SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NAAC – UGC with 'A' Grade

Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

23CSB101

OBJECT ORIENTED PROGRAMMING

Data Type, Variable, Array

By

M.Kanchana

Assistant Professor/CSE

Data Types

- **Data type is used to allocate sufficient memory space for the data. Data types specify the different sizes and values that can be stored in the variable.**
  - ***Java is a strongly Typed Language.***

```
int x = 10;
float y = x; // Implicit conversion allowed in C
```

```
int x = 10;
float y = x; // Implicit widening conversions in java
```

```
float x = 10;
Int  y = x; // does not allow implicit narrowing conversions
```

Type Casting

In Java, type casting is the process of converting a value from one data type to another.

Explicit (Narrowing) Type Casting – Done manually using (type).

```
float x = 10.5;
int y = (int) x;
```

```
Float value: 10.5
Converted int value: 10
```

```
float x = 10;
Int  y = x; // does not allow implicit narrowing conversions
```

Types of Data types

- Primitive data types (Intrinsic or built-in types)
- Non-primitive data types (Derived or Reference Types)

Primitive data types

- **Primitive data types (Intrinsic or built-in types)**
  Primitive data types are those whose variables allow us to store only one value and never allow storing multiple values of same type. This is a data type whose variable can hold maximum one value at a time.

Eight primitive types in Java:

Integer Types:
1. int
2. short
3. long
4. byte

Floating-point Types:
5. float
6. double

Others:
7. char
8. Boolean

Primitive data types

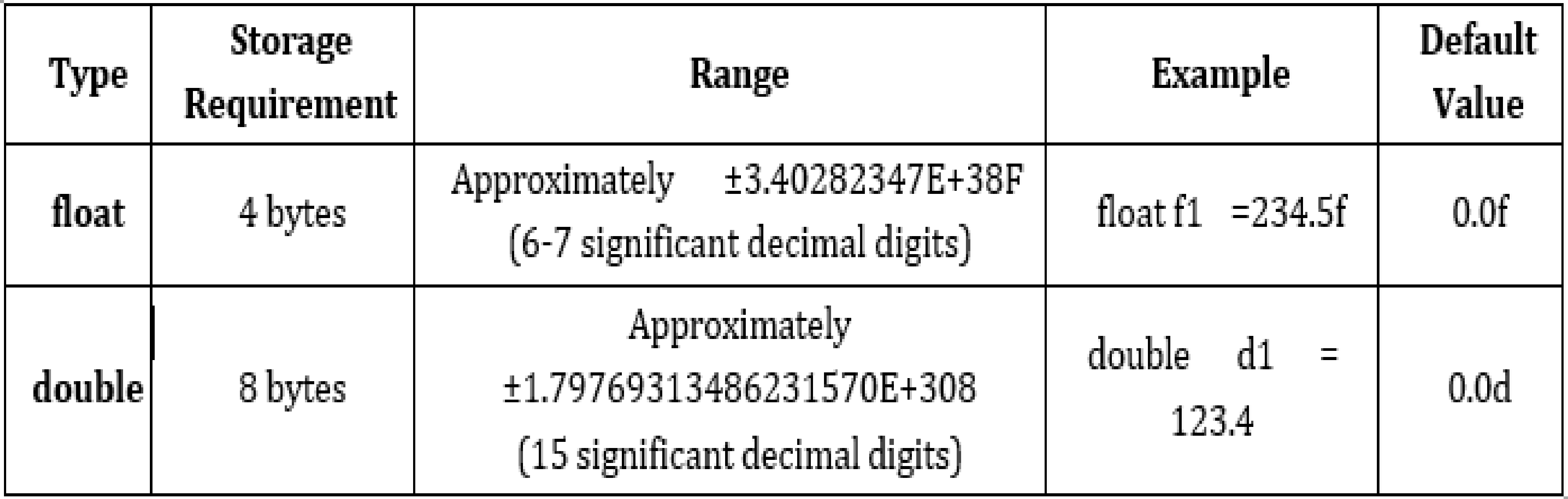
Integer Types:

The integer types are form numbers without fractional parts. Negative values are allowed.

Type	Storage Requirement	Range	Example	Default Value
int	4 bytes	-2,147,483,648(-2^31) to 2,147,483,647 (2^31-1)	int a = 100000, int b = -200000	0
short	2 bytes	-32,768 (-2^15) to 32,767 (2^15-1)	short s = 10000, short r = -20000	0
long	8 bytes	-9,223,372,036,854,775,808 (-2^63) to 9,223,372,036,854,775,808 (2^63-1)	long a = 100000L, int b = -200000L	0L
byte	1 byte	-128 (-2^7) to 127 (2^7-1)	byte a = 100 , byte b = -50	0

Primitive data types

Floating-point Types:
The floating-point types denote numbers with fractional parts



Type	Storage Requirement	Range	Example	Default Value
float	4 bytes	Approximately ±3.40282347E+38F (6-7 significant decimal digits)	float f1 =234.5f	0.0f
double	8 bytes	Approximately ±1.79769313486231570E+308 (15 significant decimal digits)	double d1 = 123.4	0.0d

Primitive data types

char:

- char data type is a single 16-bit Unicode character.
- Minimum value is '\u0000' (or 0).
- Maximum value is '\uffff' (or 65,535 inclusive).
- Char data type is used to store any character.
- **Example: char letterA ='A'**

Primitive data types

boolean:

- boolean data type represents one bit of information.
- There are only two possible values: **true and false.**
- This data type is used for simple flags that track true/false conditions.
- Default value is false.
- Example: **boolean one = true**

Non-primitive data types
(Derived or Reference Types)

- **Derived data types** are built using primitive data types
- They provide structured and reusable ways to manage data.

1. Arrays
2. Classes
3. Interfaces
4. Strings
5. Enum
6. Objects

ARRAY

- An **array** is a collection of elements of the **same data type** stored in **contiguous memory locations**.

- Arrays allow **efficient indexing** and **easy data manipulation**

```
dataType[] arrayName;
                    Or
dataType arrayName[];
                    Or
dataType []arrayRefVar
```

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

Why Use Arrays?

- **Efficient storage** for multiple values of the same type.
- **Easy access** using index positions.
- **Reduces code complexity** compared to multiple variables.

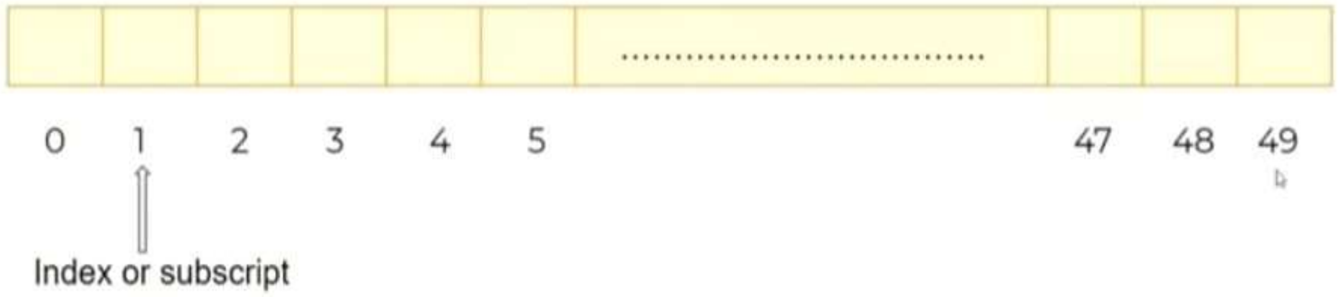
Instead of declaring multiple variables:

```
int num1, num2, num3;
```

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

Arrays

```
int marks[] = new int[50];
```



Declaring and Initialization of an Array

- **Reduces code complexity** compared to multiple variables.

Instead of declaring multiple variables:

```
int num1, num2, num3, num4, num5;
```

int[] numbers = new int[5];

or

int[] numbers = {10, 20, 30,40,50};// direct assignment

Accessing Array Elements

- Access elements using **indexing** (0-based index).

```
int[] numbers = {10, 20, 30, 40, 50};

System.out.println(numbers[0]); // Output: 10
System.out.println(numbers[3]); // Output: 40
```

Modifying Array Elements

```
int[] numbers = {10, 20, 30,40,50};

numbers[1] = 50; // Change value at index 1

System.out.println(numbers[1]); // Output: 50
```

Array Length Property

Use **.length** to get array size.

```
int[] numbers = {10, 20, 30, 40};
System.out.println(numbers.length);
```

Looping Through an Array

Use **.length** to get array size.

```
int[] numbers = {10, 20, 30, 40};
for (int i = 0; i < numbers.length; i++)
 {
System.out.println(numbers[i]);

}
```

Multi-Dimensional Arrays

- Arrays can have **multiple dimensions (rows & columns).**

Declaring a **2D array:**

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

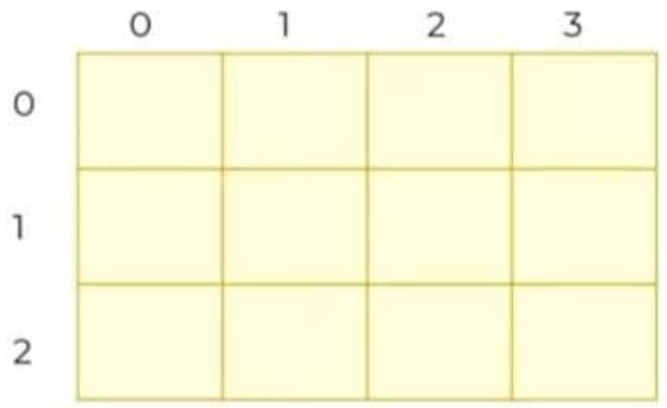
Initializing a **2D array:**

```
int[][] matrix = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };
System.out.println(matrix[1][2]);


 // Output: 6
```

Arrays

```
int marks[][] = new int[3][4];
```

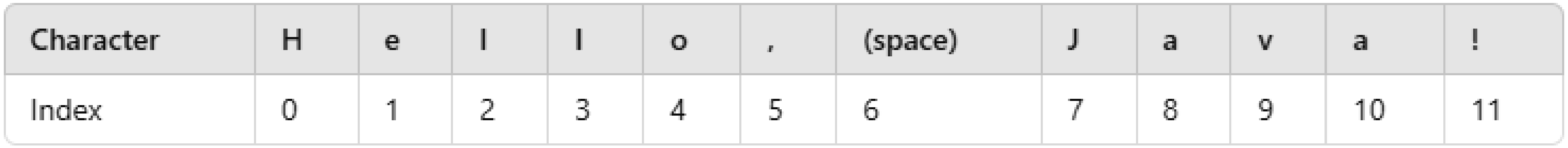


Strings

A sequence of characters, represented as an object of the String class.

```
String str = "Hello, Java!";
System.out.println(str.length());
```

Output: 12



Character	H	e	l	l	o	,	(space)	J	a	v	a	!
Index	0	1	2	3	4	5	6	7	8	9	10	11

enum

An enum (short for enumeration) in Java is a special data type that defines a fixed set of constants. It is used when a variable can take one of a limited set of predefined values.

Example Use Cases:

•Days of the week (SUNDAY, MONDAY, ...)
•Directions (NORTH, SOUTH, EAST, WEST)
•Traffic light signals (RED, YELLOW, GREEN)
•Order statuses (PENDING, SHIPPED, DELIVERED)

```
 enum Day {SUNDAY, MONDAY, TUESDAY};
  Day today = Day.MONDAY;
```

JAVA Variables

- A Variable is a named piece of memory that is used for storing data in java Program.
- A variable is an identifier used for storing a data value.
- A Variable may take different values at different times during the execution if the program, unlike the constants.

Syntax to declare variables:

datatype identifier [=value][,identifier [=value] …];

int average=0.0, height, total height;

JAVA Variables

Rules followed for variable names:

1. A variable name must begin with a letter and must be a sequence of letter or digits.
2. They must not begin with digits.
3. Uppercase and lowercase variables are not the same.

   Example: **T**otal and **t**otal are two variables which are distinct.
4. It should not be a keyword.
5. Whitespace is not allowed.
6. Variable names can be of any length.

JAVA Variables

Two ways to initialize a variable:

1. Initialize after declaration:

Syntax:

int months;
months=1;

2.Declare and initialize on the same line:

Syntax:

int months=12;

JAVA Variables

Dynamic Initialization of a Variable:
Java allows variables to be initialized dynamically using any valid expression at the time the variable is declared.

Example: Program that computes the remainder of the division operation:

```
class FindRemainer
{
public static void main(String arg[]) {
int num=10,den=2;
int rem=num%den;
System.out.println("Remainder is"+rem);
}
}
```

JAVA Variables_Types

Local Variable

- These are variables that are declared inside a method or a block and can only be used within that method or block.
- They are created when the method is called and destroyed when the method finishes execution.
- Local variables do not have any default value, so they need to be explicitly initialized before use.

```
public void exampleMethod() {
    int localVar = 10; // local variable
    System.out.println(localVar);
}
```

JAVA Variables_Types

Instance Variables:

•These are variables that are declared inside a class but outside any method or constructor.

•They are specific to each object (instance) of the class. Each object gets its own copy of instance variables.

•Instance variables can have default values (e.g., 0 for integers, null for objects) if they are not explicitly initialized.

```
public class Person {
    String name; // instance variable
    public void setName(String name) {
        this.name = name; // refers to the instance variable
    }
}
```

JAVA Variables_Types

Class/Static Variables:

•These are variables declared with the **static** keyword, meaning they belong to the class rather than to any particular instance of the class.

•They are initialized once when the class is loaded into memory..

1.The static variable is shared across all objects of the class.

1.The static method can be called directly on the class without creating an instance.

JAVA Variables_Types

```
public class Counter {
    // Static variable (shared among all instances of the
class)
    static int count = 0;

    // Method to increment the static variable
    public void increment() {
        count++;
    }

    // Static method to access the static variable without
creating an object
    public static void showCount() {
        System.out.println("Count: " + count);
    }

  public static void main(String[] args) {
      // Creating instances (objects) of Counter class
      Counter c1 = new Counter();
      Counter c2 = new Counter();

      // Calling the increment method on both objects
      c1.increment();
      c2.increment();

      // Static method can be called using the class name
      Counter.showCount();  // Output: Count: 2
  }
}
```

THANK YOU