    }  
}
```

- A. 5 6 5 6
- B. 5 6 5
- C. Runtime error
- D. Compilation error





## Question: 23

If we have to store the following information about a student, how many values of from each data type group do we need.

1. Student name
2. College name
3. Roll Number
4. Total Marks
5. Percentage
6. Height in Feet
7. Gender

- A. 0 Boolean, 1 Characters, 3 Floating Point and 3 Integers
- B. 1 Boolean, 2 Characters, 2 Floating Point and 2 Integers
- A. 2 Boolean, 1 Characters, 3 Floating Point and 1 Integers
- A. 0 Boolean, 5 Characters, 2 Floating Point and 0 Integers



## Question: 24

What is not an example of literals?

A. 23.4

B. 198

A. 'a'

B. None of the above

## Question: 25

```
class array_output
{
    public static void main(String args[])
    {
        int array_variable [] = new int[10];
        for (int i = 0; i < 10; ++i)
        {
            array_variable[i] = i;
            System.out.print(array_variable[i] + " ");
            i++;
        }
    }
}
```

- A. 0 2 4 6 8
- B. 1 2 3 4 5
- C. 0 1 2 3 4 5 6 7 8 9
- D. 1 2 3 4 5 6 7 8 9 10



/ethnuscodemithra



Ethnus Codemithra



/ethnus



/code\_mithra

<Codemithra /><sup>TM</sup>



<https://learn.codemithra.com>



Explore | Expand | Enrich



[codemithra@ethnus.com](mailto:codemithra@ethnus.com)



+91 7815 095 095



+91 9019 921 340