**Singleton class**

**What is a singleton class?**

|  |
| --- |
| It is a class for which we can create only one object |

**Example on singleton class**

class One

{

    private static One o1;

    int a,b;

    private One()

    {   super();

        System.out.println("One class default constructor.");

        /\*a=0;b=0; \*/

    }

    public static One getInstance()//One

    {

        if(o1==null)

        {

            o1=new One();

        }

        return o1;

    }

    public void setA(int a)//r1=1002

    {

        this.a=a;

    }

    public void setB(int b)//r1

    {

        this.b=b;

    }

    public int getA()

    {return a;}

    public int getB()

    {return b;}

}

public class Singleton

{

    public static void main(String[] args)

    {        //r1==1002

           One r1=One.getInstance();

           r1.setA(100);

           r1.setB(200);

           One r2=One.getInstance();

           One r3=One.getInstance();

           System.out.println("r1 object state...");

           System.out.println("a:\t"+r1.getA());

           System.out.println("b:\t"+r1.getB());

           System.out.println("r2 object state...");

           System.out.println("a:\t"+r2.getA());

           System.out.println("b:\t"+r2.getB());

           System.out.println("r3 object state...");

           System.out.println("a:\t"+r3.getA());

           System.out.println("b:\t"+r3.getB());

    }

}

**Immutable object**

|  |
| --- |
| It is an object whose data (object state) can’t be changeable |

**What is the default value of a final variable?**

|  |
| --- |
| 1. Compiler doesn’t initialize final fields 2. Final fields must be initialized explicitly by the programmer. |

**Can we initialize final fields by using setter methods?**

|  |
| --- |
| No we can’t. we have to initialize them either at the time of declaration or by using constructor. |

**Example on creating immutable objects**

**ImmutableDemo.java**

final class One

{

    private final int a,b;

    One(int a,int b)

    {

        this.a=a;

        this.b=b;

    }

    public int getA()

    {return a;}

    public int getB()

    {return b;}

}

public class ImmutableDemo

{

    public static void main(String[] args)

    {

        One o1=new One(100,200);

        System.out.println("Object state....");

        System.out.println("o1.getA():\t"+o1.getA());

        System.out.println("o1.getB():\t"+o1.getB());

    }

}

**Output:**

bject state....

o1.getA(): 100

o1.getB(): 200

**contextual keyword record**

|  |
| --- |
| By using record contextual keyword we can write a class for which we can create only immutable objects. |

**Example**

record One(int a,int b){

    public int add()

    {

        return a+b;

    }

}

public class ImmutableDemo

{

    public static void main(String[] args)

    {   One o1=new One(100,200);

        System.out.println("Object state....");

        System.out.println("o1.a():\t"+o1.a());

        System.out.println("o1.b():\t"+o1.b());

        System.out.println("o1.add():\t"+o1.add());

        System.out.println(o1);

    }

}

Output:

Object state....

o1.a(): 100

o1.b(): 200

o1.add(): 300

One[a=100, b=200]

**Concepts we have covered till now**

* + 1. **Introduction**
    2. **Features**
    3. **Jdk installations**
    4. **VSCode installation**
    5. **JCL**
    6. **Java API**
    7. **Escape sequences**
    8. **Naming conventions**
    9. **Compilation**
    10. **Execution**
    11. **Data types**
    12. **If**
    13. **If..else**
    14. **If..else..if**
    15. **Nested if**
    16. **Switch**
    17. **Array**
    18. **1D arrays**
    19. **2d arrays**
    20. **3d arrays**
    21. **Jagged arrays**
    22. **For loop**
    23. **For each**
    24. **While**
    25. **Do..while..**
    26. **Functions(methods)**
    27. **Class**
    28. **Object**
    29. **Instance variables**
    30. **Static variables**
    31. **Instance methods**
    32. **Static methods**
    33. **Instance block**
    34. **Static block**
    35. **Constructors**
    36. **This keyword**
    37. **String constant pool**
    38. **Accessor and mutator methods**
    39. **Inheritance**
    40. **Super()**
    41. **Method overloading**
    42. **Method overriding**
    43. **Upcasting & Down casting**
    44. **Static and dynamic binding**
    45. **Compile time and runtime Polymorphism**
    46. **Abstract classes**
    47. **Interfaces**
    48. **Inner classes**
    49. **Anonymous inner classes**
    50. **Functional interfaces**
    51. **Lambda expressions**
    52. **Singleton**
    53. **Immutable object**
    54. **Creating record using contextual keyword called ‘record’**