8. Java Constructor In Depth |

Different Types of Constructor with Examples.

(Private Constructor → Used for Singleton Class)

Q) Why does Constructor have the same name as class name?

A) Easy to identify. Other method names can have the same class name but its not promoted.

Q) Why constructor does not have return type?

A) Java implicitly adds the return type as of the class name.

```
class A{
```

Java implicitly adds return type as

```
public A(){        A(ClassName)
}
}
```

Out of the two methods

```
public A(){
}
```

{This one is the constructor because it has no return type}

Return type is class itself

```
public A A(){
}
```

Q) Why cannot constructor be declared as final / Abstract?

A) The child class which extends its parent class. Inherits all the methods of the parent except the Constructor. Basically you cannot override the method. Which is same as final. So it makes no sense to use final.

Same is the reasoning for Abstract class. Since Constructors cannot be overridden and get implemented in the child class is not allowed. Hence it defeats the purpose of Abstract method.

Q) Why Constructor cannot be static?

A) (✗) Constructors may be used to initialize instance variable. Hence non-static members will also be there. Inside static block, those cannot be accessed.

(✗) Also from the static block, You won't be able to call the ~~static block~~ super class methods in the Constructor

(✗) Constructor is used to create the object. Post construction we allocate static block. So no point making constructor static.

8. Java Constructor In Depth !

Different Types of Constructor with Examples

Constructor chaining

```
public class Calculation{

    String name;
    int empID;

    Calculation(){
        this(10);
    }

    Calculation(int empId){
        this("sj", empId);
    }

    Calculation( String name, int empId){
        this.name = name;
        this.empId = empId;
    }
}
```

Constructor chaining.

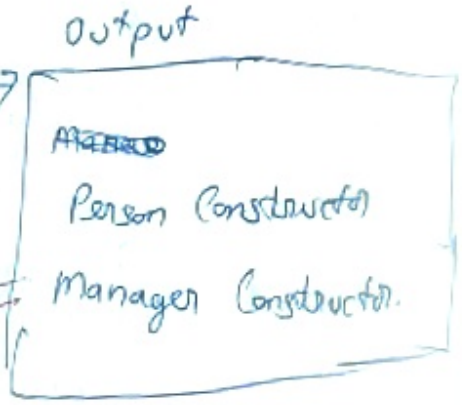
One constructor is calling the next constructor.

Inheritance Constructor Chaining.

```
public class Person {
    super() ---
    Person() { sout("Person constructor");
}

public class Manager extends Person {
    super() ---
    Manager() { sout("Manager constructor"); }
}

Main() {
    super() ---
    Manager m = new Manager();
}
```



This happens because. Java internally calls the super() Constructor of each ~~object~~ class. It is just hidden. Even if we explicitly provide it. It won't make any difference.

Consider → Person had parameterised constructor → Person(int age)

then Manager() {
    super(10); ⇒ Explicit calling is required. Java will not take care.
}