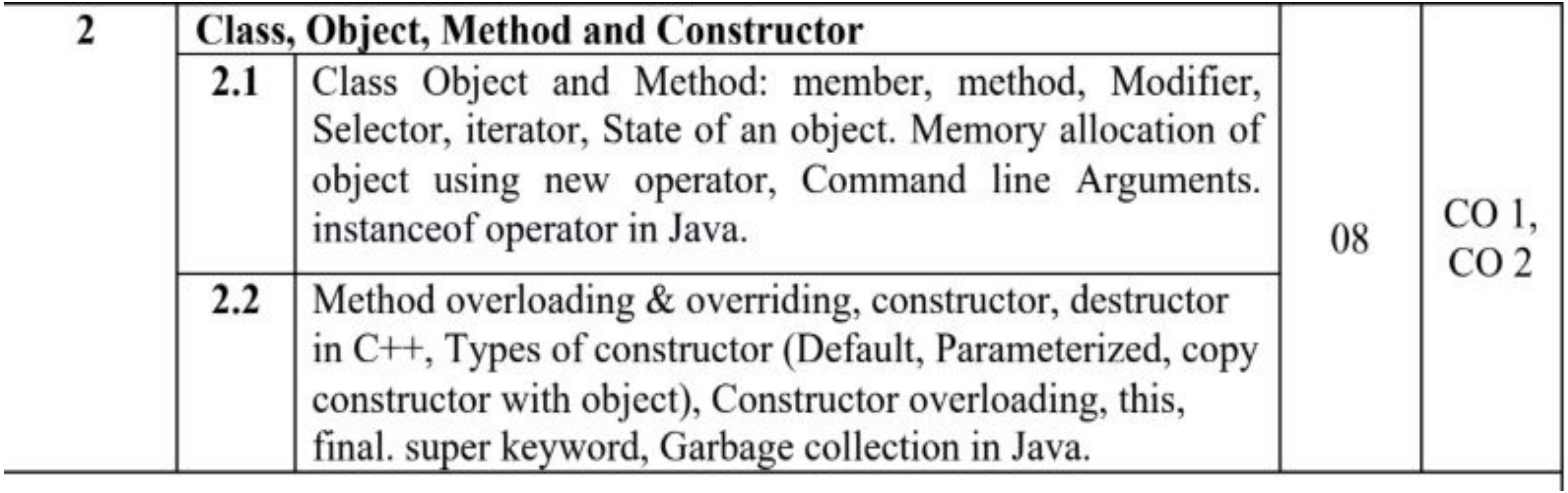
SOMAIYA
VIDYAVIHAR UNIVERSITY
K J Somaiya School of Engineering
(formerly K J Somaiya College of Engineering)

Java Programming –Core Concepts

Dr. Shila Jawale
Asst.Professor, KJSSE

Affilia

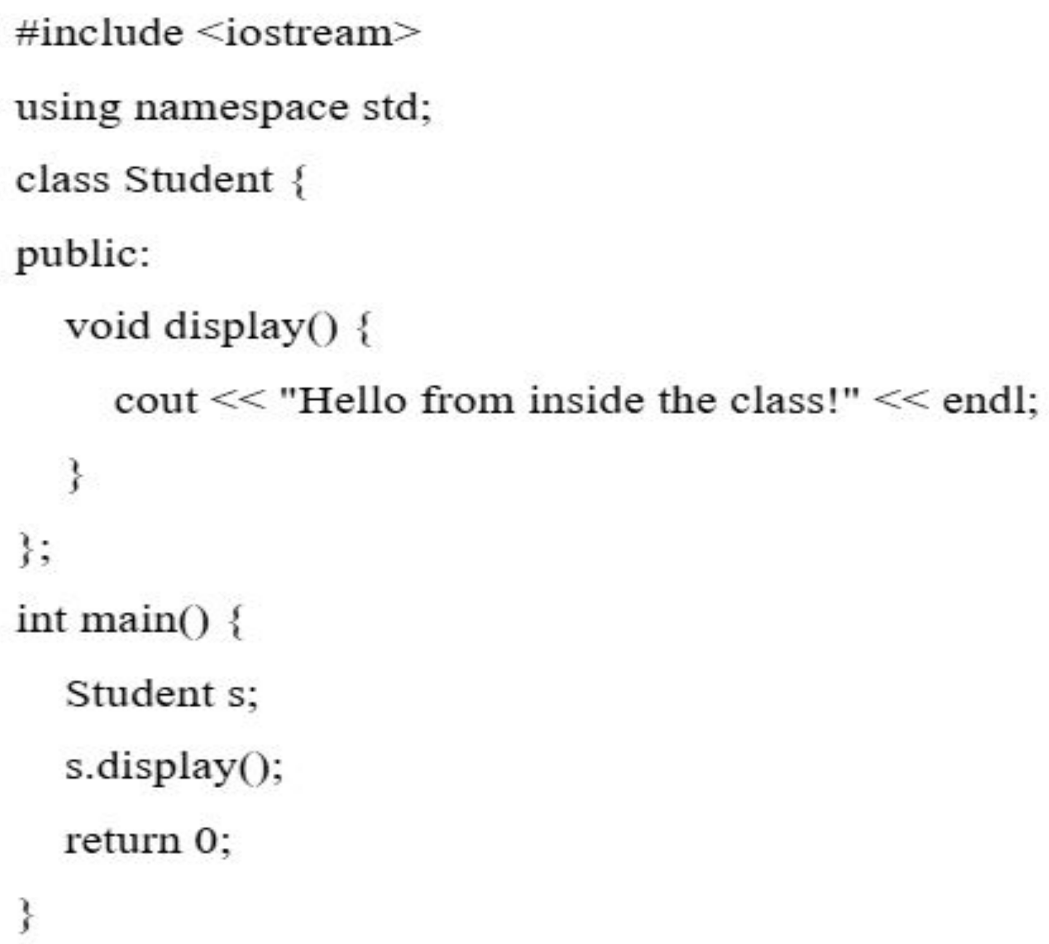


2	**Class, Object, Method and Constructor**		08	CO 1, CO 2
	2.1	Class Object and Method: member, method, Modifier, Selector, iterator, State of an object. Memory allocation of object using new operator, Command line Arguments. instanceof operator in Java.		
	2.2	Method overloading & overriding, constructor, destructor in C++, Types of constructor (Default, Parameterized, copy constructor with object), Constructor overloading, this, final. super keyword, Garbage collection in Java.		

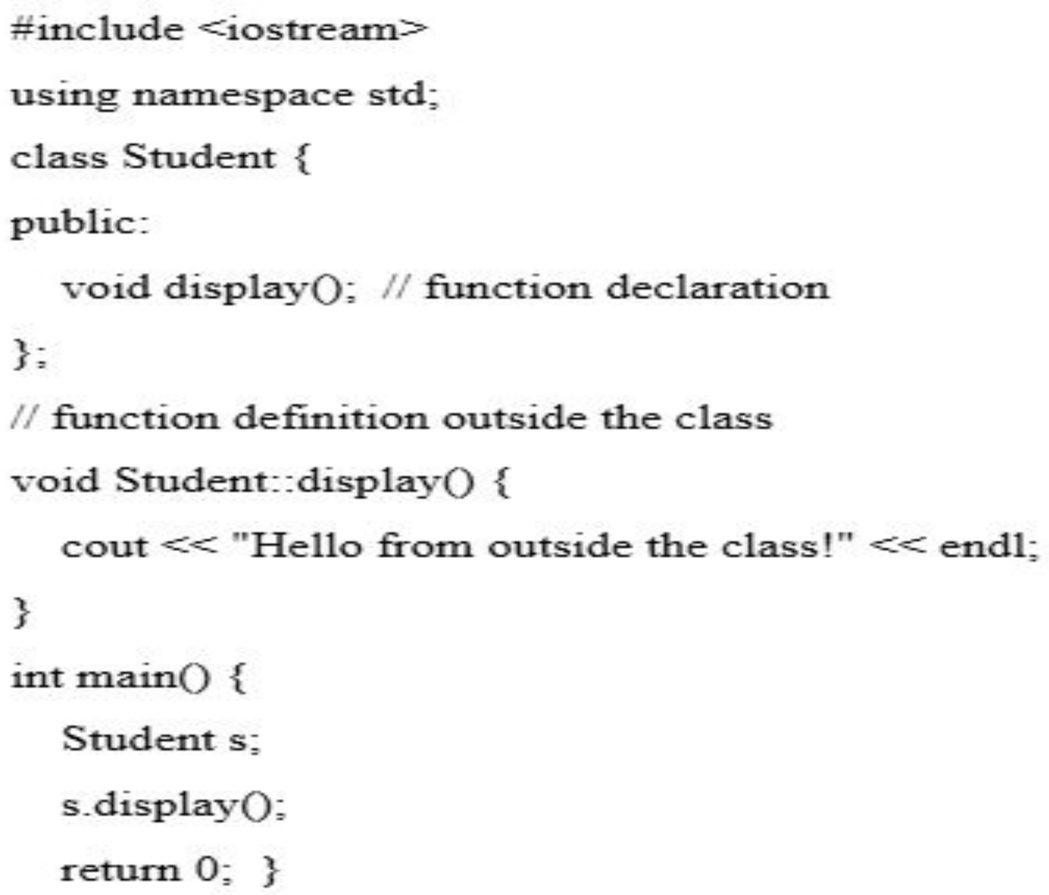
What is a Member Function?

A **member function** is a function that is **defined inside a class** and has access to the class's members (variables and other functions).

EXAMPLE ----C++



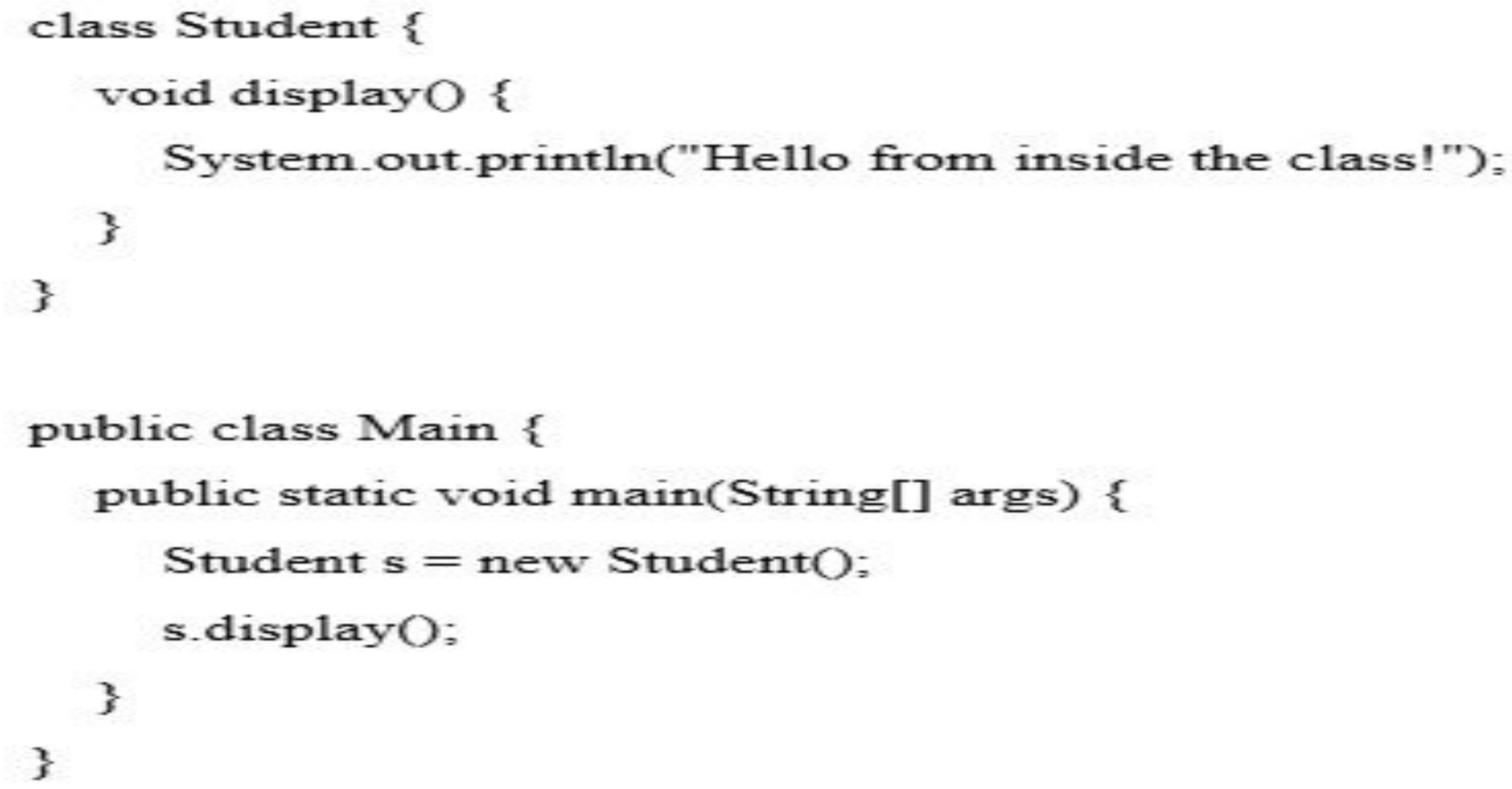
```
#include <iostream>
using namespace std;
class Student {
public:
    void display() {
        cout << "Hello from inside the class!" << endl;
    }
};
int main() {
    Student s;
    s.display();
    return 0;
}
```



```
#include <iostream>
using namespace std;
class Student {
public:
    void display();  // function declaration
};
// function definition outside the class
void Student::display() {
    cout << "Hello from outside the class!" << endl;
}
int main() {
    Student s;
    s.display();
    return 0;  }
```

Java

In Java, all member functions **must be defined inside the class**



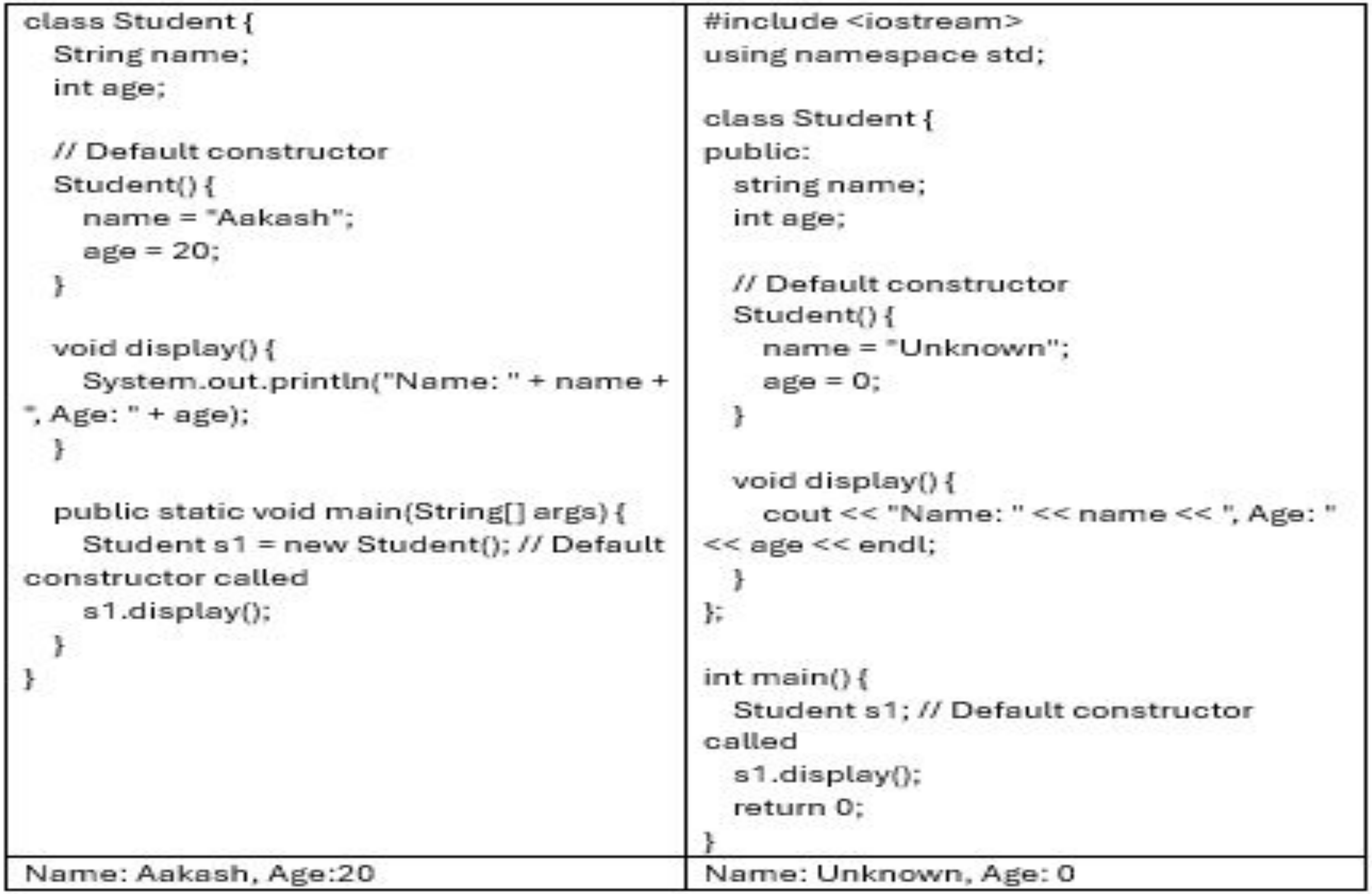
```
class Student {
    void display() {
        System.out.println("Hello from inside the class!");
    }
}

public class Main {
    public static void main(String[] args) {
        Student s = new Student();
        s.display();
    }
}
```

Constructor in C++	Constructor in Java
• Default Constructor • Parameterized Constructor • Copy Constructor (built-in)	• Default Constructor • Parameterized Constructor • Copy Constructor (user-defined)

Common in both : **Constructor Overloading**

Default Constructor

Java	C++



```
class Student {
   String name;
   int age;

   // Default constructor
   Student() {
      name = "Aakash";
      age = 20;
   }

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

   public static void main(String[] args) {
      Student s1 = new Student(); // Default
constructor called
      s1.display();
   }
}
```

```
Name: Aakash, Age:20
```

```
#include <iostream>
using namespace std;

class Student {
public:
   string name;
   int age;

   // Default constructor
   Student() {
      name = "Unknown";
      age = 0;
   }

   void display() {
      cout << "Name: " << name << ", Age: "
<< age << endl;
   }
};

int main() {
   Student s1; // Default constructor
called
   s1.display();
   return 0;
}
```

```
Name: Unknown, Age: 0
```

Problem 1:

Create a class Book for a library system.

- Use a **default constructor** to initialize:
  - title = "Untitled"
  - author = "Unknown"
  - price = 0.0
- Add methods to set and display book details.
- In the main program, create a book object using a default constructor, then later update its information.

Problem 2

Create a class Attendance that stores student attendance records.

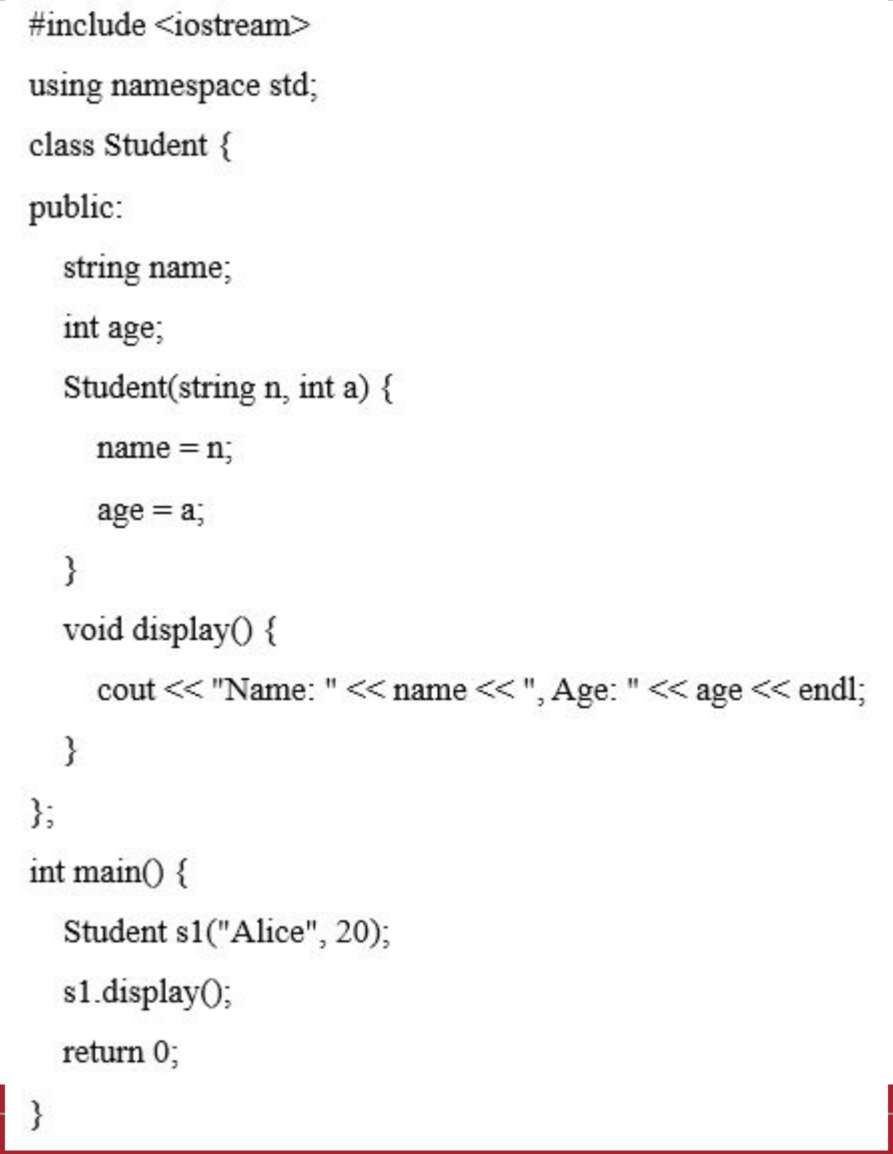
- Default constructor initializes:
  - studentName = "NA"
  - totalClasses = 0
  - attendedClasses = 0
- Method calculatePercentage() to compute attendance.
- Display whether the student meets the minimum attendance requirement (e.g., 75%).

Parameterized Constructor

```
Class A{

A ( para1, para2….)
{

}
```

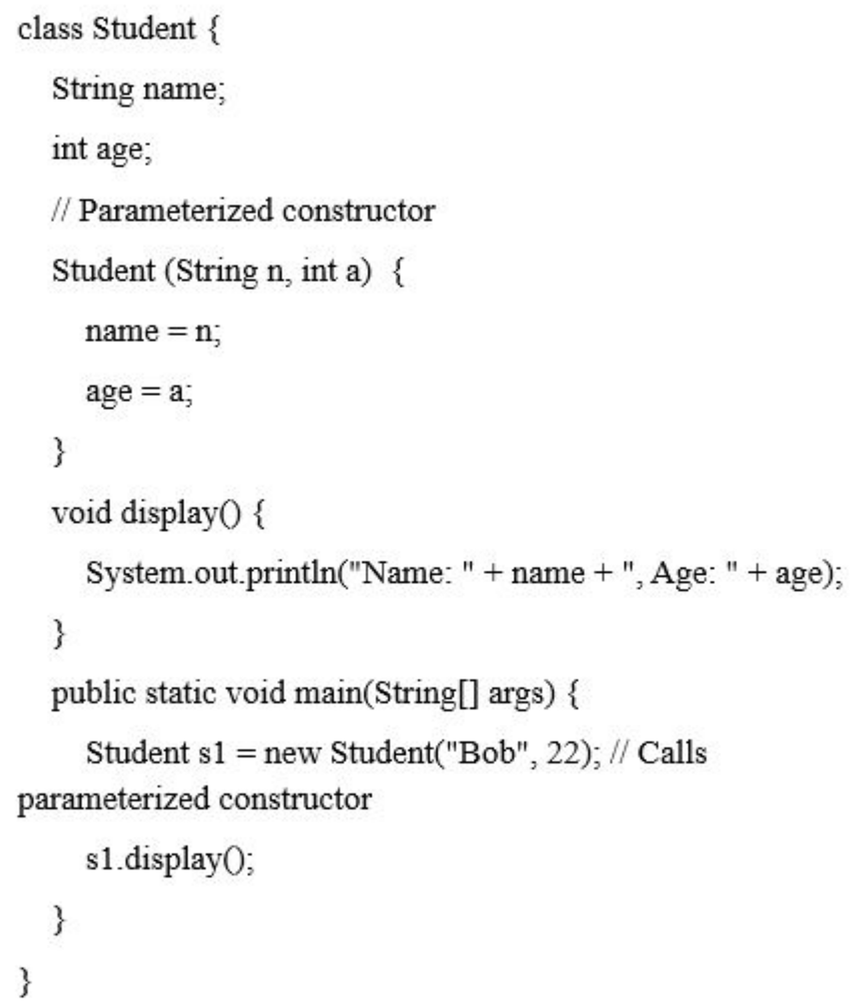
Parameterized Constructor

C++



```
#include <iostream>
using namespace std;
class Student {
public:
    string name;
    int age;
    Student(string n, int a) {
        name = n;
        age = a;
    }
    void display() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};
int main() {
    Student s1("Alice", 20);
    s1.display();
    return 0;
}
```

Java



```
class Student {
    String name;
    int age;
    // Parameterized constructor
    Student (String n, int a)  {
        name = n;
        age = a;
    }
    void display() {
        System.out.println("Name: " + name + ", Age: " + age);
    }
    public static void main(String[] args) {
        Student s1 = new Student("Bob", 22); // Calls
parameterized constructor
        s1.display();
    }
}
```

Problem 1:

Create a class Ticket with attributes movieName, seatNumber, and price.

- Initialize ticket details using a **parameterized constructor**.
- Add a method printTicket() to display booking details.

Problem 2:

Product Inventory System

Create a class Product with attributes productName, price, and quantity.

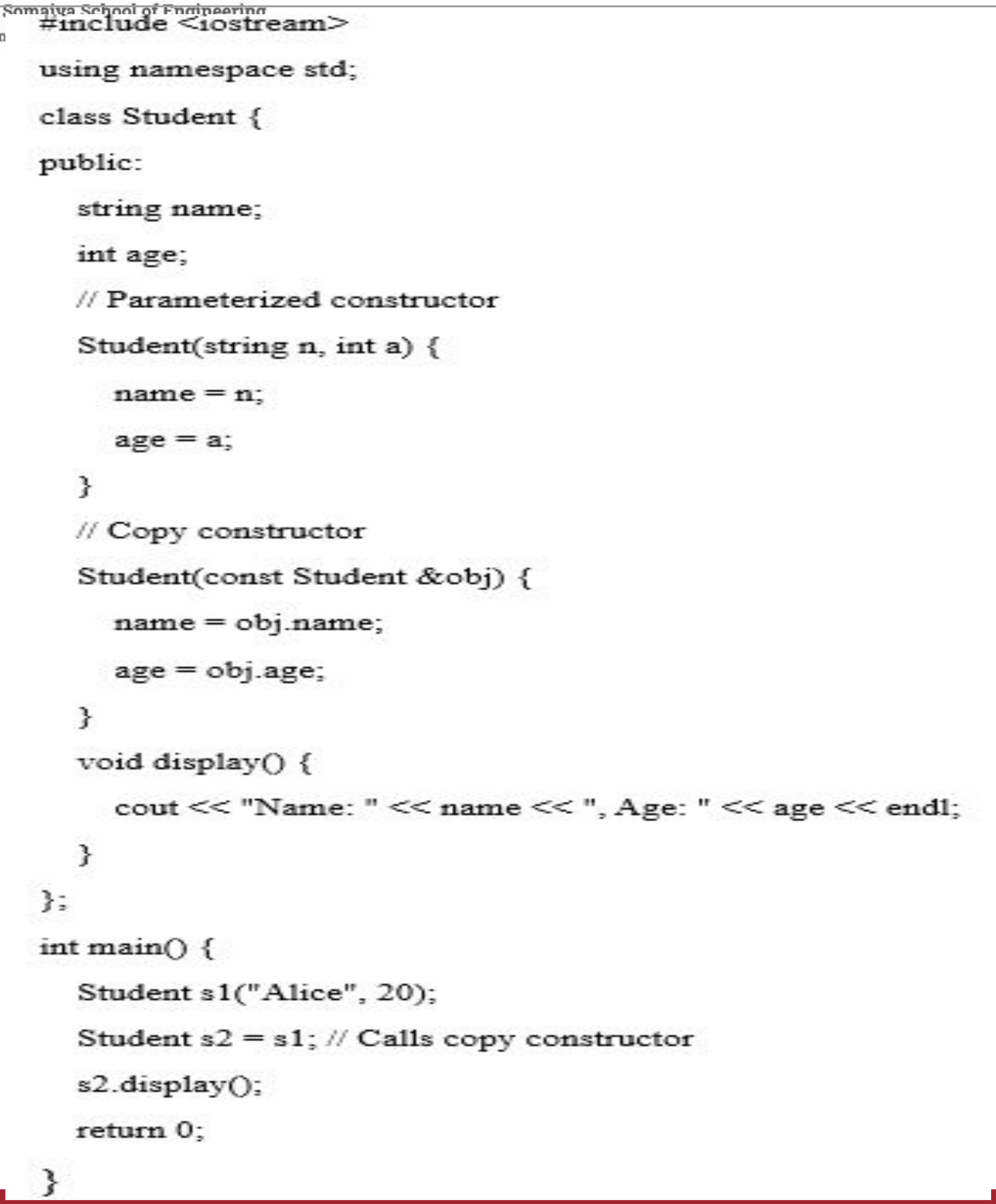
- Use a **parameterized constructor** to initialize all attributes.
- Add a method displayProduct() to print product details.
- Create multiple product objects using the parameterized constructor.

Copy Constructor (built-in)

C++	Java
Definition: A copy constructor creates a new object as a copy of an existing object. **Syntax:** ClassName(const ClassName &obj) { ... } C++ automatically provides a default copy constructor, but you can define your own for customized copying.	**Java does NOT have a built-in copy constructor** like C++. We create it **manually** as a constructor that takes another object of the same class

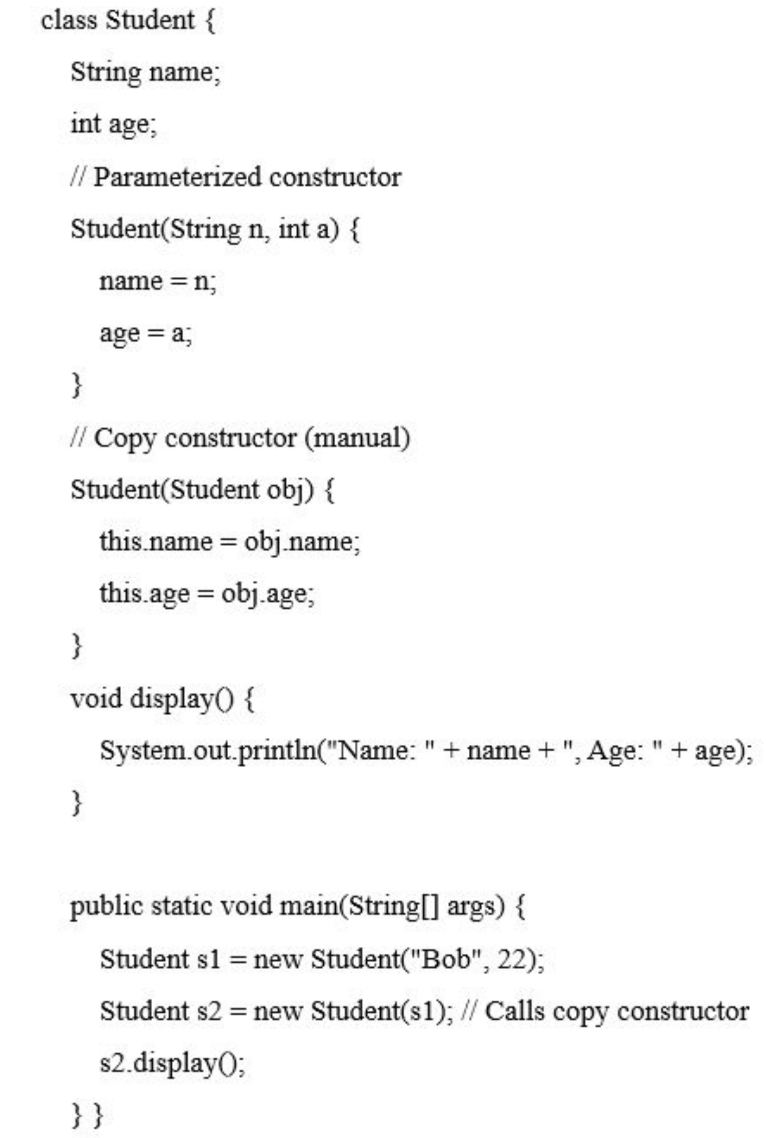
C++	Java

C++



```cpp
#include <iostream>
using namespace std;
class Student {
public:
    string name;
    int age;
    // Parameterized constructor
    Student(string n, int a) {
        name = n;
        age = a;
    }
    // Copy constructor
    Student(const Student &obj) {
        name = obj.name;
        age = obj.age;
    }
    void display() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};
int main() {
    Student s1("Alice", 20);
    Student s2 = s1; // Calls copy constructor
    s2.display();
    return 0;
}
```

Java



```java
class Student {
  String name;
  int age;
  // Parameterized constructor
  Student(String n, int a) {
    name = n;
    age = a;
  }
  // Copy constructor (manual)
  Student(Student obj) {
    this.name = obj.name;
    this.age = obj.age;
  }
  void display() {
    System.out.println("Name: " + name + ", Age: " + age);
  }

  public static void main(String[] args) {
    Student s1 = new Student("Bob", 22);
    Student s2 = new Student(s1); // Calls copy constructor
    s2.display();
  } }
```

Vehicle Registration System

Problem: Create a Vehicle class with attributes ownerName, vehicleNumber, and type.

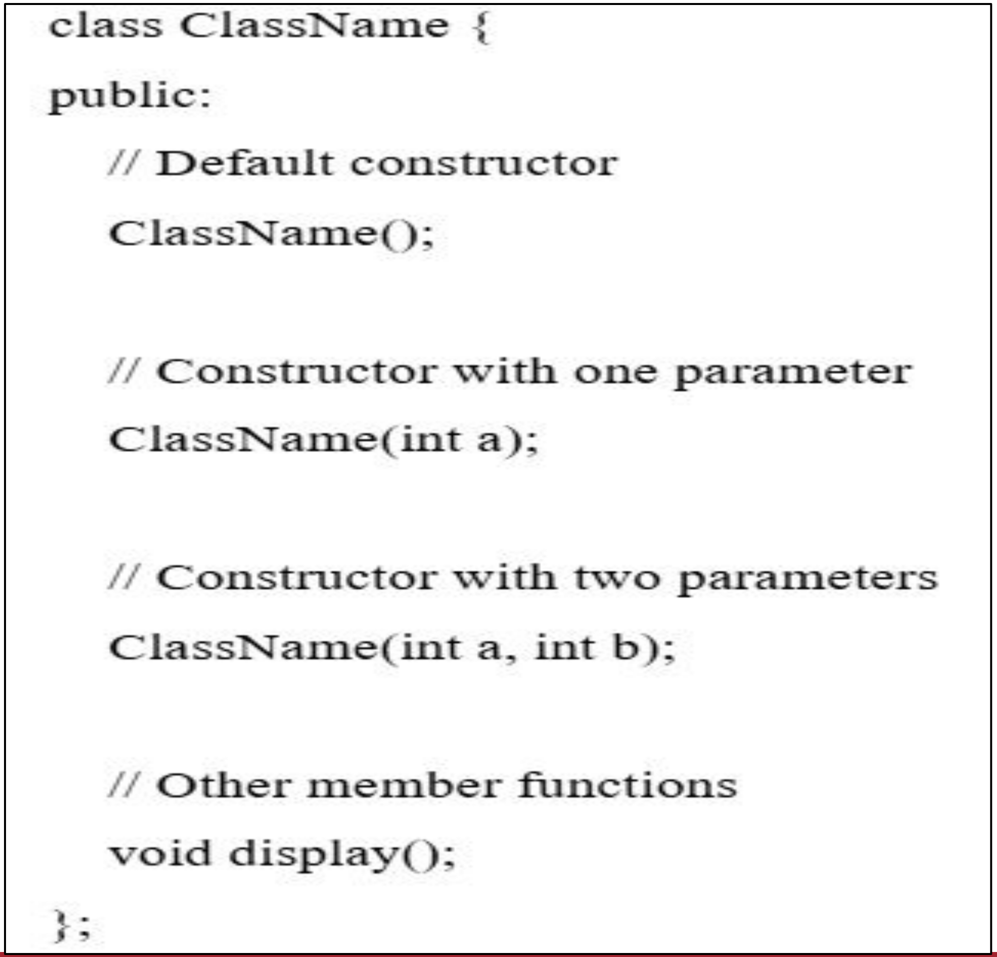
- Implement:
  - A parameterized constructor to initialize values.
  - A copy constructor to duplicate vehicle records.
- **Task:** Copy an existing vehicle’s data to a new object (like when transferring ownership).

Employee Record Backup

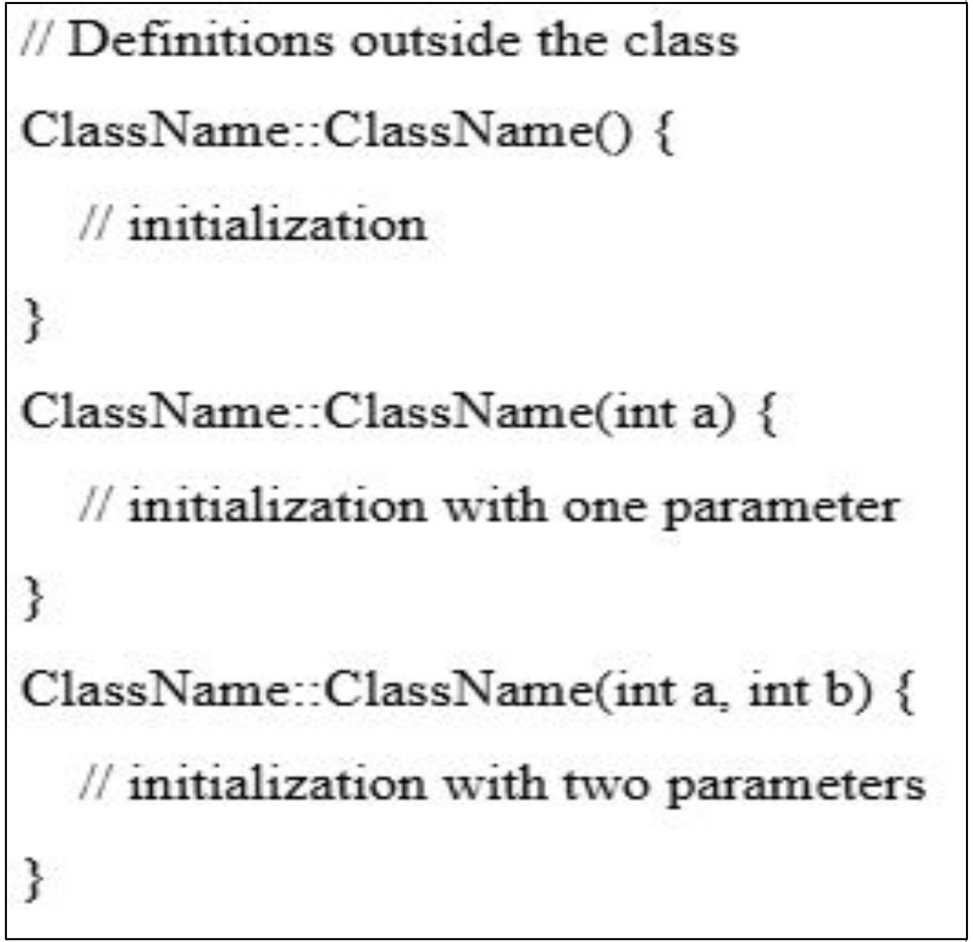
- **Problem:** Create a class `Employee` with `name`, `designation`, and `salary`.
- Add:
  - A parameterized constructor.
  - A copy constructor to create a backup of an employee record.
- **Task:** Create an employee object and another using a copy constructor to store a backup.

Constructor overloading

Constructor overloading means having **more than one constructor in a class** with **different parameters**. It allows you to create objects in different ways.



```
class ClassName {
public:
    // Default constructor
    ClassName();

    // Constructor with one parameter
    ClassName(int a);

    // Constructor with two parameters
    ClassName(int a, int b);

    // Other member functions
    void display();
};
```



```
// Definitions outside the class
ClassName::ClassName() {
    // initialization
}
ClassName::ClassName(int a) {
    // initialization with one parameter
}
ClassName::ClassName(int a, int b) {
    // initialization with two parameters
}
```

Employee Salary System

Employee Record Creation Using Constructor Overloading

Problem Statement:

Develop a class Employee to manage:

- empId, name, designation, and salary

Overload constructors for:

1. **Default values**
2. **ID and name only**
3. **D, name, and salary**

Add:

- A display() method

Distance Converter

Title:

Distance Class Using Constructor Overloading

Problem Statement:

Create a class Distance with data members kilometers and meters.

- Overload constructors to:
  Set values to 0
  Set using kilometers only
  Set using both kilometers and meters

Add a function **displayDistance()** to show total distance in meters

Destructor

What is a Destructor?

A **destructor** is a special function that is called **automatically when an object is destroyed**. It is used to **release resources** like memory, files, or database connections.

Destructor in C++

- **Features:**

- Has the **same name** as the class preceded by a ~ (tilde)
- **No return type**, not even `void`
- **Called automatically** when an object goes out of scope
- Mainly used to **free dynamic memory** (allocated using `new`)

```
#include <iostream>
using namespace std;

class Demo {
public:
    Demo() {
        cout << "Constructor called!" << endl;
    }

    ~Demo() {
        cout << "Destructor called!" << endl;
    }
};

int main() {
    Demo d;  // Constructor is called here
    // Destructor will be called automatically at the end of main()
    return 0;
}
```

Destructor in Java? Not Exactly

Java **doesn't have destructors** like C++. Instead, Java uses:

✅ Garbage Collector (GC)

- Automatically reclaims memory when no references to an object remain.

✅ finalize() Method (Deprecated since Java 9)

- Was once used as a **destructor-like method**, but is now **discouraged** and considered **unsafe**.

Today Agenda (5/8/25)

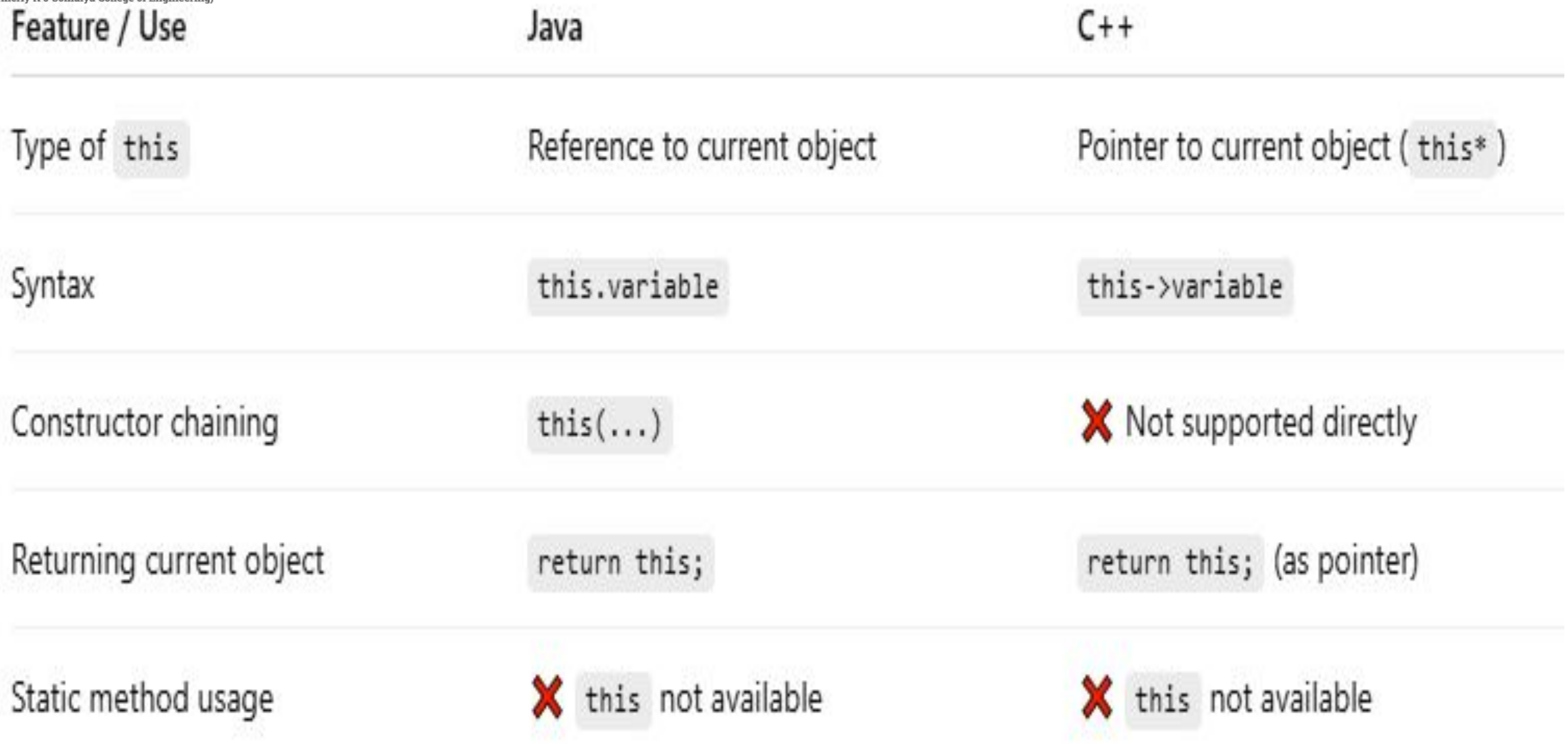
- Examples on VSCode both java and C++
- Parameterized constructor
- Constructor overloading
- Copy constructor
- Use of this and Super
- Dynamically accepting the data
- Array basics
- Quiz

“this”keyword

C++	Java
Uses of `this` in C++: 1. To **access current object's members** 2. To **return the current object** 3. To **resolve naming conflicts**	**Uses of `this` in Java:** 1. To **refer to current class instance variables** 2. To **invoke current class methods or constructors** 3. To **pass the current object as a parameter** 4. To **return the current object**

C++	Java

#include <iostream>
using namespace std;
 	
class Student {
    int id;
    string name;
 	
public:
    Student(int id, string name) {
        this->id = id;   // 'this' pointer used to refer to current object's variable
 	
 this->name = name;
    }
	class Student {
	    int id;
	    String name;
	
	    Student(int id, String name) {
	        this.id = id; // 'this' distinguishes between instance and local variables
	        this.name = name;
	    }



Feature / Use	Java	C++
Type of `this`	Reference to current object	Pointer to current object (`this*`)
Syntax	`this.variable`	`this->variable`
Constructor chaining	`this(...)`	❌ Not supported directly
Returning current object	`return this;`	`return this;` (as pointer)
Static method usage	❌ `this` not available	❌ `this` not available

Constructor Chaining

Constructor chaining is the process of calling **one constructor from another constructor** within the same class or from a **parent class constructor**.

Constructor Chaining in Java

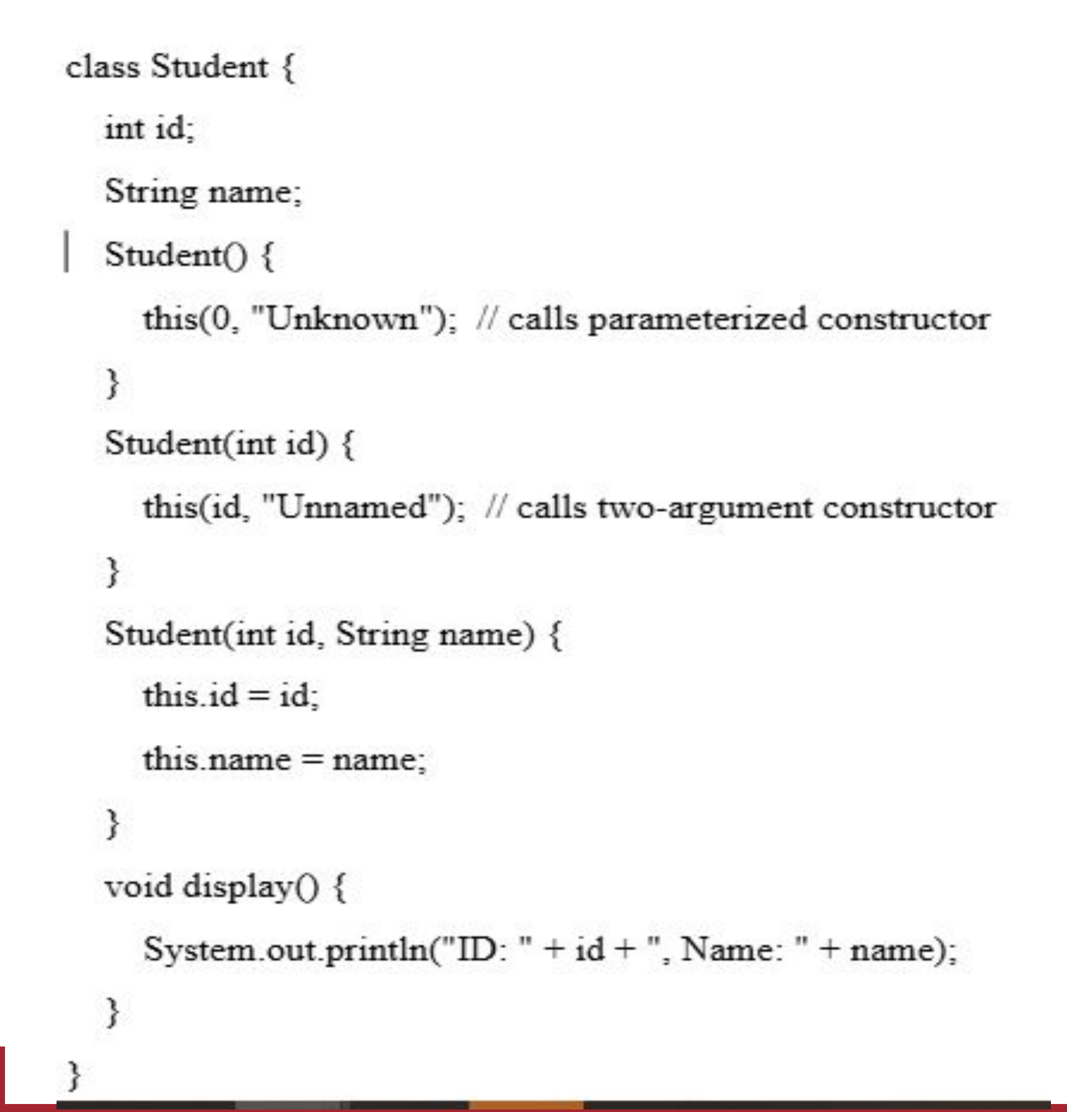
Java supports constructor chaining using:

- **this()** → Calls another constructor **in the same class**
- **super()** → Calls a constructor from the **parent class**

Rules:

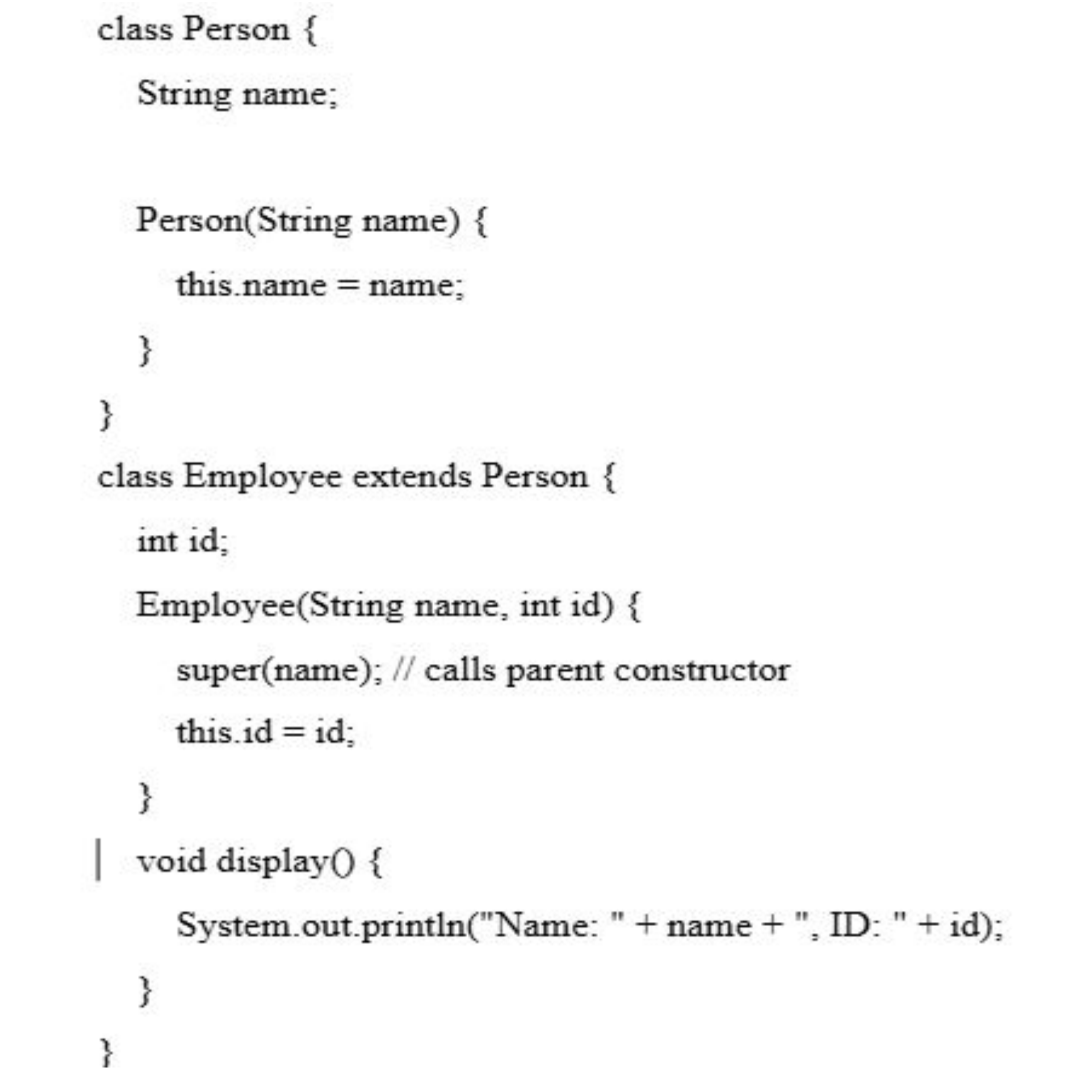
- **`this()` or `super()` must be the first statement** in a constructor.
- Only one can be used at a time **(either `this()` or `super()`).**

Java (this keyword)



```
class Student {
   int id;
   String name;
   Student() {
      this(0, "Unknown");  // calls parameterized constructor
   }
   Student(int id) {
      this(id, "Unnamed");  // calls two-argument constructor
   }
   Student(int id, String name) {
      this.id = id;
      this.name = name;
   }
   void display() {
      System.out.println("ID: " + id + ", Name: " + name);
   }
}
```

Java(Super keyword)



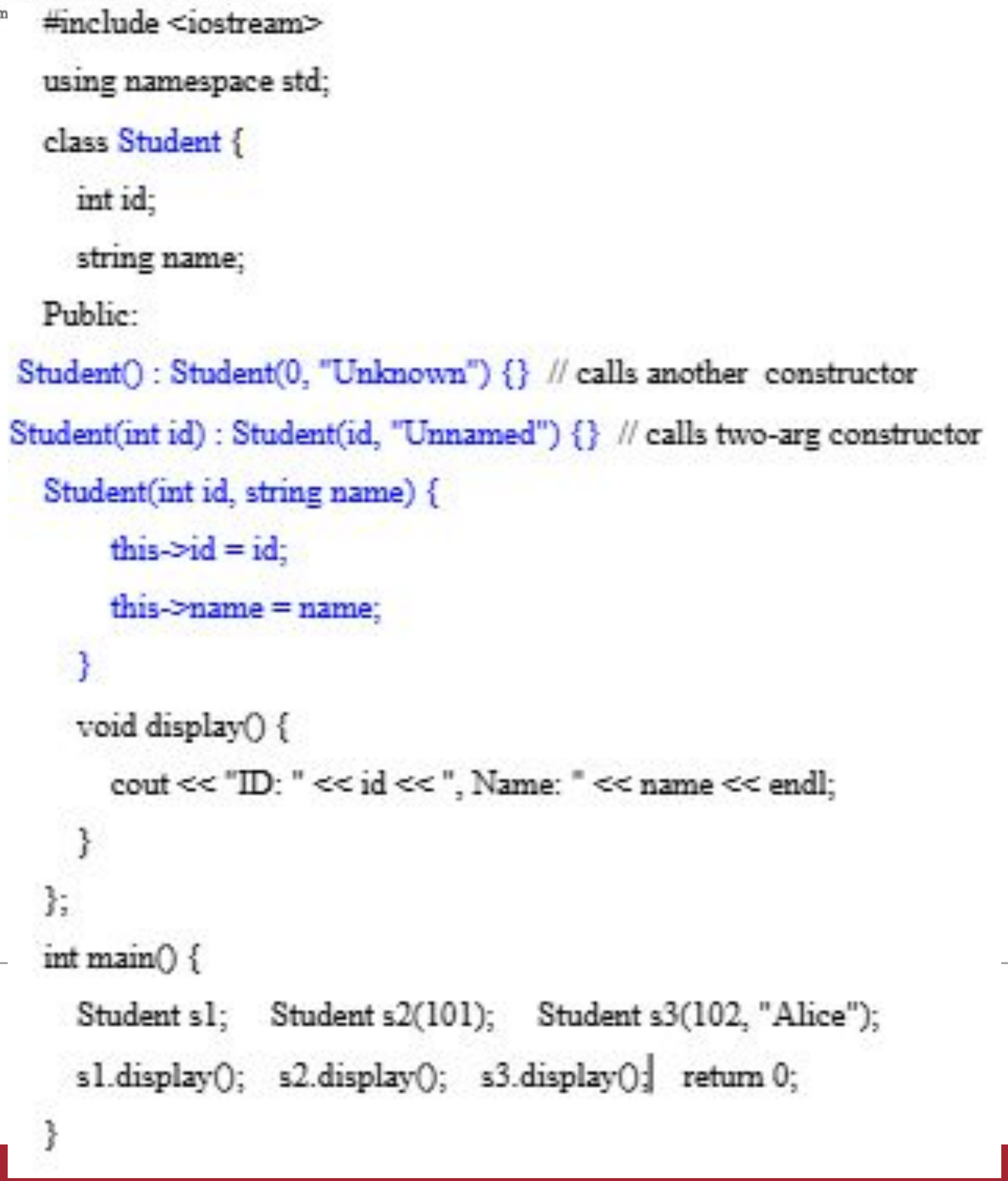
```
class Person {
   String name;

   Person(String name) {
      this.name = name;
   }
}
class Employee extends Person {
   int id;
   Employee(String name, int id) {
      super(name); // calls parent constructor
      this.id = id;
   }
   void display() {
      System.out.println("Name: " + name + ", ID: " + id);
   }
}
```

Constructor Chaining in C++

C++ **does not have a this() or super() keyword**, but it supports chaining through:

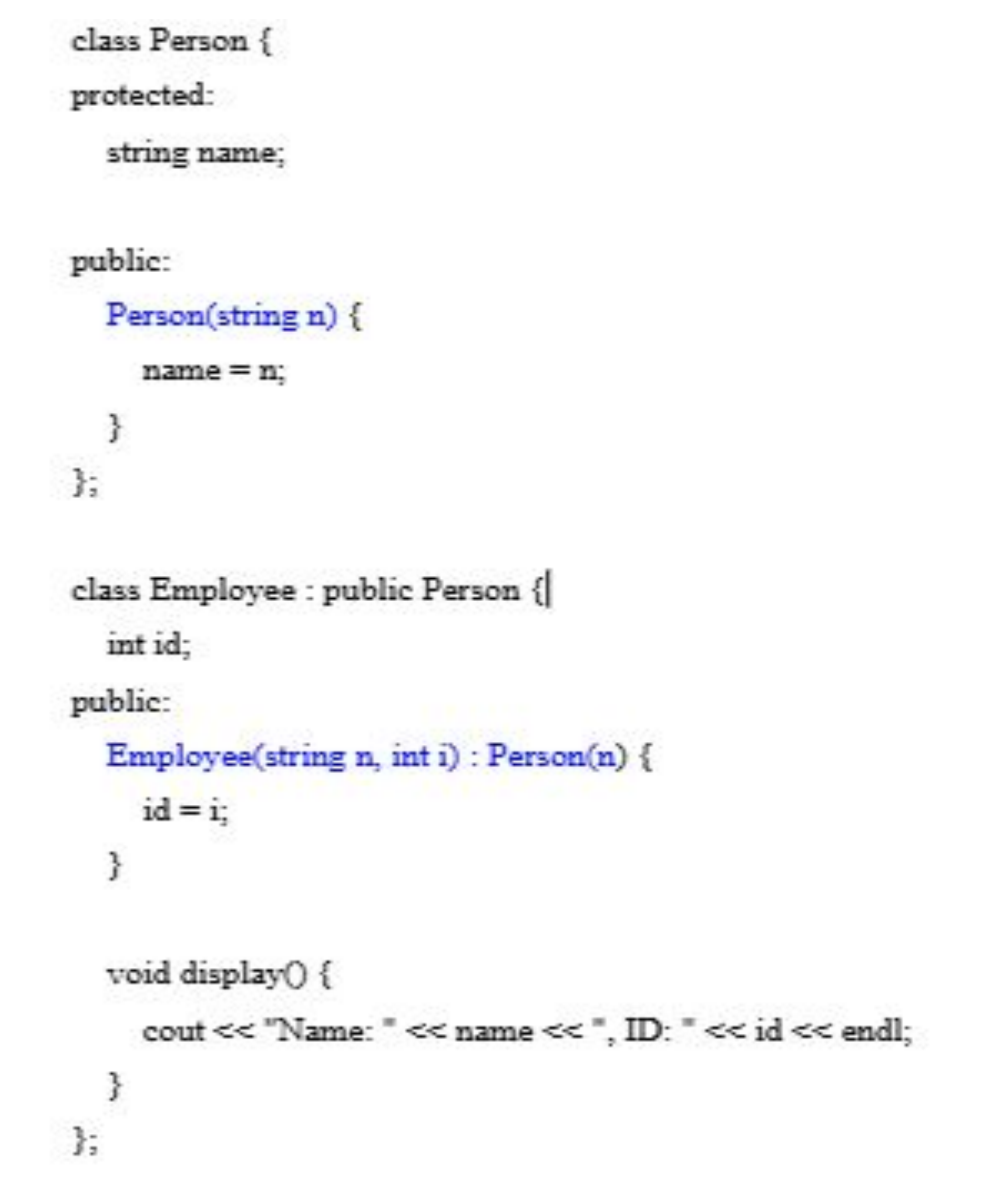
- **Constructor initializer list**
- **Calling base class constructor in derived class**

C++ (Chaining using initializer list)

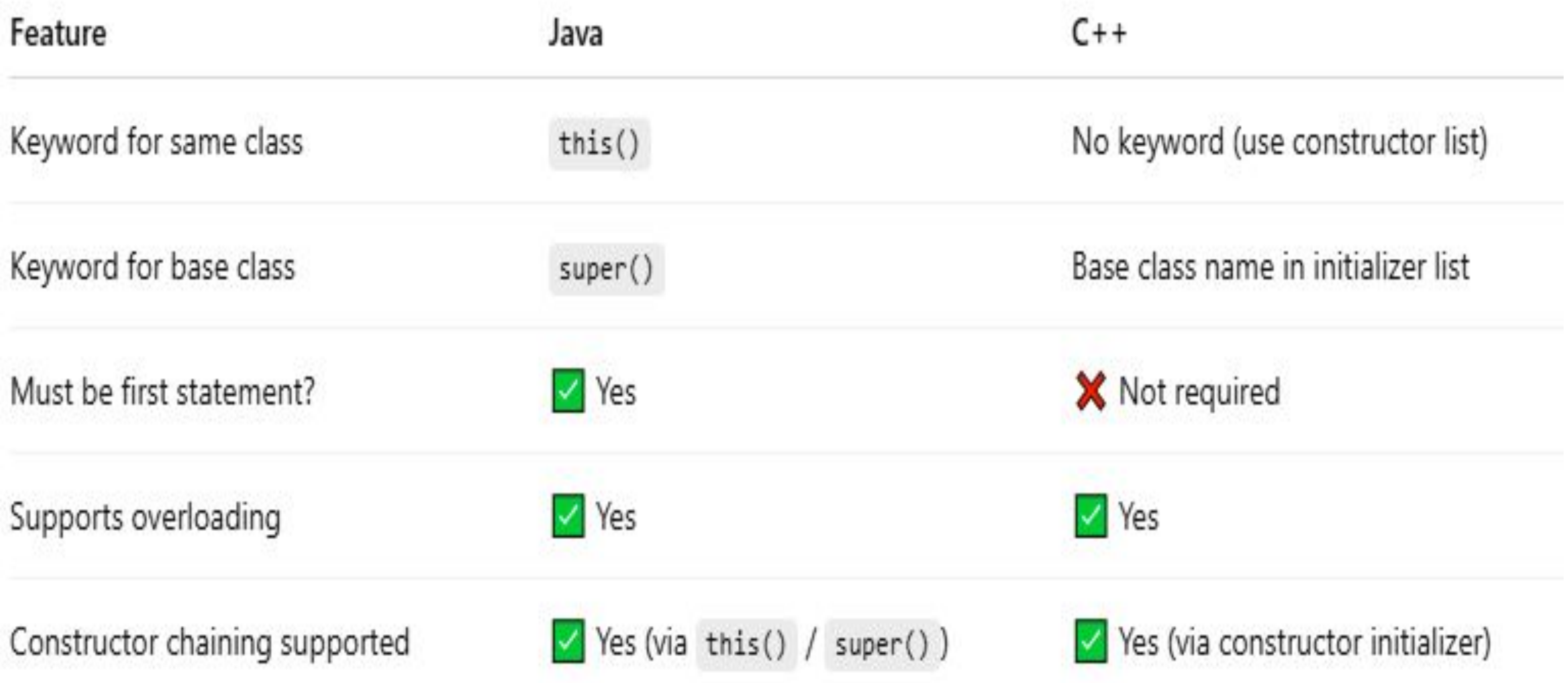


```
#include <iostream>
using namespace std;
class Student {
   int id;
   string name;
Public:
Student() : Student(0, "Unknown") {}  // calls another  constructor
Student(int id) : Student(id, "Unnamed") {}  // calls two-arg constructor
   Student(int id, string name) {
        this->id = id;
        this->name = name;
     }
   void display() {
        cout << "ID: " << id << ", Name: " << name << endl;
     }
};
int main() {
   Student s1;    Student s2(101);    Student s3(102, "Alice");
   s1.display();   s2.display();   s3.display();   return 0;
}
```

C++ (Base class constructor chaining)



```
class Person {
protected:
   string name;

public:
   Person(string n) {
      name = n;
   }
};

class Employee : public Person {
   int id;
public:
   Employee(string n, int i) : Person(n) {
      id = i;
   }

   void display() {
      cout << "Name: " << name << ", ID: " << id << endl;
   }
};
```



Feature	Java	C++
Keyword for same class	`this()`	No keyword (use constructor list)
Keyword for base class	`super()`	Base class name in initializer list
Must be first statement?	✅ Yes	❌ Not required
Supports overloading	✅ Yes	✅ Yes
Constructor chaining supported	✅ Yes (via `this()` / `super()`)	✅ Yes (via constructor initializer)

Expt no 3:

To implement a student result processing system in C++ using **classes and objects**, focusing on encapsulation, constructors, and member functions

https://wayground.com/admin/quiz/68909a576e3fec866edc521b?at=68909ede4a0d4a50016faab6&MCQ_saved=true