ST. JOSEPH'S
COLLEGE OF ENGINEERING
AND TECHNOLOGY,
- PALAI -

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

24SJPBCST304 - OBJECT ORIENTED PROGRAMMING

Prof. Sarju S

1 August 2025

Module 2

- **Polymorphism :-** Method Overloading, Recursion**.** Static Members, Final Variables, Inner Classes
- **Inheritance -** Super Class, Sub Class, Types of Inheritance, The ***super*** keyword, protected Members, Calling Order of Constructors. Method Overriding, Dynamic Method Dispatch, Using ***final*** with Inheritance.

Overloading

Overloading

- Same name for different methods/constructors in class as long as their parameter declarations are different.
- Constructor Overloading
- Method Overloading

Constructor Overloading

- More than one constructor with different parameters list, in such a way so that each constructor performs a different task.

```java
public class ConstructorOverloadingDemo {

    public static void main(String[] args) {
        Student student1 = new Student();
        //Calling Constructor with no arguments
        System.out.println("Student 1 details");
        System.out.println("Department : "+student1.department);

        Student student2 = new Student(12,"Robin Sharma");
        //calling constructor with 2 arguments
        System.out.println("Student 2 details");
        System.out.println("Roll Number : "+student2.rollNumber);
        System.out.println("Name : "+student2.name);

        //calling constructor with 3 arguments
        Student student3 = new Student(13,"Praveen","ME");
        System.out.println("Student 3 details");
        System.out.println("Roll Number : "+student3.rollNumber);
        System.out.println("Name = "+student3.name);
        System.out.println("Department : "+student1.department);
        System.out.println();

    }

}
```

```java
class Student{
    int rollNumber;
    String name;
    String department;
    //No argument constructor
    Student(){
        this.department ="CSE";
    }
    //parameterised constructor with two argument
    Student(int rollNumber, String name){
        this.rollNumber = rollNumber;
        this.name = name;
        department = "CSE";
    }
    //parameterised constructor with three argument
    Student(int rollNumber, String name, String department){
        this.rollNumber = rollNumber;
        this.name = name;
        this.department = department;
    }
}
```

Constructor Chaining

- When A constructor calls another constructor of same class then this is called constructor chaining.
- Constructors with fewer arguments should call those with more

Beginnersbook.com

```
public class MyClass{
   ....
   MyClass( ) {
       this("BeginnersBook.com");
   }
   MyClass(String s) {
       this(s, 6);
   }
   MyClass(String s, int age) {
       this.name =s;
       this.age = age;
   }
   public static void main(String args[]) {
       MyClass obj = new MyClass();
       ....
   }
}
```

Method Overloading

- Two or more methods can have same name inside the same class if they accept different arguments. This feature is known as method overloading.
- Method overloading is achieved by either:
  - changing the number of arguments.
  - changing the datatype of arguments.
  - Changing the order of arguments.
- Method overloading is not possible by changing the return type of methods.

Method Overloading - changing the number of arguments

```java
public class MethodOverloadingDemo {

    public static void main(String[] args) {
        Add add = new Add();
        add.add(10, 20);
        add.add(10, 20, 30);

    }

}
```

```java
class Add{
    void add(int number1, int number2) {
        int sum = number1 + number2;
        System.out.println("Sum of two numbers");
        System.out.println(number1+" + "+number2+" = "+sum);
    }
    void add(int number1,int number2, int number3) {
        int sum = number1 + number2+number3;
        System.out.println("Sum of three numbers");
        System.out.println(number1+" + "+number2+" + "+number3+" = "+sum);
    }
}
```

Output

```
Sum of two numbers
10 + 20 = 30
Sum of three numbers
10 + 20 + 30 = 60
```

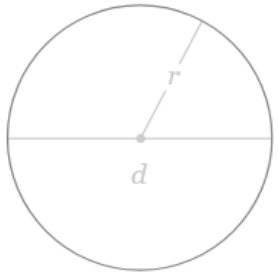
Method Overloading - changing the type of arguments

Solve for
area

$$A = \pi r^2$$

r Radius

Enter value

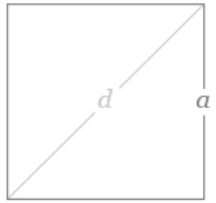


Square
Solve for area

$$A = a^2$$

a Side

Enter value



Method Overloading - changing the type of arguments

```java
class Area{
    void calculateArea(int a) {
        int area = a*a;
        System.out.println("Area of the square= "+ area);
    }
    void calculateArea(float r) {
        float area = 3.14f*r*r;
        System.out.println("Area of Circle: "+area);
    }
```

```java
public class MethodOverloadingDemo2 {

    public static void main(String[] args) {
        Area area = new Area();
        //calculates area of square
        area.calculateArea(10);
        //calculate area of circle
        area.calculateArea(10f);

    }

}
```

Output

```
Area of the square= 100
Area of Circle: 314.0
```

Test your Knowledge #2

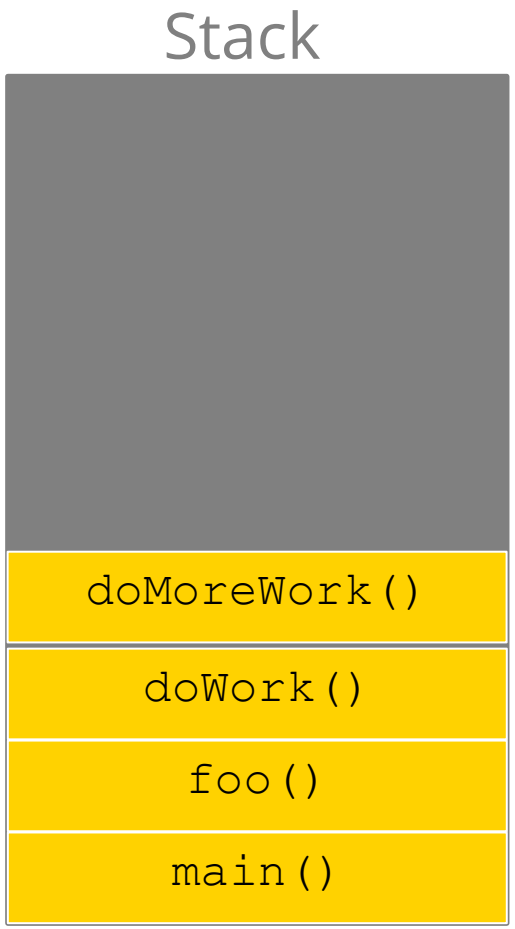
Recursion

Recursion

- The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function.

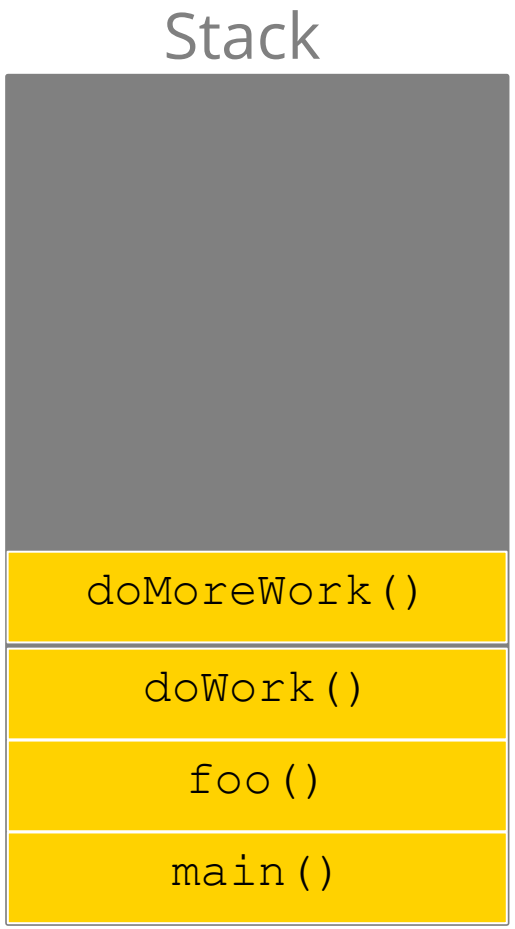
Recursion

```
main{
      foo();
}
foo{
      doWork();
}
doWork{
      doMoreWork();
}
```



Recursion

```
main{
      foo();
}
foo{
      doWork();
      next statement;
}
doWork{
      doMoreWork();
}
```



Recursion

```
main{
        foo();
}
foo{
        foo();
        foo();
        foo();
}
```

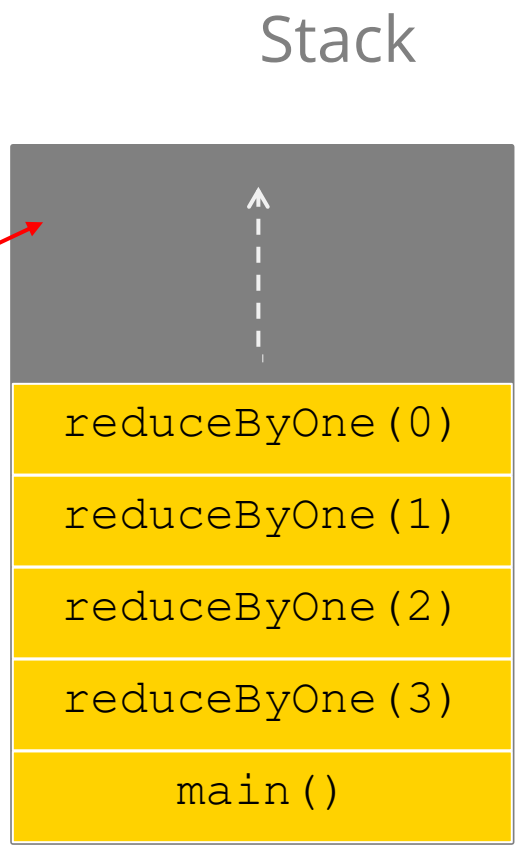
Stack

foo()
foo()
foo()
main()

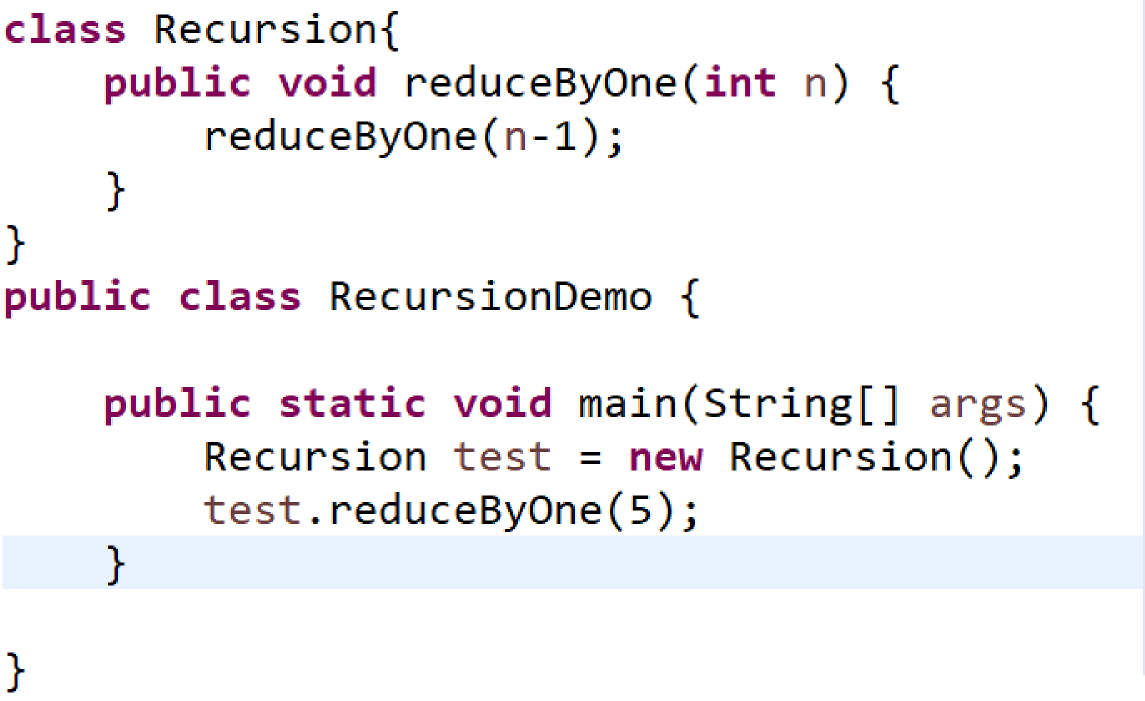
Recursion

```
main{
      reduceByOne(3);
}
reducebyOne(int n){
      reduceByOne(n-1);
}
```

No Base condition
Will cause stack overflow error



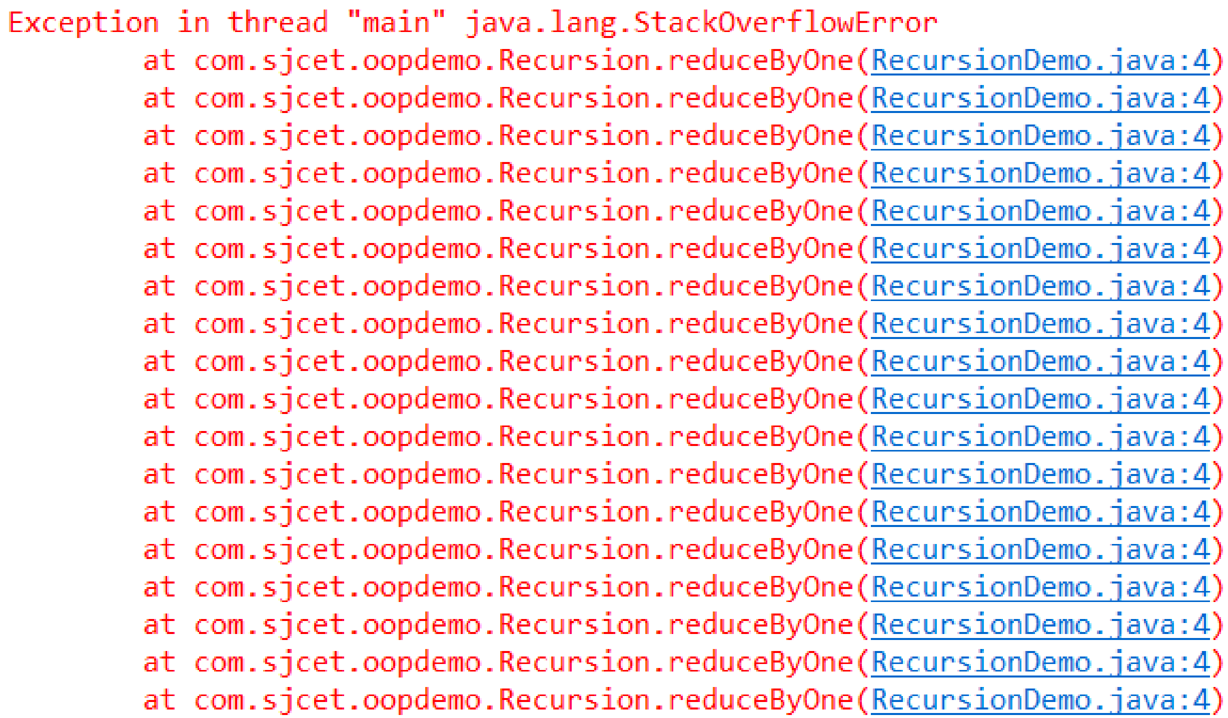
Recursion



```java
class Recursion{
    public void reduceByOne(int n) {
        reduceByOne(n-1);
    }
}
public class RecursionDemo {

    public static void main(String[] args) {
        Recursion test = new Recursion();
        test.reduceByOne(5);
    }

}
```



```
Exception in thread "main" java.lang.StackOverflowError
	at com.sjcet.oopdemo.Recursion.reduceByOne(RecursionDemo.java:4)
	at com.sjcet.oopdemo.Recursion.reduceByOne(RecursionDemo.java:4)
	at com.sjcet.oopdemo.Recursion.reduceByOne(RecursionDemo.java:4)
	at com.sjcet.oopdemo.Recursion.reduceByOne(RecursionDemo.java:4)
	at com.sjcet.oopdemo.Recursion.reduceByOne(RecursionDemo.java:4)
	at com.sjcet.oopdemo.Recursion.reduceByOne(RecursionDemo.java:4)
	at com.sjcet.oopdemo.Recursion.reduceByOne(RecursionDemo.java:4)
	at com.sjcet.oopdemo.Recursion.reduceByOne(RecursionDemo.java:4)
	at com.sjcet.oopdemo.Recursion.reduceByOne(RecursionDemo.java:4)
	at com.sjcet.oopdemo.Recursion.reduceByOne(RecursionDemo.java:4)
	at com.sjcet.oopdemo.Recursion.reduceByOne(RecursionDemo.java:4)
	at com.sjcet.oopdemo.Recursion.reduceByOne(RecursionDemo.java:4)
	at com.sjcet.oopdemo.Recursion.reduceByOne(RecursionDemo.java:4)
	at com.sjcet.oopdemo.Recursion.reduceByOne(RecursionDemo.java:4)
	at com.sjcet.oopdemo.Recursion.reduceByOne(RecursionDemo.java:4)
	at com.sjcet.oopdemo.Recursion.reduceByOne(RecursionDemo.java:4)
	at com.sjcet.oopdemo.Recursion.reduceByOne(RecursionDemo.java:4)
	at com.sjcet.oopdemo.Recursion.reduceByOne(RecursionDemo.java:4)
```

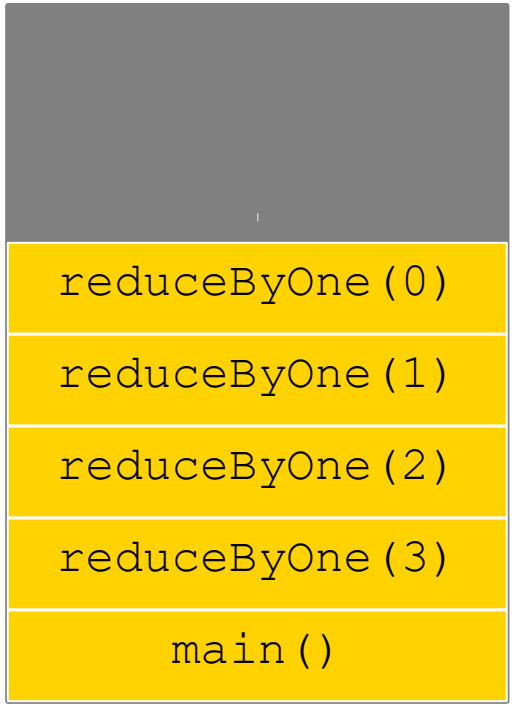
Recursion

```
main{
      reduceByOne(3);
}
reducebyOne(int n){
      if(n>=0){
            reduceByOne(n-1);
      }

}
```

Base Condition added

Stack



Recursion

```java
class Recursion{
    public void reduceByOne(int n) {
        if(n>=0) {
            reduceByOne(n-1);
        }
        System.out.println("Completed Call: " + n);
    }
}
public class RecursionDemo {

    public static void main(String[] args) {
        Recursion test = new Recursion();
        test.reduceByOne(10);
    }

}
```

Output

```
Completed Call: -1
Completed Call: 0
Completed Call: 1
Completed Call: 2
Completed Call: 3
Completed Call: 4
Completed Call: 5
Completed Call: 6
Completed Call: 7
Completed Call: 8
Completed Call: 9
Completed Call: 10
```

Static Members

Java static keyword

- The **static** keyword in java is used for **memory management** mainly.
- We can apply java static keyword with variables, methods, blocks and nested class.

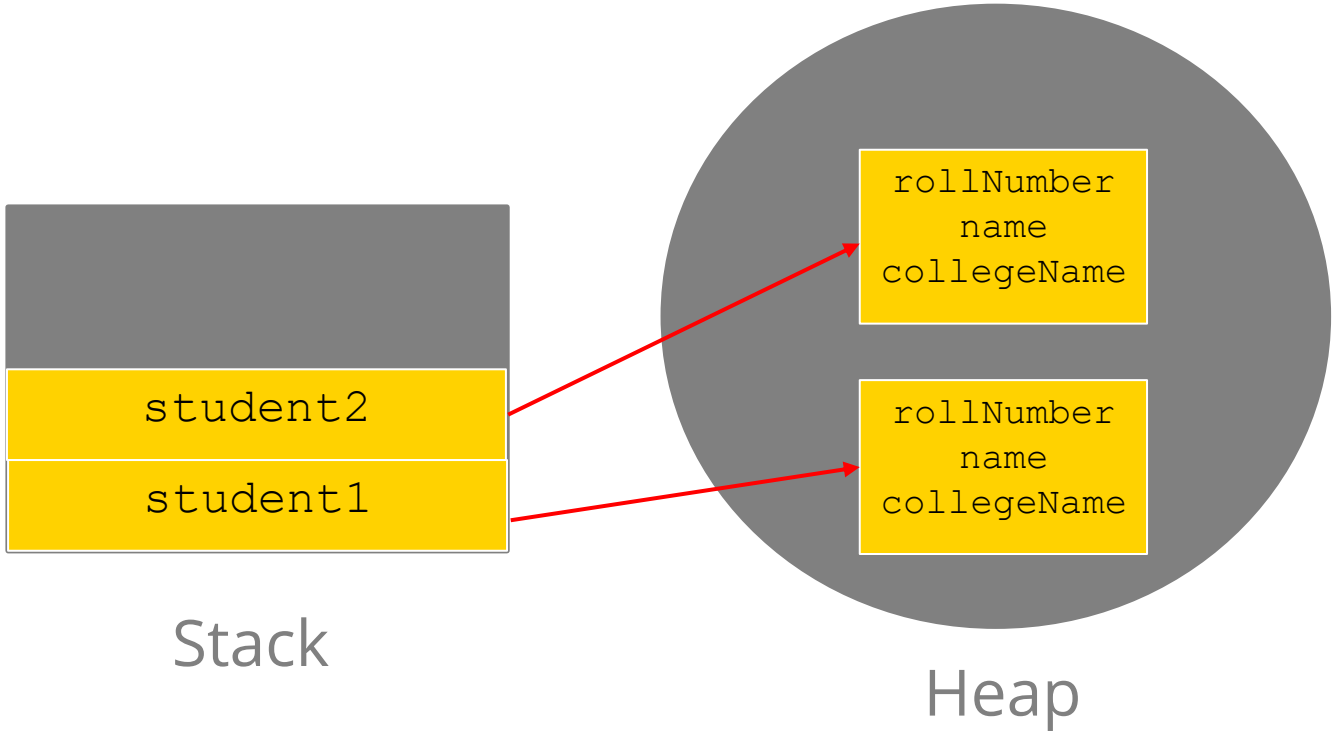
Non static variables

```
class Student{
int rollNumber;
String name;
String collegeName;
}

Student student1 = new Student();
Student student2 = new Student();

//To access members
student1.collegeName;
student2.name;
```

College name is same for all the students, but
It is duplicated in heap area(memory wastage)



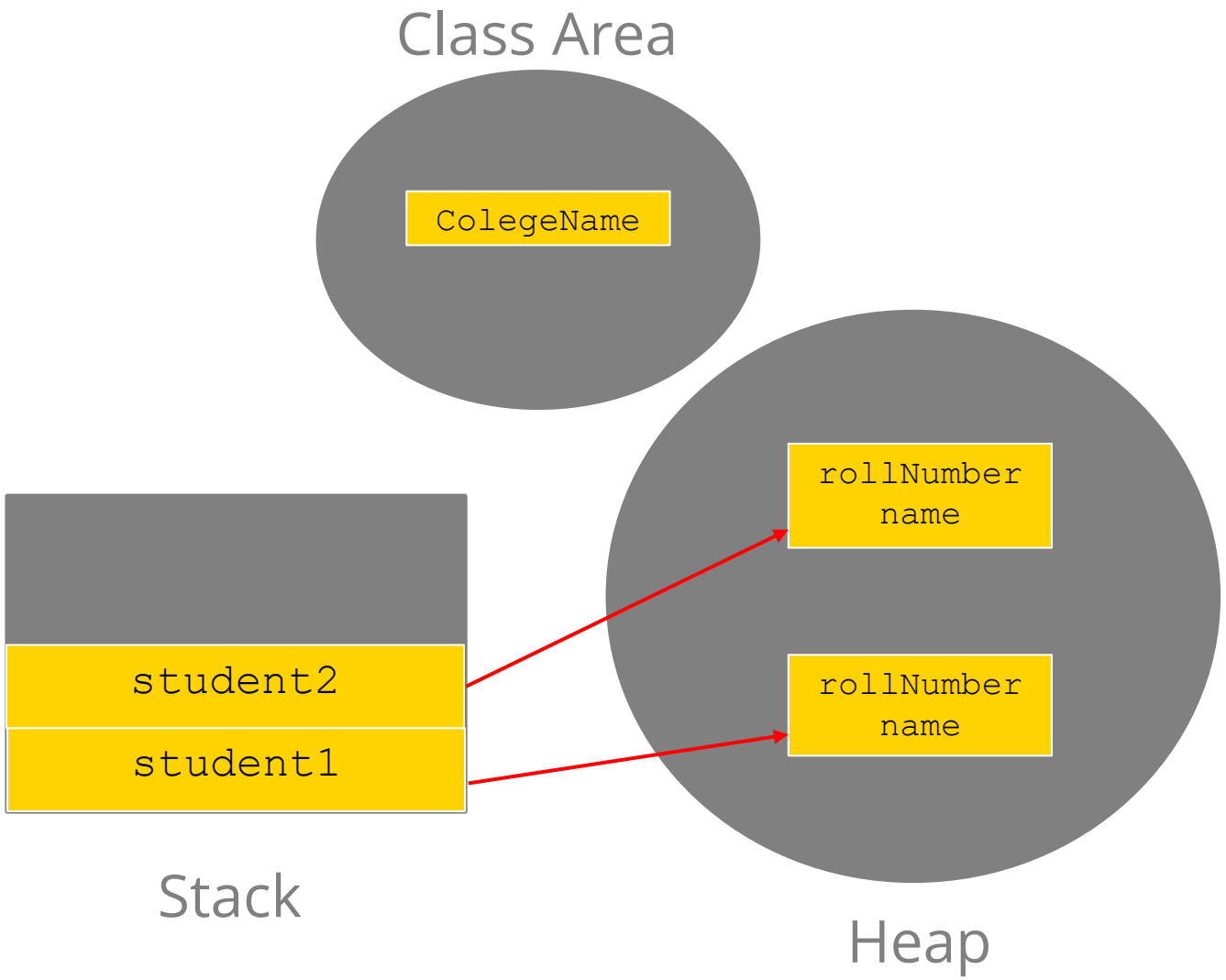
static variables

```
class Student{
int rollNumber;
String name;
static String collegeName;
}

Student student1 = new Student();
Student student2 = new Student();

//To access non static members
student2.name;
student1.rollNumber;

//To access static member
//use class name
Student.collegeName;
```



Java static Variable

- Static variables are not associated with objects
- It belongs to class
- Syntax
  - ClassName.variableName
  - Eg: Student.collegeName;
- Advantage :
  - Makes you program memory efficient

Programming Question: Counting Objects Using Static Keyword

Problem Statement:
You are asked to create a Java class Student that keeps track of the number of student objects created using a **static variable**.
Each time a new Student object is created, the counter should increase automatically. Also, create a method to display the number of students created so far.
Requirements:
1.Create a class Student with:
- A non-static variable name to store the student's name.
- A static variable studentCount to store the total number of students created.
- A constructor that accepts the student name and increases the count.
- A static method getStudentCount() that returns the value of studentCount.

2.In the Main class:
- Create 3 Student objects with names.
- Print the number of students using the static method.

Java static methods

```
class Rectangle{
  //Non Static Method
  public int getArea(int length, int breadth) {
  return length*breadth;
 }
//More methods and variables
}
```

```
Rectangle newRectangle = new Rectangle();
int area  = newRectangle.getArea(10, 20);
```

```
class Rectangle{
  //Static Method
  public static int getArea(int length, int breadth) {
  return length*breadth;
 }
//More methods and variables
}
```

```
int area = Rectangle.getArea(10, 20);
```

- Static method don't require any object
- Static methods are part of class not object

Java static methods

```
class StudentDetails{
    int rollNumber=10;
    String name ="Hari";
    static String collegeName="SJCET";
    public static void getStudentDetails() {
        System.out.println(collegeName);
        System.out.println(rollNumber);
    }
}

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

Cannot make a static reference to the non-static field rollNumber

1 quick fix available:

Change 'rollNumber' to 'static'

Press 'F2' for focus

- They can only directly access static data.

Exercise

1. Create a class called `Student`.
2. Include the following:
   - Instance variables: `name`, `rollNumber`.
   - A **static variable** `studentCount` that tracks the total number of `Student` objects created.
3. Constructor:
   - Accepts `name` and `rollNumber`.
   - Increments `studentCount` each time a new `Student` is created.
4. Static Method:
   - `displayStudentCount()` – displays the total number of students.
5. In the `main()` method:
   - Create at least **3 Student objects**.
   - Call the static method to show the total number of students.

Access static Variables and Methods within the Class

- We are accessing the static variable from another class.
- Hence, we have used the class name to access it.
- However, if we want to access the static member from inside the class, it can be accessed directly.

Access static Variables and Methods within the Class

Non static methods and variables

```java
public class Main {

    // Non static variable
    int age;
    // Non static method
    void display() {
        System.out.println("Static Method");
    }
    public static void main(String[] args) {
        Main objRef = new Main();
        // access the Non static variable
        objRef.age = 30;
        System.out.println("Age is " + objRef.age);

        // access the Non static method
        objRef.display();
    }
}
```

static methods and variables

```java
public class Main {

    // static variable
    static int age;
    // static method
    static void display() {
        System.out.println("Static Method");
    }
    public static void main(String[] args) {

        // access the static variable
        age = 30;
        System.out.println("Age is " + age);

        // access the static method
        display();
    }
}
```

Java static block

- Used to initialize the static data member.

```
static {
    // variable initialization
}
```

```
class Test {
    // static variable
    static int age;

    // static block
    static {
        age = 23;
    }
}
```

Java static block Demo

```java
public class StaticBlockDemo {
    // static variables
        static int a = 23;
        static int b;
        static int max;
        // static blocks
        static {
            System.out.println("First Static block.");
            b = a * 4;
        }
        static {
            System.out.println("Second Static block.");
            max = 30;
        }
        // static method
        static void display() {

            System.out.println("a = " + a);
            System.out.println("b = " + b);
            System.out.println("max = " + max);
        }
        public static void main(String args[]) {
            // calling the static method
            display();
        }
}
```

- It is executed before main method at the time of loading the class by class loader.

Output

```
First Static block.
Second Static block.
a = 23
b = 92
max = 30
```

Final Keyword In Java

Final Keyword In Java

- The final keyword in java is used to **restrict the user.** The java final keyword can be used in many context. Final can be:
  - Variable
  - Method
  - Class
- Java final method - If you make any method as final, you cannot override it. (Will discuss about this at the end of second module)
- Java final class - If you make any class as final, you cannot extend it.

  (Will discuss about this at the topic inheritance)

Final Keyword In Java

- Java final variable - If you make any variable as final, you cannot change the value of final variable(It will be constant).

```
class Bike{
    final int topSpeed = 120;
    void changeTopSpeed() {
        topSpeed = 150;
    }
}
public class FinalVariableDemo {

}
```

The final field Bike.topSpeed cannot be assigned

Press 'F2' for focus

Exercise

1. Create a class `Book` with:
   - Instance variables: `String title`, `String author`, `final int bookID`.
   - A **static variable** `int bookCounter` initialized to 1000.
   - A **static final** variable `LIBRARY_NAME = "Central Library"`.
2. Constructors:
   - A **default constructor** that sets default values for title and author.
   - A **parameterized constructor** that takes title and author as parameters.
   - Each constructor should assign a unique `bookID` using the static `bookCounter`.
3. Method Overloading:
   - `displayInfo()` – displays book title, author, ID.
   - `displayInfo(boolean showLibrary)` – if `showLibrary` is true, also show `LIBRARY_NAME`.
4. Static Method:
   - `displayTotalBooks()` – prints total number of books added.
5. In `main()`:
   - Create 3 `Book` objects using both constructors.
   - Demonstrate both overloaded `displayInfo()` methods.
   - Display total books using the static method.

Java Nested and Inner Class

- In Java, you can define a class within another class. Such class is known as **nested class**.
- There are two types of nested classes you can create in Java.
  - Non-static nested class (inner class)
  - Static nested class

```
class OuterClass {
    // ...
    class NestedClass {
        // ...
    }
}
```

Non-Static Nested Class (Inner Class)

- A non-static nested class is a class within another class and It is commonly known as inner class.
- It has access to members of the enclosing class (outer class).
- Since the inner class exists within the outer class, you must instantiate the outer class first, in order to instantiate the inner class.
- Members of the inner class are known only within the scope of the inner class and may not be used by the outer class.

Non-Static Nested Class (Inner Class)

```java
public class InnerClassDemo {
    public static void main(String[] args) {

        // create object of Outer class CPU
        CPU cpu = new CPU();
        cpu.price = 15000;
        System.out.println("CPU Price = "+cpu.price);

        // create an object of inner class Processor using outer class
        CPU.Processor processor = cpu.new Processor();

        // create an object of inner class RAM using outer class CPU
        CPU.RAM ram = cpu.new RAM();
        System.out.println("Processor Cache = " + processor.getCache());
        System.out.println("Ram Clock speed = " + ram.getClockSpeed());
    }

}
```

```java
class CPU {
    double price;
    // nested class
    class Processor{
        // members of nested class
        double cores;
        String manufacturer;

        double getCache(){
            return 4.3;
        }
    }
    // nested protected class
    class RAM{
        // members of protected nested class
        double memory;
        String manufacturer;

        double getClockSpeed(){
            return 5.5;
        }
    }
}
```

Static Nested Class

- In Java, we can also define a static class inside another class.
- Such class is known as **static nested class**.
- Static nested classes are **not called static inner classes**.
- Unlike inner class, a static nested class cannot access the member variables of the outer class.
  - It is because the static nested class doesn't require you to create an instance of the outer class.

Static Nested Class

```java
package com.sjcet.oopdemo;
class MotherBoard {

        // static nested class
        static class USB{
            int usb2 = 2;
            int usb3 = 1;
            int getTotalPorts(){
                return usb2 + usb3;
            }
        }

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

            // create an object of the static nested class
            // using the name of the outer class
            MotherBoard.USB usb = new MotherBoard.USB();
            System.out.println("Total Ports = " + usb.getTotalPorts());
        }
}
```

static nested class doesn't require you to create an instance of the outer class

Instance of the outer class MotherBoard is not created

Thank You

Prof. Sarju S
Department of Computer Science and Engineering
St. Joseph’s College of Engineering and Technology, Palai
sarju.s@sjcetpalai.ac.in