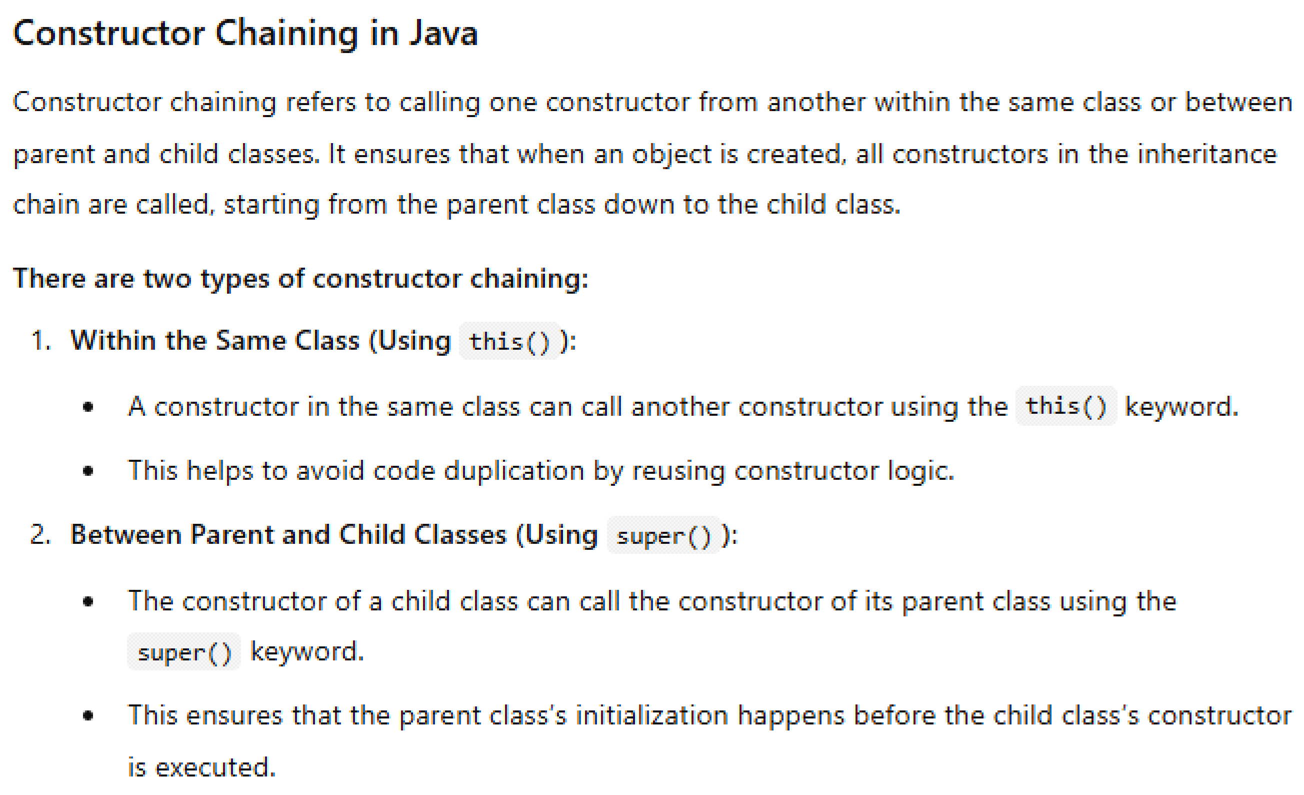
Constructor

27 September 2024 18:49

Constructor:

- *A constructor in Java is a special type of method that is used to initialize objects. It is called when an instance of a class is created and allows the object to set initial values for its fields (instance variables). Constructors do not have a return type, not even void, and their name must exactly match the class name and they are always non static in nature*
- *In our framework, we are using constructors majorly for Initializing Page Objects and Creating Test Data*
- *We can overload the Constructor but can't Overriding it*

- ***Types of Constructor:***
    - **Default Constructor:** *If you do not explicitly define any constructor in a class, Java automatically provides a default constructor with no arguments. The default constructor initializes the object with default values*
    - **Parameterized Constructor**: *You can create a parameterized constructor that accepts arguments to initialize the object with specific values.*
    - **No-Argument Constructor**: *A constructor that does not take any arguments is called a no-argument constructor. If you define this constructor explicitly, it can initialize default values.*



Constructor Chaining in Java

Constructor chaining refers to calling one constructor from another within the same class or between parent and child classes. It ensures that when an object is created, all constructors in the inheritance chain are called, starting from the parent class down to the child class.

There are two types of constructor chaining:

1. **Within the Same Class (Using `this()`):**
    - A constructor in the same class can call another constructor using the `this()` keyword.
    - This helps to avoid code duplication by reusing constructor logic.
2. **Between Parent and Child Classes (Using `super()`):**
    - The constructor of a child class can call the constructor of its parent class using the `super()` keyword.
    - This ensures that the parent class's initialization happens before the child class's constructor is executed.

Example of Constructor Chaining in Java:

1. **Within the Same Class (Using `this()`):**

```java
class Employee {
    String name;
    int age;

    // Constructor 1
    public Employee() {
        this("Unknown", 0); // Calls Constructor 2
    }

    // Constructor 2
    public Employee(String name, int age) {
        this.name = name;
        this.age = age;
    }

    void display() {
        System.out.println("Name: " + name + ", Age: " + age);
    }
}

public class Main {
    public static void main(String[] args) {
        Employee emp = new Employee(); // Calls Constructor 1, which calls Constructor 2
        emp.display();
    }
}
```

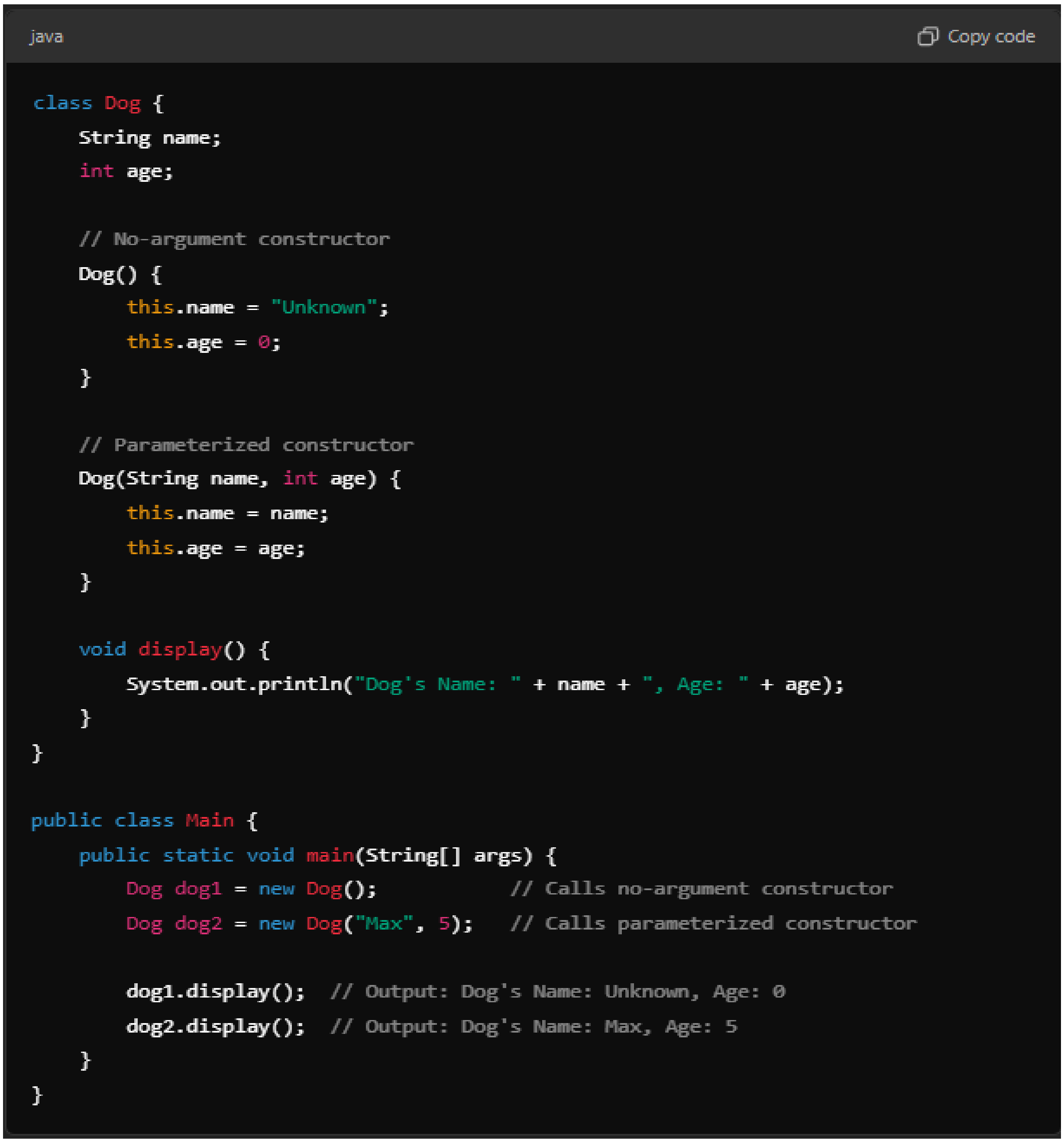
2. **Between Parent and Child Class (Using `super()`):**

java

```java
class Person {
    String name;

    // Constructor 1 in Parent class
    public Person(String name) {
        this.name = name;
    }
}

class Employee extends Person {
    int id;

    // Constructor in Child class
    public Employee(String name, int id) {
        super(name); // Calls the Parent class constructor
        this.id = id;
    }

    void display() {
        System.out.println("Name: " + name + ", ID: " + id);
    }
}

public class Main {
    public static void main(String[] args) {
        Employee emp = new Employee("Alice", 123); // Calls Child class constructor, which
        emp.display();
    }
}
```

Constructor Overloading:

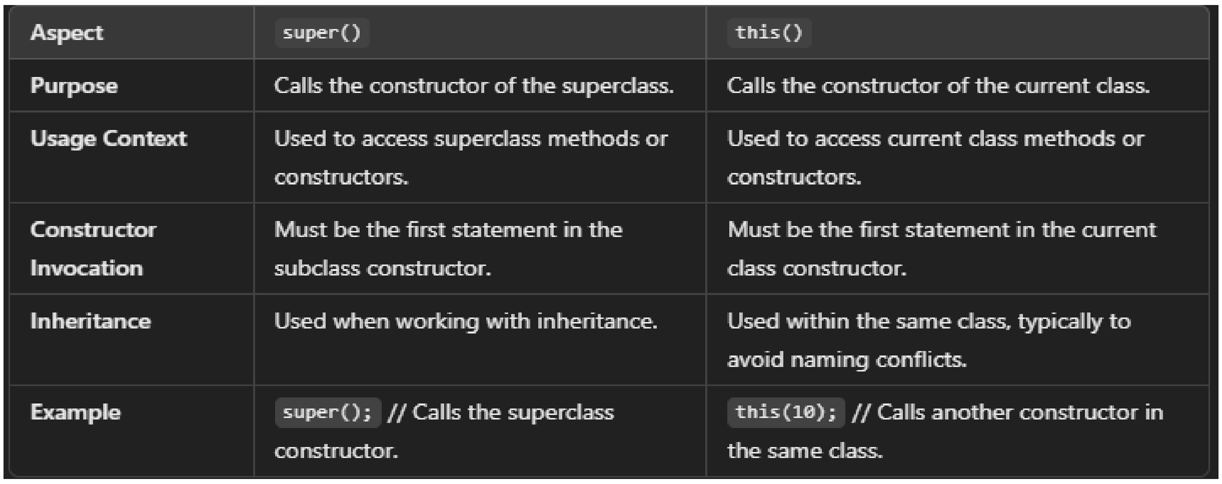
You can define multiple constructors in a class with different parameter lists. This is known as constructor overloading. It allows flexibility in how objects are initialized



```java
class Dog {
    String name;
    int age;

    // No-argument constructor
    Dog() {
        this.name = "Unknown";
        this.age = 0;
    }

    // Parameterized constructor
    Dog(String name, int age) {
        this.name = name;
        this.age = age;
    }

    void display() {
        System.out.println("Dog's Name: " + name + ", Age: " + age);
    }
}

public class Main {
    public static void main(String[] args) {
        Dog dog1 = new Dog();           // Calls no-argument constructor
        Dog dog2 = new Dog("Max", 5);   // Calls parameterized constructor

        dog1.display();  // Output: Dog's Name: Unknown, Age: 0
        dog2.display();  // Output: Dog's Name: Max, Age: 5
    }
}
```

Comparison of super() and this():

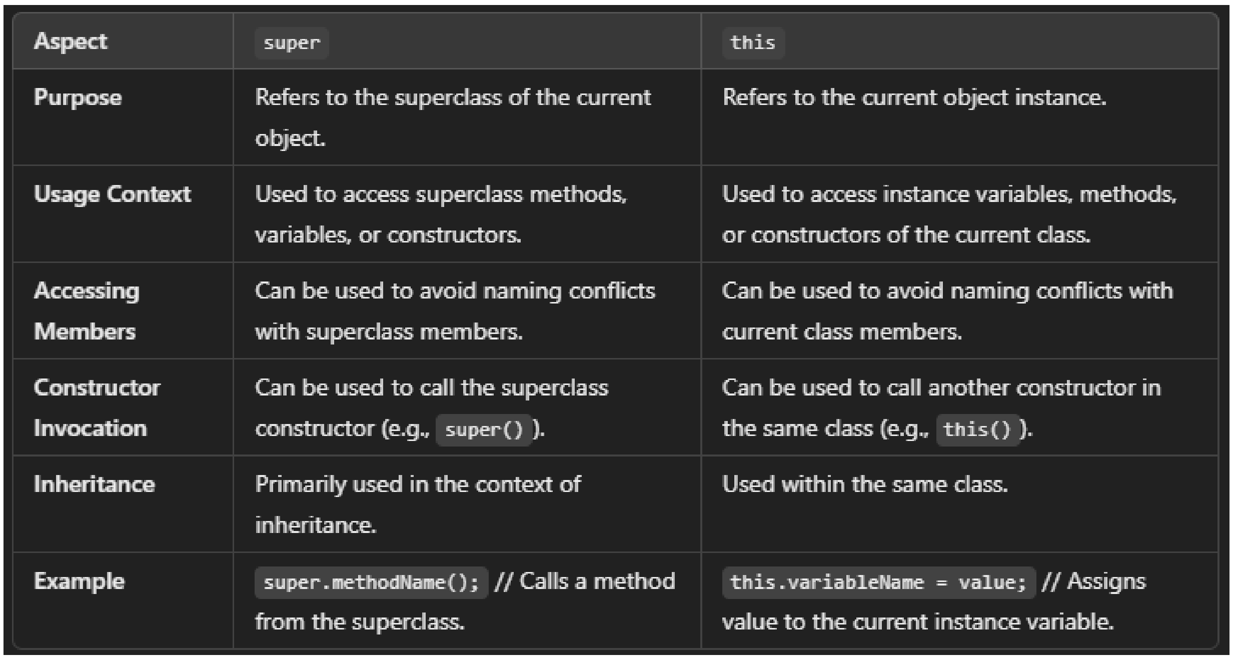
- ***super()*** *is used for accessing and invoking superclass constructors or methods.*
- ***this()*** *is used for invoking other constructors in the same class.*



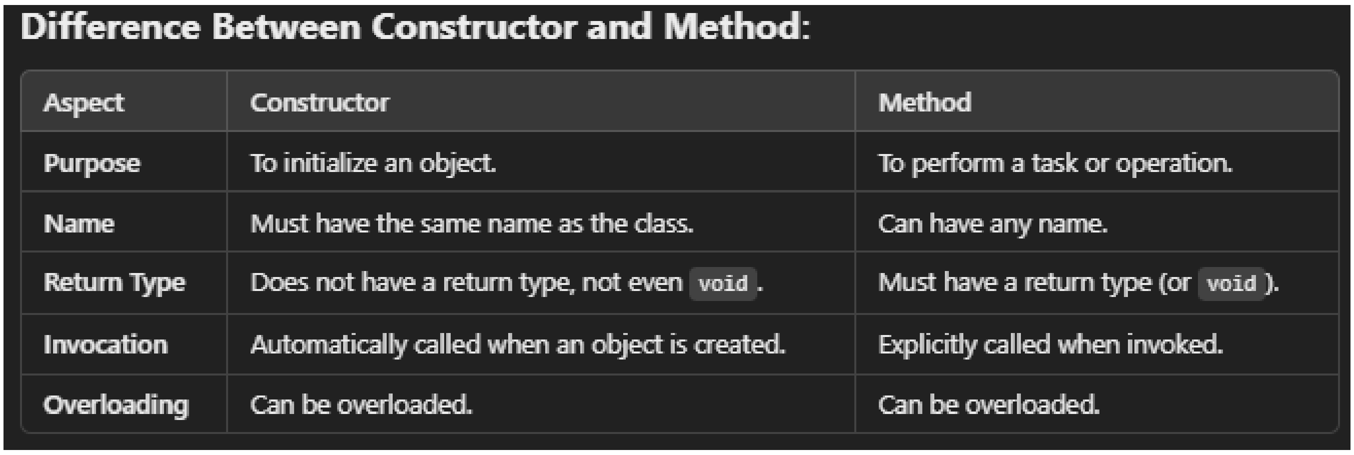
Aspect	`super()`	`this()`
Purpose	Calls the constructor of the superclass.	Calls the constructor of the current class.
Usage Context	Used to access superclass methods or constructors.	Used to access current class methods or constructors.
Constructor Invocation	Must be the first statement in the subclass constructor.	Must be the first statement in the current class constructor.
Inheritance	Used when working with inheritance.	Used within the same class, typically to avoid naming conflicts.
Example	`super();` // Calls the superclass constructor.	`this(10);` // Calls another constructor in the same class.

Comparison of the super and this keywords

- *The super keyword is used to refer to the superclass of the current object, allowing access to its methods and constructors.*
- *The this keyword is used to refer to the current object instance, allowing access to its own methods and constructors.*



Aspect	`super`	`this`
Purpose	Refers to the superclass of the current object.	Refers to the current object instance.
Usage Context	Used to access superclass methods, variables, or constructors.	Used to access instance variables, methods, or constructors of the current class.
Accessing Members	Can be used to avoid naming conflicts with superclass members.	Can be used to avoid naming conflicts with current class members.
Constructor Invocation	Can be used to call the superclass constructor (e.g., `super()`).	Can be used to call another constructor in the same class (e.g., `this()`).
Inheritance	Primarily used in the context of inheritance.	Used within the same class.
Example	`super.methodName();` // Calls a method from the superclass.	`this.variableName = value;` // Assigns value to the current instance variable.



Difference Between Constructor and Method:

Aspect	Constructor	Method
Purpose	To initialize an object.	To perform a task or operation.
Name	Must have the same name as the class.	Can have any name.
Return Type	Does not have a return type, not even `void`.	Must have a return type (or `void`).
Invocation	Automatically called when an object is created.	Explicitly called when invoked.
Overloading	Can be overloaded.	Can be overloaded.