6. Algorithm vs Code

Algorithm -> Steps to solve any problem. these are programming independent.

Example :
Implementation of Stack
Algorithm -> Implementation using any programming language

To start with Coding an algorithm following concepts are must required :

1. Variables and Data Types
    a. Creating and using variables
    b. Reference Variables (Pointers)
2. Array 1D and 2D
    a. Creating
    b. Initializing
    c. Traversing
    d. Modifying
3. Control Flow Statements
    a. if
    b. if else
    c. if else-if
    d. for
    e. while
    f. do While
    g. break
    h. jump
4. Classes and Objects
    a. How to create classes and objects
    b. constructors
    c. using methods

Variable and Data Types in Java

Variables are used to store data that can be used and manipulated throughout the program.

```java
public static void integers(){  1 usage
    int x = 50;
    int y = 10;
    System.out.println("value of x is : " + x);
    System.out.println("value of y is : " + y);
}
```

Data Types :

Primitive Data Types : These are Data Type which comes with or bundled with java.

1. byte : 8 bit integer
2. short : 16 bit integer
3. int : 32 bit i.e. 4 bytes
4. long : 64 bit
5. float : 32 bit floating point, used for decimals
6. double : 64 bit floating point, used for decimals
7. char : 16 bit Unicode characters, stores single character
8. boolean : holds true and false

Non-Primitive Data Types : These are User Defined Data Types i.e. user creates these data types using classes and uses them through objects in java.

Example :

Suppose there is student who has name, class, city, age, marks.
Here none of the primitive data type can be used to store these details.
so we need to created non-primitive data type Student i.e. Class

```java
class Student{  2 usages

    String name;  1 usage

    String className;  1 usage

    String city;  1 usage

    int age;  1 usage

    float marks;  1 usage
}
```

```java
    /**
     * Using non-primitive Data Type Student
     */
    Student student = new Student();
    System.out.println("\nnon-primitive data type in Java");
    System.out.println(student.name = "Jhon Doe");
    System.out.println(student.age = 22);
    System.out.println(student.city = "San Francisco");
    System.out.println(student.className = "12th");
    System.out.println(student.marks = 91.00f);
```

Reference Variables
in above program all the data is stored in student object.
so here student is the reference variable.

Java Operators

Unary operator : expr++, expr--, ++expr, --expr, +expr, -expr, ~|
Arithmetic operator : +, -, *, /, %
Relational operator : <, >, <=, >=, ==, !=
Logical operator : &&, ||
Ternary operator : ? :
    Example : expression ? value1 : value2
    here expression will return true or false, if true then value is 1 and if false then value is 2
Assignment operator : =, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=
Bitwise operator : &, ^, |

Control Flow Statements

if statement

```
int n = 12;

if(n > 0){
    System.out.println("yes " + n + " is greater then 0");
}
```

if else

```
int n = -12;

if(n > 0){
    System.out.println("yes " + n + " is greater then 0");
}
else {
    System.out.println("no " + n + " is not greater then 0");
}
```

ladder if

```
int a = 5;

if(a > 0){
    System.out.println("yes " + a + " is greater then 0");
    if(a > 1){
        System.out.println("yes " + a + " is greater then 1");
        if(a > 2){
            System.out.println("yes " + a + " is greater then 2");
            if(a > 3){
                System.out.println("yes " + a + " is greater then 3");
                if(a > 4){
                    System.out.println("yes " + a + " is greater then 4");
                }
            }
        }
    }

}
```

ladder if/Nested if with else

```java
int a = -5;

if(a > 0){
    System.out.println("yes " + a + " is greater then 0");
    if(a > 1){
        System.out.println("yes " + a + " is greater then 1");
        if(a > 2){
            System.out.println("yes " + a + " is greater then 2");
            if(a > 3){
                System.out.println("yes " + a + " is greater then 3");
                if(a > 4){
                    System.out.println("yes " + a + " is greater then 4");
                }
            }
        }
    }
}
else {
    System.out.println("no " + n + " is not greater then 0");
}
```

if else-if

```java
int x = -5;

if(x > 4){
    System.out.println("yes " + x + " is greater then 4");
} else if (x > 0) {
    System.out.println("yes " + x + " is greater then 0");
}
else{
    System.out.println("no " + x + " is not greater then 0");
}
```

switch statement

traditional switch statement :

```java
public static void switchStatement() {  no usages
    int itemCode = 0;
    switch (itemCode) {
        case 0:
            System.out.println("yes");
            break;
        case 1:
            System.out.println("no");
            break;
        case 2:
            System.out.println("yes");
            break;
        case 3:
            System.out.println("no");
            break;
        default:
            System.out.println("no");
            break;
    }
```

Enhanced switch statement :

```java
public static void switchStatement() {  no usages
    int itemCode = 0;
    switch (itemCode){
        case 1,2,3,4,5:
            System.out.println("its an electronic gadget.");
            break;

        case 6, 7, 8, 9, 10:
            System.out.println("its not an electronic gadget.");
            break;
    }
}
```

```java
String response = "";
switch (response){
    case "ok", "accepted":
        System.out.println("1XX");
        break;
    case "no", "rejected":
        System.out.println("4XX");
}
```

Loops

Used in programming language for repeating statements for n number of times

for loop

start : init

end : termination

step : jumps

```java
public static void forLoopStatement() {  1 usage
    for(int i = 0; i <= 10; i++){
        System.out.println("Value of i : " + i);
    }

    for(int i = 10; i <= 1; i--){
        System.out.println("Value of i : " + i);
    }
}
```

while loop also known as entry control loop

while(condition/expr){

statement

)

```
public static void whileLoopStatement() {  no usages
    int i = 100;
    while (i<=110) {
        System.out.println("Value of i : " + i);
        i++;
    }
}
```

do while loop also known as exit control loop

do{

}

while(condition);

```
public static void doWhileLoopStatement() {  1 usage
    int i = 100;
    do {
        System.out.println("Value of i : " + i);
        i++;
    }
    while (i<=110);
}
```

Enhanced for loop also known as for-each loop mostly used to traverse arrays and Collection dataStructures

for(int x : num){

statement wrt. x;

}

```
public static void forEachLoopStatement() {  1 usage
    int arr[] = {1,2,3,4,5,6,7,8,9,10};
    for(int i : arr){
        System.out.println("Value of i : " + i);
    }
}
```

Break and Continue

Break :

```java
public static void breakStatement() {  1 usage
    for(int i = 0; i <= 10; i++){
        System.out.println("Value of i : " + i);
        if(i == 5){
            break;
        }
    }
}
```

Continue :

```java
public static void continueStatement() {  no usages
    for(int i = 0; i <= 10; i++){
        System.out.println("Value of i : " + i);
        if(i == 5){
            continue; //here when i have value of 5 it will again re-iterate the loop and code/implementation below continue statement will be skipped.
        }
        if(i == 5){
            break;
        }
    }
}
```

Classes and Objects - OOP

OOP is a programming paradigm that organizes code around objects.

Here are some key concepts of OOPS:

a. Objects : Instance of class that represents real-world entities or concepts. Object can have properties (attributes) and behaviours(method).

b. Class : Blueprint for creating object. They define the properties and behaviour that object of the class will have.

c. Inheritance: Creating new classes that inherits properties and behaviours from existing class, it can be achieved using extends keyword
   - final class cannot be inherited.
   - final methods cannot be overridden.
   - final variables cannot be changed.

d. Encapsulation : Bundling data and methods together within an object, protecting data from unauthorized access.

e. Polymorphism : The Ability of object of different class to respond differently to the same message.

f. This keyword is used to call current class constructor,it is used to remove the naming confliction between instance variables and local variables.

g. Constructor Overloading :
   - Number of parameters are different.

- Type of parameters are different.
- Order of parameters are different.