1️⃣

Object Oriented Programming : Introduction & Concepts - Classes, Objects, Constructors, Keywords

GitHub - kunal-kushwaha/DSA-Bootcamp-Java: This repository consists of the code samples, assignments, and notes for the Java data st
This repository consists of the code samples, assignments, and notes for the Java data structures & algorithms + interview preparation bootcamp of WeMake

https://github.com/kunal-kushwaha/DSA-Bootcamp-Java/tree/main?tab=readme-ov-file

```
// storing 5 roll numbers
public class Main{
    public static void main(String[] args){
        // stores 5 roll values
        int[] numbers=new int[5];
        //stores 5 names
        String[] names=new String[5];


        //Data of 5 students : Roll num,Name and Marks

    int[] roll_num=new int[5];
    String[] name=new String[5];
    float[] marks=new flaot[5];

    /*What if i need a single Data
    structure which has all the three data of students in it
    instead of having 3 diff */
```

Classes

- Named group of properties and Functions.
- Example: in the above example, If i want to combine the 3 diff data types into a single entity. I need to create a class.

[creation of own data type can be also donne using classes]

Creating a Class

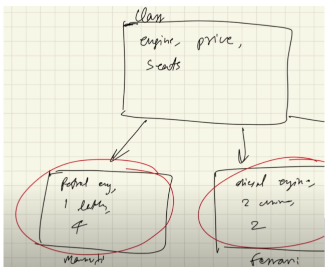
```
//Class name always should start with CAPITAL LETTER
class Student{
    int[] roll_num=new int[5];
    String[] name=new String[5];
    float[] marks=new flaot[5];
}

/*Class acts as a Template*/
```

now, if i want to create first students data,

```
Student[] students=new Student[5];
Student Aish;
```

Real world Examples

- Cars (companies create diff cars based on certain properties)
- Humans



'Car' is a class with Objects 'Maruti', 'Ferrari'

> Now, **Class can be defined as Template of an objects and logical Constructs** and **Objects are instance (Physical real world entity) of a class [*Ocuupies space in a memory*]**

Objects are Categorised by

- **State** : Value from its Data type
- **Identity** : One obj is different from another
- **Behaviour :** Effect of datatype operations

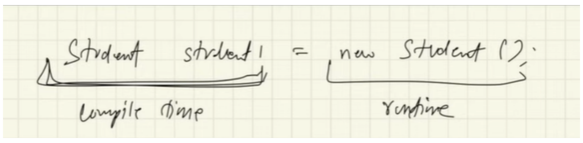
How to access the Instance Variable

- we use sepearator(dot operator)linking Red varibale name along with the instance variable name

Creating new objects

```
//data type for every single student
class Student{
    int rollNo;
    String name;
    float marks;}
```

```

Student[] students=new Student[20];


//Aish is a object and Student is a data type
Student Aish; //declaring the object
/* 'new' Dynamically allots the memory and return
reference to it */
Aish=new Student();
```



Manupulation of Objects

```
class Student{
    int rollNo;
    String name;
    float marks; }

Student Aish=new Student();
System.out.println(Aish.rollNo); //gives output as '0'
System.out.println(Aish.name); // null
System.out.println(Aish.marks); //0.0

-----------------------------------------------------

Aish.name="AISHWARYA";
Aish.rollNo=13;
Aish.marks=89;
System.out.println(Aish.rollNo);
System.out.println(Aish.name);
System.out.println(Aish.marks);

------------------------------------------------------
// if the values are defualt:
class Student{
    int rollNo;
    String name;
    float marks=90; }

Aish.name="AISHWARYA";
Aish.rollNo=13;
Aish.marks
System.out.println(Aish.rollNo);
System.out.println(Aish.name);
System.out.println(Aish.marks);

-------------------------------------------------------

/* If, i have 1000 objects, the code snippet bellow cannot be
written 1000 times -> Do this inside a CONSTRUCTOR

Aish.name="AISHWARYA";
Aish.rollNo=13;
Aish.marks=89;
System.out.println(Aish.rollNo);
System.out.println(Aish.name);
System.out.println(Aish.marks); */
```

Constructor

- Defines what happens when an object is created

- **Is a special fxn that runs when we create an object** and it allocates some variables.

```
class Student{
    int rollNo;
    String name;
    float marks;
}

Student Aish=new Student(); //Student() is a By-default Constructor

/* constructor where we want the 3 args to bind with
the object :
Student Aish=new Student(20,"Aishwarya",98);
System.out.println(Aish.rollNo); // this gives 20
System.out.println(Aish.name); //Aishwarya
System.out.println(Aish.marks); //98 */

----------------------------------------------------------------------------

class Student{
    int rollNo;
    String name;
    float marks;


    /* we need to add the values of the above properties
    object by object.


    we need one word to access every object */

    Student(){
        this.name="AISHWARYA";
        this.rollNo=13;
        this.marks=89;
    }

    /* "this" Keyword When the new object is created,
    points to the Constructor */

    // 'this' is replaced with the name of the Ref variable
}
--------------------------------------------------------------------

class Student{
    int rollNo;
    String name;
    float marks;

    Student(int roll, String naam, float mark){
        this.name=naam;
        this.rollNo=roll;
        this.marks=mark;
    }

}
```

```
Student Aish=new Student(19,"whisky",92.4);
```

Constructor Overloading

```
class Student{
    int rollNo;
    String name;
    float marks;

    Student(){
        this.name="Whisky";
        this.rollNo=18;
        this.marks=90;
    }

    Student(int roll, String naam, float mark){
        this.name=naam;
        this.rollNo=roll;
        this.marks=mark;
    }

}


Student Whisky=new Student();
// This calls the constructor 'Student()'
Student Aish=new Student(19,"whisky",92.4);
// This calls the constructor ' Student(19,"whisky",92.4)'
```

Constructor that takes value from another object

```
class Student {
    int rollNo;
    String name;
    float marks;

    // Normal constructor
    Student(int roll, String naam, float mark) {
        this.rollNo = roll;
        this.name = naam;
        this.marks = mark;
    }

    // Constructor that takes another Student object
    Student(Student other) {
        this.rollNo = other.rollNo;
        this.name = other.name;
        this.marks = other.marks;
    }
}

// Usage example
Student original = new Student(19, "Aishwarya", 92.4f);
Student copy = new Student(original);

System.out.println(copy.name);  // Output: Aishwarya
System.out.println(copy.rollNo);  // Output: 19
System.out.println(copy.marks);  // Output: 92.4
```

In this example, we've added a new constructor that takes another Student object as a parameter. This constructor copies the values from the passed object to create a new Student instance with the same data. This is often called a copy constructor.

Calling a CONSTRUCTOR from another CONSTRUCTOR

```
class Student{
    int rollNo;
    String name;
    float marks;

    Student(){
    //calling a constructor from another constructor
        this(13,"random Person",99);
    }

    Student(int roll, String naam, float mark){
        this.name=naam;
        this.rollNo=roll;
        this.marks=mark;
    }

}

Student random=new Student();
```

```
Student one=new Student();
Student two=one;

/*These are pointing to the same object in the
Heap memory*/

one.name="blah blah blah";
System.out.println(two.name);
```

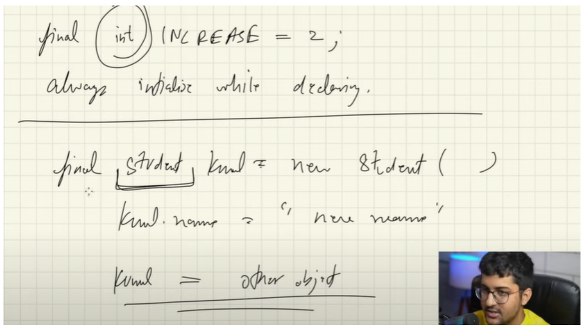
Wrapper Classes

```
public class WrapperExamples{
    public static void main(String[] args){
        int a=10;//primitive

        Integer num=45;//object of type integer
        }
    }
```

Final : Keyword

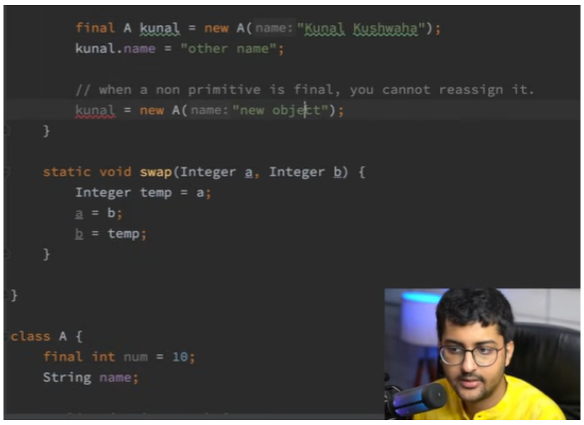
- prevent content being modified (keeping it constant)
- **Final variable need to be always initialised while we declare it.**
- This is only when the data types are primitives.

```
final int INCREASE =10;
INCREASE =90; // this throws an error
```



final (int) INCREASE = 2;
always initialise while declaring.

final student kunal = new Student( )
kunal.name = "new name"
kunal = other object



```
        final A kunal = new A( name: "Kunal Kushwaha");
        kunal.name = "other name";

        // when a non primitive is final, you cannot reassign it.
        kunal = new A( name: "new object");
    }

    static void swap(Integer a, Integer b) {
        Integer temp = a;
        a = b;
        b = temp;
    }

}

class A {
    final int num = 10;
    String name;
```

Garbage Collection

Refer Intro

Finalise() : what to do when the memory is destroyed.(Memory is destroyed by default when garbage collection hits)