

Course code : **CSE2005**

Course title : **Object Oriented Programming**

# **Introduction to Object Oriented Concepts**

# Objectives

This session will give the knowledge about

- Object Oriented Concepts
- Basics of Class and Objects
- Constructors
- this keyword
- static keyword

# Introduction to OOP

Object-oriented programming is a method of implementation in which programs are

- organized as cooperative **collections of objects**,
- each of which represents an **instance of some class**, and
- whose classes are **all members of a hierarchy of classes** united via inheritance relationships.

- Grady Booch

# Introduction to OOP

## Structured Approach

- Based on functions
- goto branching
- C, C++, COBOL, Pascal
- Some disadvantages: No constructs for encapsulation, chances of code repetition, No strong data hiding concept, difficult to debug

## Object-oriented Approach

- Smalltalk, Java, C#,

# OOP Concepts

- Object
- Class
- Data Abstraction and Encapsulation
- Polymorphism
- Inheritance
- Data Binding and
- Message Passing

# Object





- A **thing in a real world** that can be either physical or conceptual. An object in object oriented programming can be physical or conceptual.
- Conceptual objects are entities that are not tangible in the way real world physical objects are.
- Bulb is a physical object. While college is a conceptual object.
- Conceptual objects may not have a real world equivalent. For instance, a Stack object in a program.

# Object

- Object has state and behavior.
- The object's state is determined by the value of its properties or attributes.
- Properties or attributes -> member **variables** or data members
- The object's behavior is determined by the operations that it provides.
- Operations -> member functions or **methods**

# Putting it together

A bulb:

1. It's a real-world thing.  **Object**
2. Can be switched on and switched off.  **Methods**
3. It has real features like the glass covering, filament and holder.  **Member variables**
4. It also has conceptual features like power.
5. A bulb manufacturing factory produces many bulbs based on a basic description / pattern of what a bulb is.  **Class**

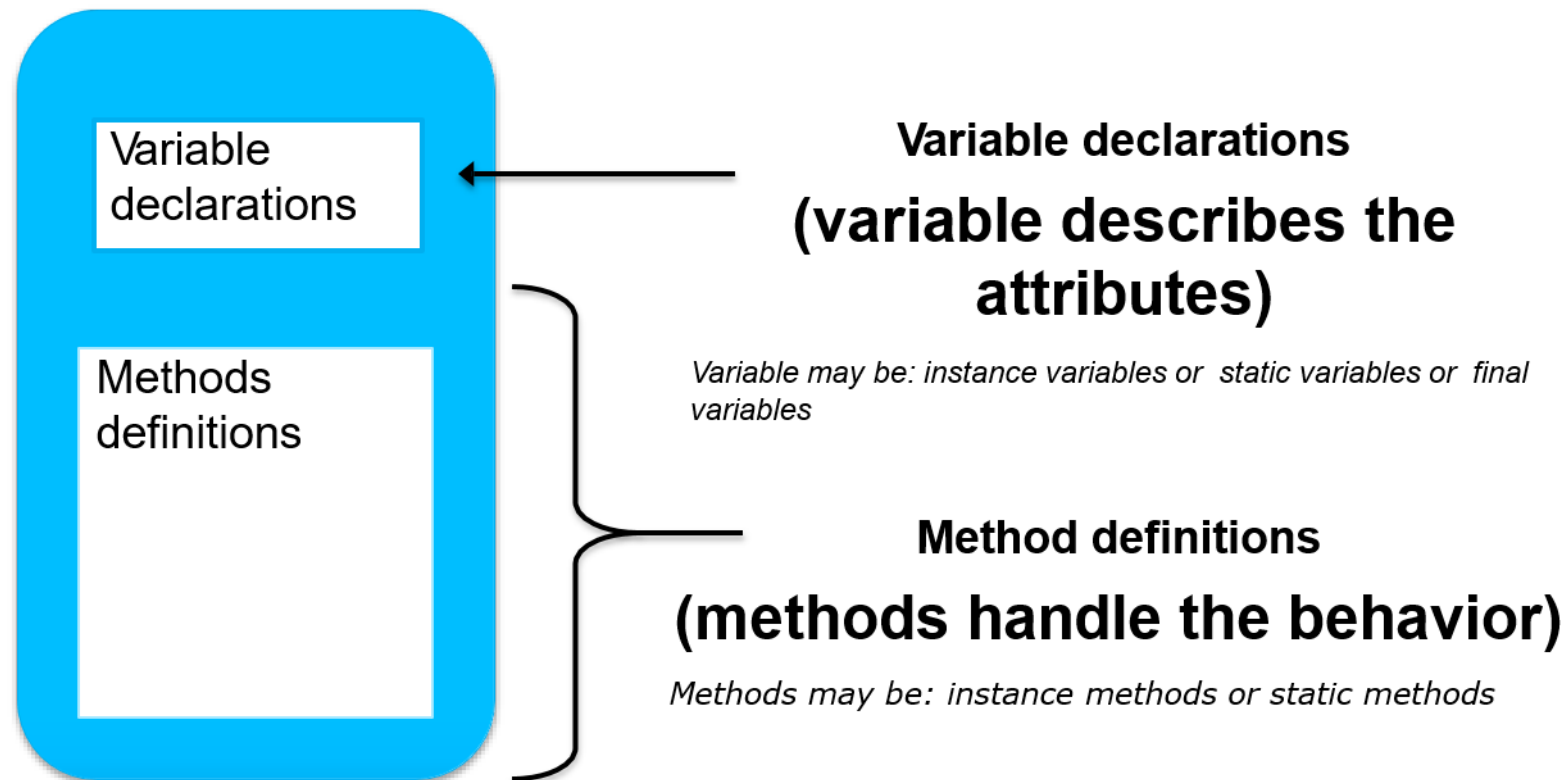


# Class

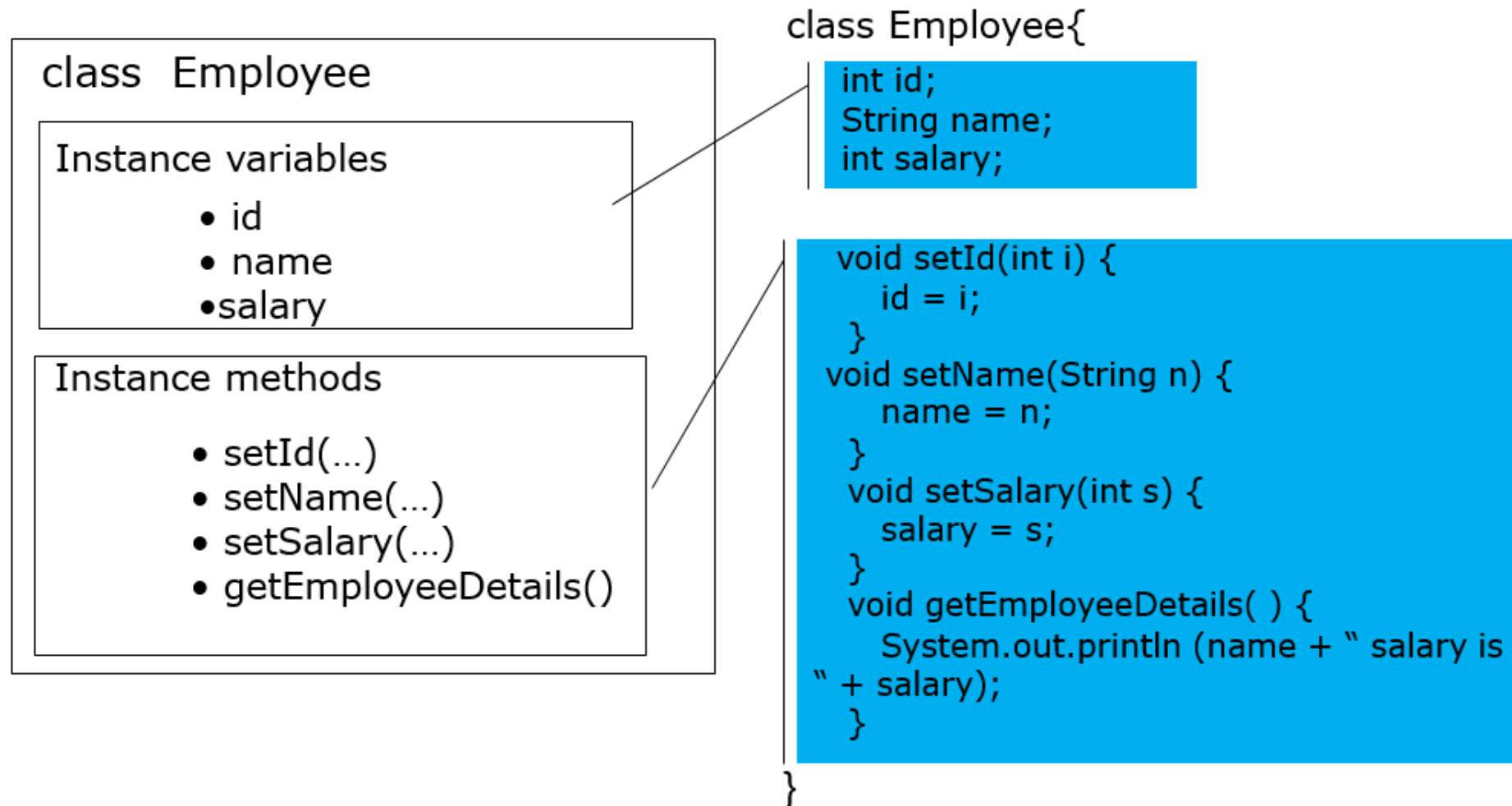
- A class is a construct created in object-oriented programming languages that enables creation of objects.
- Also sometimes called blueprint or template or prototype from which objects are created.
- It defines members (variables and methods).
- A class is an **abstraction**.

# Class

A class contains variable declarations and method definitions



# Defining a Class in java



# Basic information about a class

```
public class Account {  
    double balance;  
    public void deposit( double amount ){  
        balance += amount;  
    }  
    public double withdraw( double amount ){  
        int minimum_balance=5000;  
        if (balance >= (amount+minimum_balance)){  
            balance -= amount;  
            return amount;  
        } }  
    public double getbalance(){  
        return balance;  
    } }  
}
```

Instance  
Variable

Parameter  
or  
argument

local  
Variable

## Member variables

A class contains members which can either be variables(fields) or methods(behaviors).

A variable declared within a class(outside any method) is known as an **instance variable**.

A variable declared within a method is known as **local variable**.

Variables with method declarations are known as **parameters or arguments**.

A class variable can also be declared as static whereas a local variable cannot be static.

# Objects and References

Once a class is defined, you can declare a variable (object reference) of type class

**Student stud1;**

**Employee emp1;**

The **new** operator is used to create an object of that reference type

**Employee emp = new Employee();**

Object reference

object

Object references are used to store objects.

Reference can be created for any type of classes (like concrete classes, abstract classes) and interfaces.

## new operator

The **new** operator,

- Dynamically allocates memory for an object
- Creates the object on the heap
- Returns a reference to it
- The reference is then stored in the variable

# Example of a class

```
package vit.demo;
```

```
class Employee {  
    int id;  
    String name;  
    int salary;  
  
    public void setId(int id) {  
        this.id = id;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
}
```

```
    public void setSalary(int salary) {  
        this.salary = salary;  
    }  
    void getEmployeeDetails() {  
        System.out.println(name + "  
        salary is " + salary);  
    }  
}
```



## Example of a class

```
public class Main {  
    public static void main(String[] args) {  
        Employee emp1 = new Employee();  
        emp1.setId(101);  
        emp1.setName("John");  
        emp1.setSalary(12000);  
        emp1.getEmployeeDetails();  
    }  
}
```

# Constructors

While designing a class, the class designer can define within the class, a special method called 'constructor'

Constructor is automatically invoked whenever an object of the class is created

Rules to define a constructor

- A constructor **has the same name as the class name**
- A constructor **should not have a return type**
- A constructor **can be defined with any access specifier** (private, public)
- A class **can contain more than one constructor, So it can be overloaded**

# Constructors

```
package vit.demo;
```

```
class Sample {  
    private int id;  
    Sample(){  
        id=101;  
        System.out.println(id);  
    }  
    Sample(int no){  
        id=no;  
        System.out.println(id);  
    }  
}
```

# Constructors

```
public class Main {  
    public static void main(String[] args) {  
        Sample obj1=new Sample();  
        Sample obj2=new Sample(102);  
    }  
}
```

Output:

## this keyword

- Each **class member function** contains an **implicit reference** of its class type, named **this**
- this reference is created **automatically by the compiler**
- It **contains the address of the object** through which the function is invoked
- Use of this keyword
  - **this can be used to refer instance variables when there is a clash with local variables or method arguments**
  - this can be used to call overloaded constructors from another constructor of the same class

## this keyword

- Use **this.variableName** to explicitly refer to the instance variable.
- Use variable Name to refer to the parameter.
- The **this** reference is implicitly used to refer to instance
- variables and methods.
- It **CANNOT** be used in a static method.

# this keyword

```
package vit.demo;
class Sample {
    int id=100;
    Sample(){
        int id=102;
        this.id=101;
        System.out.println(id);
    }
}
public class Main {
    public static void main(String[] args) {
        Sample obj1=new Sample();
        System.out.println(obj1.id);
    }
}
```

```
package vit.demo;
class Sample {
    int id=100;
    Sample(){
        this(101);
        System.out.println(id);
    }
    public Sample(int i) {
        id=i;
        System.out.println(id);
    }
}
```

# Static Class Members

Static class members are **the members of a class that do not belong to an instance of a class**

We can access static members directly by prefixing the members with the class name

`ClassName.staticVariable`

`ClassName.staticMethod(...)`



# Static Class Members

## **Static variables:**

- Shared among all objects of the class
- Only one copy exists for the entire class to use
- Stored within the class code, separately from instance variables that describe an individual object
- Public static final variables are global constants

# Static Class Members

## Static methods:

- Static methods can only access directly the static members and manipulate a class's static variables
- Static methods cannot access non-static members (instance variables or instance methods) of the class
- Static method cant access this and super references

# static keyword

```
package vit.demo;
class StaticDemo {
    static int a;
    int b;

    public void set(int i, int j) {
        a = i;
        b = j;
    }

    public void show() {
        System.out.println("static value " + a);
        System.out.println("non-static value " + b);
    }
}
```

```
class Main {
    public static void main(String arg[]) {
        StaticDemo x = new StaticDemo();
        StaticDemo y = new StaticDemo();
        x.set(1, 1);
        x.show();
        y.set(2, 2);
        y.show();
        x.show();
    }
}
```

# static method

```
package vit.demo;
```

```
class StaticDemo {  
    public static void show1(){  
        System.out.println("i am static");  
    }  
    public void show2(){  
        System.out.println("i am non-static");  
    }  
}
```

```
class Main {  
    public static void main(String arg[]) {  
        StaticDemo x = new StaticDemo();  
        x.show1();  
        x.show2();  
        show1();  
        show2();  
    }  
}
```

## Time to Think

Why is main() method static ?

If a java application has to be executed, there has to be a starting point. main() method is supposed to be that starting point. But Java doesn't allow you to execute any method unless you create an object of the class where the method resides. Unless we execute the code, how do we create an instance?

# Quiz

What will be the result, if we try to compile and execute the following code as **java Sample**

```
class Sample {  
    int i_val=10;  
    public static void main(String arg[]) {  
        System.out.println("i_val is "+this.i_val);  
    }  
}
```

# Quiz

What will be the result, if we try to compile and execute the following code as **java Sample**

```
class Sample {  
    int i_val=10;  
    Sample(int x){  
        i_val=x;  
        System.out.println("i_val is "+i_val);  
    }  
    public static void main(String arg[]) {  
        Sample obj=new Sample(20);  
    } }
```

# Static Block

A static block is a block of code enclosed in braces, preceded by the keyword `static`

```
static{  
    System.out.println("i am static block");  
}
```

The statements within the static block are executed automatically when the class is loaded into JVM



## Static Block

- A class can have any number of static blocks and they can appear anywhere in the class
- They are executed in the order of their appearance in the class
- JVM combines all the static blocks in a class as single static block and executes them
- You can invoke static methods from the static block and they will be executed as and when the static block gets executed

# Guess the output?

```
class Main {  
    static{  
        System.out.println("i am static 1");  
    }  
    Main(){  
        System.out.println("i am constructor");  
    }  
    static void method(){  
        System.out.println("i am static method");  
    }  
    static{  
        System.out.println("i am static 3");  
    }  
}
```

```
public static void main(String arg[]) {  
    Main obj = new Main();  
    method();  
}  
  
static{  
    System.out.println("i am static 2");  
}  
}
```

# Guess the output?

```
class Main {  
    static int a = 3;  
    static int b;  
  
    static void show(int x) {  
        System.out.println("x = " + x);  
        System.out.println("a = " + a);  
        System.out.println("b = " + b);  
    }  
    static {  
        System.out.println("Static block");  
        b = a * 4;  
    }  
}
```

```
public static void main(String args[]) {  
    show(42);  
}
```

# Guess the output?

```
package vit.demo;
```

```
class Sample {  
    static int a = 42;  
    static int b = 99;  
    static void callme() {  
        System.out.println("a = " + a);  
    }  
}
```

```
class Main {  
    public static void main(String args[]) {  
        Sample.callme();  
        System.out.println("b = " + Sample.b);  
    }  
}
```

# Summary

We have discussed about

- Object Oriented Concepts
- Basics of Class and Objects
- Constructors
- this keyword
- static keyword