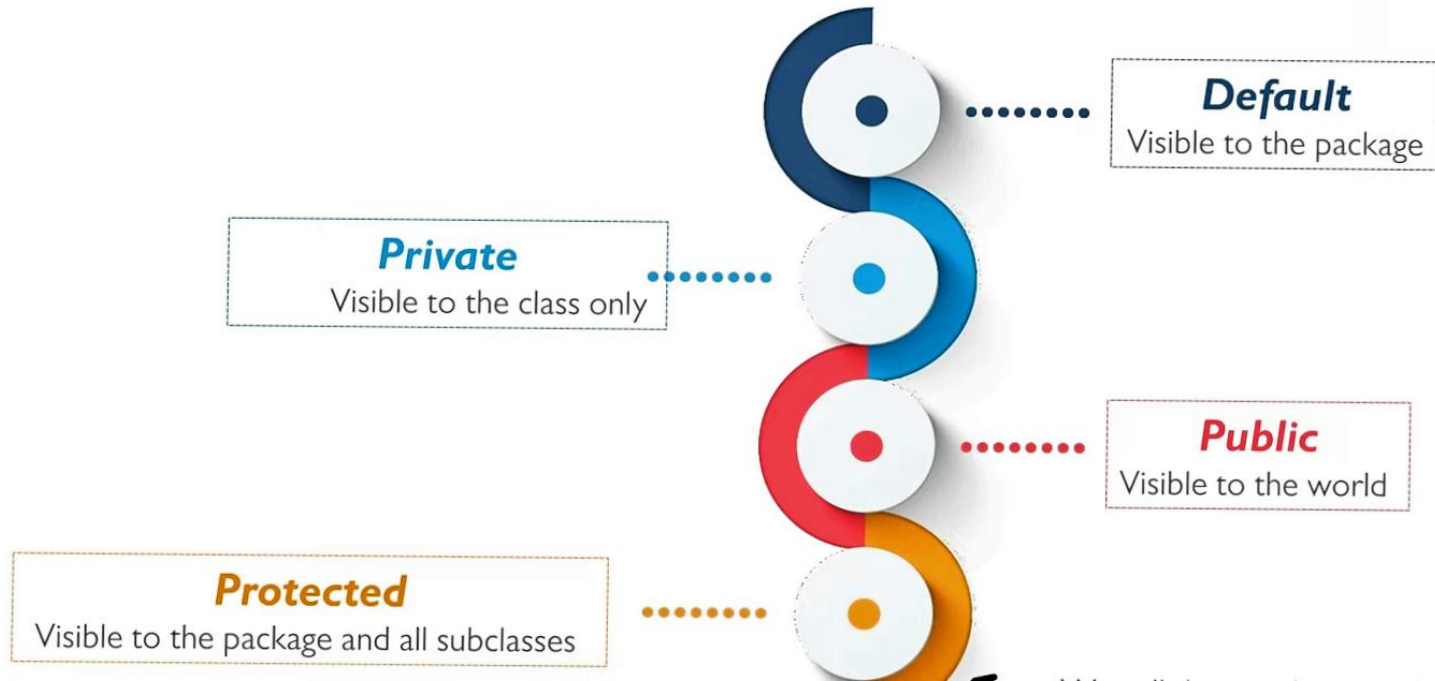


# Access Control Modifiers

Java provides a number of access control modifiers to set access levels for classes, variables, methods and constructors

The four access levels are –



# Non-Access Modifiers

Java provides a number of non-access modifiers to provide additional functionalities to a class, variable, method or constructor

Some of them are:

**static**

The static modifier for calling methods and variables without an object to which it belongs

**final**

The final modifier for finalizing the implementations of classes, methods, and variables

**abstract**

The abstract modifier for creating abstract classes and methods

**synchronized and volatile**

The synchronized and volatile modifiers, which are used for threads

# Java *Static* Keyword

---

Static is a non access modifier used in Java, applicable for blocks, methods, class variables

When the static keyword is used to declare any parameter, then memory is allocated only once for that parameter

The Static keyword is used for memory management

Static keyword is used to refer the common properties of an object



# Demo – Java *Static* Keyword

1

company is a Static variable. It allocates memory only once

```
class Employees{
    int id;
    int salary;
    static String company="SRT Traders";

    Employees(int i, int s){
        id=i;
        salary=s;
    }

    void display() {
        System.out.println(id+" "+salary+" "+company);
    }
}

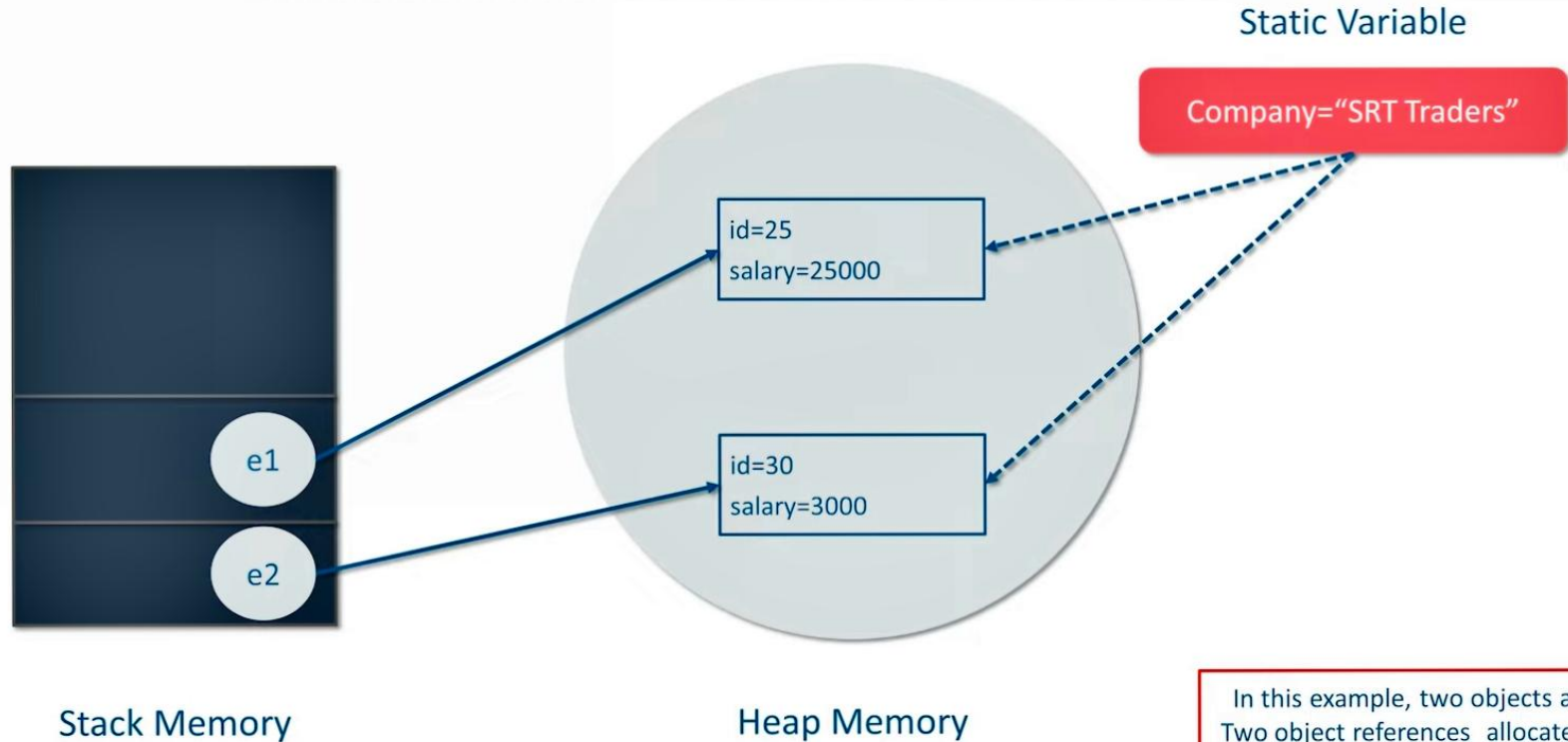
public class StaticKey {
    public static void main(String[] args) {

        Employees e1=new Employees(25,25000);
        Employees e2=new Employees(30,3000);
        e1.display();
        e2.display();
    }
}
```

Output

25 25000 SRT Traders  
30 3000 SRT Traders

# Demo – Java *Static* Keyword



In this example, two objects are created. Two object references allocate memory in the Stack. But as the **company** variable is **static**, memory is allocated only once for it

# Demo – Java *static* Keyword

2

A static method can be invoked without the need for creating instance of a class

```
class Employees{
    int id;
    int salary;
    static String company="SRT Traders";

    static void check() {
        company="WIIT";
    }

    Employees(int i, int s){
        id=i;
        salary=s;
    }

    void display() {
        System.out.println(id+" "+salary+" "+company);
    }
}

public class StaticKey {
    public static void main(String[] args) {
        Employees.check();
        Employees e1=new Employees(25,25000);
        Employees e2=new Employees(30,3000);
        e1.display();
        e2.display();
    }
}
```

Output

25 25000 WIIT  
30 3000 WIIT

# Java *this* Keyword

The *this* keyword is used as reference variable to the current object

This can be passed as an argument in the constructor call

*this* can be used to invoke current class method

*this* can be passed as an argument in the method call

*this*() can be used to invoke current class constructor

The *this* keyword is mainly used to refer current class instance variable and it can also be used to return the current class instance from the method.





# Demo – Java *this* Keyword

1

```
class ClassInfo{
    int rollno;
    String name;

    ClassInfo(int rollno,String name){
        this.rollno=rollno;
        this.name=name;
    }
    void display(){System.out.println(rollno+" "+name);}
}

public class ThisDemo {

    public static void main(String[] args) {
        ClassInfo c1=new ClassInfo(10,"John");
        ClassInfo c2=new ClassInfo(12,"Annie");
        c1.display();
        c2.display();
    }
}
```

Output

10 John  
12 Annie

If local variables and instance variables are different, there is no need to use this keyword



# Demo – Java *this* Keyword

2

```
class Message{  
  
    Message() {  
        this("Annie");  
        System.out.println("Welcome to Edureka");  
    }  
  
    Message(String n){  
        System.out.println(n);  
    }  
}  
  
public class This2 {  
  
    public static void main(String[] args) {  
        Message m=new Message();  
    }  
}
```

Output

Annie  
Welcome to Edureka

# How Does this Keyword Work?

When an object of the class is created, a default constructor is called

2

```
class Message{  
    Message() {  
        this("Annie");  
        System.out.println("Welcome to Edureka");  
    }  
    Message(String n){  
        System.out.println(n);  
    }  
}
```

The main() function is executed first

1

```
public class This2 {  
    public static void main(String[] args) {  
        Message m=new Message();  
    }  
}
```

4

The remaining statements from default constructor gets executed

3

The this() calls the parameterized constructor from default constructor