## National University of Computer & Emerging Sciences, Karachi

**Computer Science Department Spring 2024, Lab Manual – 05**

|  |  |
| --- | --- |
| **Course Code: CL-217** | **Course: Object Oriented Programming Lab** |
| **Instructor(s) :** | **Ali Fatmi** |

LAB – 5

Final Keyword

Static Keyword

Arrays of Objects

**Final Keyword In Java**

The final keyword in java is used to restrict the user. The java final keyword can be used in many contexts. Final can be:

1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.

**Java final variable**

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

### **Example of final variable**

There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

**class** Bike9{

**final** **int** speedlimit=90;//final variable

**void** run(){

  speedlimit=400;

 }

**public** **static** **void** main(String args[]){

 Bike9 obj=**new**  Bike9();

 obj.run();

 }

}//end of class

.

Output:Compile Time Error

**Java final method**

If you make any method as final, you cannot override it.

**Example of final method**

**class** Bike{

**final** **void** run(){System.out.println("running");}

}

**class** Honda **extends** Bike{

**void** run(){System.out.println("running safely with 100kmph");}

**public** **static** **void** main(String args[]){

   Honda honda= **new** Honda();

   honda.run();

   }

}

Output:Compile Time Error

**Java final class**

If you make any class as final, you cannot extend it.

**Example of final class**

**final** **class** Bike{}

**class** Honda1 **extends** Bike{

**void** run(){System.out.println("running safely with 100kmph");}

**public** **static** **void** main(String args[]){

  Honda1 honda= **new** Honda1();

  honda.run();

  }

}

Output:Compile Time Error

### **Is final method inherited?**

Ans) Yes, final method is inherited but you cannot override it. For Example:

**class** Bike{

**final** **void** run(){System.out.println("running...");}

}

**class** Honda2 **extends** Bike{

**public** **static** **void** main(String args[]){

**new** Honda2().run();

   }

}

Output:running...

### **What is blank or uninitialized final variable?**

A final variable that is not initialized at the time of declaration is known as blank final variable.

If you want to create a variable that is initialized at the time of creating object and once initialized may not be changed, it is useful. For example PAN CARD number of an employee.

It can be initialized only in constructor.

### Example of blank final variable

**class** Student{

**int** id;

String name;

**final** String PAN\_CARD\_NUMBER;

...

}

**Can we initialize blank final variable?**

Yes, but only in constructor. For example:

**class** Bike10{

**final** **int** speedlimit;//blank final variable

  Bike10(){

  speedlimit=70;

  System.out.println(speedlimit);

  }

**public** **static** **void** main(String args[]){

**new** Bike10();

 }

}

Output: 70

### **static blank final variable**

A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block.

### **Example of static blank final variable**

**class** A{

**static** **final** **int** data;//static blank final variable

**static**{ data=50;}

**public** **static** **void** main(String args[]){

    System.out.println(A.data);

 }

}

### **What is final parameter?**

If you declare any parameter as final, you cannot change the value of it.

**class** Bike11{

**int** cube(**final** **int** n){

   n=n+2;//can't be changed as n is final

   n\*n\*n;

  }

**public** **static** **void** main(String args[]){

    Bike11 b=**new** Bike11();

    b.cube(5);

 }

}

Output: Compile Time Error

### **Can we declare a constructor final?**

No, because constructor is never inherited.

# static Keyword in Java

The **static keyword** in Java is mainly used for memory management. The static keyword in Java is used to share the same variable or method of a given class. The users can apply static keywords with variables, methods, blocks, and nested classes. The static keyword belongs to the class than an instance of the class. The static keyword is used for a constant variable or a method that is the same for every instance of a class.

**The *static* keyword is a non-access modifier in Java that is applicable for the following:**

1. Blocks
2. Variables
3. Methods
4. Classes

***Note:****To create a static member(block, variable, method, nested class), precede its declaration with the keyword static.*

### **Characteristics of static keyword:**

Here are some characteristics of the static keyword in Java:

* Shared memory allocation: Static variables and methods are allocated memory space only once during the execution of the program. This memory space is shared among all instances of the class, which makes static members useful for maintaining global state or shared functionality.
* Accessible without object instantiation: Static members can be accessed without the need to create an instance of the class. This makes them useful for providing utility functions and constants that can be used across the entire program.
* Associated with class, not objects: Static members are associated with the class, not with individual objects. This means that changes to a static member are reflected in all instances of the class, and that you can access static members using the class name rather than an object reference.
* Cannot access non-static members: Static methods and variables cannot access non-static members of a class, as they are not associated with any particular instance of the class.
* Can be overloaded, but not overridden: Static methods can be overloaded, which means that you can define multiple methods with the same name but different parameters. However, they cannot be overridden, as they are associated with the class rather than with a particular instance of the class.

When a member is declared static, it can be accessed before any objects of its class are created, and without reference to any object. For example, in the below java program, we are accessing static method *m1()* without creating any object of the *Test* class.

// Java program to demonstrate that a static member

// can be accessed before instantiating a class

**class** Test

{

    // static method

**static** **void** m1()

    {

        System.out.println("from m1");

    }

**public** **static** **void** main(String[] args)

    {

          // calling m1 without creating

          // any object of class Test

           m1();

    }

}

**Output**

from m1

### **Static blocks**

If you need to do the computation in order to initialize your **static variables**, you can declare a static block that gets executed exactly once, when the class is first loaded.

Consider the following java program demonstrating the use of static blocks.

|  |
| --- |
| // Java program to demonstrate use of static blocks    **class** Test  {      // static variable  **static** **int** a = 10;  **static** **int** b;        // static block  **static** {          System.out.println("Static block initialized.");          b = a \* 4;      }    **public** **static** **void** main(String[] args)      {         System.out.println("from main");         System.out.println("Value of a : "+a);         System.out.println("Value of b : "+b);      }  } |

**Output**

Static block initialized.

from main

Value of a : 10

Value of b : 40

### **Static variables**

When a variable is declared as static, then a single copy of the variable is created and shared among all objects at the class level. Static variables are, essentially, global variables. All instances of the class share the same static variable.

**Important points for static variables:**

* We can create static variables at the class level only.
* static block and static variables are executed in the order they are present in a program.
* Below is the Java program to demonstrate that static block and static variables are executed in the order they are present in a program.

|  |
| --- |
| // Java program to demonstrate execution  // of static blocks and variables    **class** Test  {      // static variable  **static** **int** a = m1();        // static block  **static** {          System.out.println("Inside static block");      }        // static method  **static** **int** m1() {          System.out.println("from m1");  **return** 20;      }        // static method(main !!)  **public** **static** **void** main(String[] args)      {         System.out.println("Value of a : "+a);         System.out.println("from main");      }  } |

**Output**

from m1

Inside static block

Value of a : 20

from main

### **Static methods**

When a method is declared with the static keyword, it is known as the static method. The most common example of a static method is the main( ) method. As discussed above, Any static member can be accessed before any objects of its class are created, and without reference to any object. Methods declared as static have several restrictions:

* They can only directly call other static methods.
* They can only directly access static data.

Below is the java program to demonstrate restrictions on static methods.

|  |
| --- |
| // Java program to demonstrate restriction on static methods    **class** Test  {      // static variable  **static** **int** a = 10;        // instance variable  **int** b = 20;        // static method  **static** **void** m1()      {          a = 20;          System.out.println("from m1");             // Cannot make a static reference to the non-static field b           b = 10; // compilation error             // Cannot make a static reference to the                   // non-static method m2() from the type Test           m2();  // compilation error             //  Cannot use super in a static context           System.out.println(**super**.a); // compiler error      }        // instance method  **void** m2()      {          System.out.println("from m2");      }        **public** **static** **void** main(String[] args)      {          // main method      }  } |

**Output:**

prog.java:18: error: non-static variable b cannot be referenced from a static context

b = 10; // compilation error

^

prog.java:22: error: non-static method m2() cannot be referenced from a static context

m2(); // compilation error

^

prog.java:25: error: non-static variable super cannot be referenced from a static context

System.out.println(super.a); // compiler error

^

prog.java:25: error: cannot find symbol

System.out.println(super.a); // compiler error

^

symbol: variable a

4 errors

### **When to use static variables and methods?**

Use the static variable for the property that is common to all objects. For example, in class Student, all students share the same college name. Use static methods for changing static variables.

Consider the following java program, that illustrates the use of static keywords with variables and methods.

// A java program to demonstrate use of

// static keyword with methods and variables

// Student class

**class** Student {

    String name;

**int** rollNo;

    // static variable

**static** String cllgName;

    // static counter to set unique roll no

**static** **int** counter = 0;

**public** Student(String name)

    {

**this**.name = name;

**this**.rollNo = setRollNo();

    }

    // getting unique rollNo

    // through static variable(counter)

**static** **int** setRollNo()

    {

        counter++;

**return** counter;

    }

    // static method

**static** **void** setCllg(String name) { cllgName = name; }

    // instance method

**void** getStudentInfo()

    {

        System.out.println("name : " + **this**.name);

        System.out.println("rollNo : " + **this**.rollNo);

        // accessing static variable

        System.out.println("cllgName : " + cllgName);

    }

}

// Driver class

**public** **class** StaticDemo {

**public** **static** **void** main(String[] args)

    {

        // calling static method

        // without instantiating Student class

        Student.setCllg("XYZ");

        Student s1 = **new** Student("Alice");

        Student s2 = **new** Student("Bob");

        s1.getStudentInfo();

        s2.getStudentInfo();

    }

}

**Output**

name : Alice

rollNo : 1

cllgName : XYZ

name : Bob

rollNo : 2

cllgName : XYZ

### **Static Classes**

A class can be made **static** only if it is a nested class. We cannot declare a top-level class with a static modifier but can declare [nested classes](https://www.geeksforgeeks.org/nested-classes-java/) as static. Such types of classes are called Nested static classes. Nested static class doesn’t need a reference of Outer class. In this case, a static class cannot access non-static members of the Outer class.

// A java program to demonstrate use

// of static keyword with Classes

**import** java.io.\*;

**public** **class** GFG {

**private** **static** String str = "GeeksforGeeks";

    // Static class

**static** **class** MyNestedClass {

        // non-static method

**public** **void** disp(){

          System.out.println(str);

        }

    }

**public** **static** **void** main(String args[])

    {

        GFG.MyNestedClass obj

            = **new** GFG.MyNestedClass();

        obj.disp();

    }

}

**Output**

GeeksforGeeks

|  |
| --- |
| **public** **class** ExampleClass {  **public** **static** **int** count = 0;  **public** **int** id;    **public** ExampleClass() {          count++;          id = count;      }    **public** **static** **void** printCount() {          System.out.println("Number of instances: " + count);      }    **public** **void** printId() {          System.out.println("Instance ID: " + id);      }    **public** **static** **void** main(String[] args) {          ExampleClass e1 = **new** ExampleClass();          ExampleClass e2 = **new** ExampleClass();          ExampleClass e3 = **new** ExampleClass();            e1.printId();          e2.printId();          e3.printId();            ExampleClass.printCount();      }  } |

**Output**

Instance ID: 1

Instance ID: 2

Instance ID: 3

Number of instances: 3

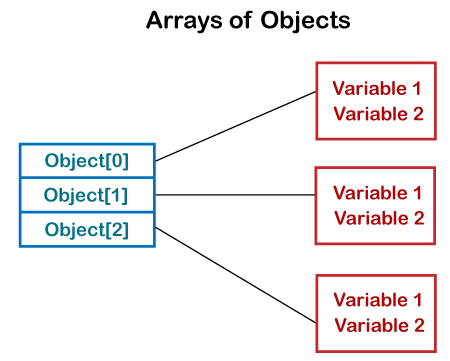
### **Advantages:**

* **Memory efficiency**: Static members are allocated memory only once during the execution of the program, which can result in significant memory savings for large programs.
* **Improved performance**: Because static members are associated with the class rather than with individual instances, they can be accessed more quickly and efficiently than non-static members.
* **Global accessibility:** Static members can be accessed from anywhere in the program, regardless of whether an instance of the class has been created.
* **Encapsulation**of utility methods: Static methods can be used to encapsulate utility functions that don’t require any state information from an object. This can improve code organization and make it easier to reuse utility functions across multiple classes.
* **Constants**: Static final variables can be used to define constants that are shared across the entire program.
* **Class-level functionality**: Static methods can be used to define class-level functionality that doesn’t require any state information from an object, such as factory methods or helper functions.

Overall, the static keyword is a powerful tool that can help to improve the efficiency and organization of your Java programs.

## Array of Objects in Java

Java is an object-oriented programming language. Most of the work done with the help of **objects**. We know that an array is a collection of the same data type that dynamically creates objects and can have elements of primitive types. Java allows us to store objects in an array. In [Java](https://www.javatpoint.com/java-tutorial), the class is also a user-defined data type. An array that conations **class type elements** are known as an **array of objects**. It stores the reference variable of the object.



## Creating an Array of Objects

Before creating an array of objects, we must create an instance of the class by using the new keyword. We can use any of the following statements to create an array of objects.

**Syntax:**

ClassName obj[]=**new** ClassName[array\_length]; //declare and instantiate an array of objects

Or

ClassName[] objArray;

Or

ClassName objeArray[];

Suppose, we have created a class named Employee. We want to keep records of 20 employees of a company having three departments. In this case, we will not create 20 separate variables. Instead of this, we will create an array of objects, as follows.

1. Employee department1[20];
2. Employee department2[20];
3. Employee department3[20];

The above statements create an array of objects with 20 elements.

Let's create an array of objects in a [Java program](https://www.javatpoint.com/java-programs).

In the following program, we have created a class named Product and initialized an array of objects using the constructor. We have created a constructor of the class Product that contains product id and product name. In the main function, we have created individual objects of the class Product. After that, we have passed initial values to each of the objects using the constructor.

**ArrayOfObjects.java**

**public** **class** ArrayOfObjects

{

**public** **static** **void** main(String args[])

{

//create an array of product object

Product[] obj = **new** Product[5] ;

//create & initialize actual product objects using constructor

obj[0] = **new** Product(23907,"Dell Laptop");

obj[1] = **new** Product(91240,"HP 630");

obj[2] = **new** Product(29823,"LG OLED TV");

obj[3] = **new** Product(11908,"MI Note Pro Max 9");

obj[4] = **new** Product(43590,"Kingston USB");

//display the product object data

System.out.println("Product Object 1:");

obj[0].display();

System.out.println("Product Object 2:");

obj[1].display();

System.out.println("Product Object 3:");

obj[2].display();

System.out.println("Product Object 4:");

obj[3].display();

System.out.println("Product Object 5:");

obj[4].display();

}

}

//Product class with product Id and product name as attributes

**class** Product

{

**int** pro\_Id;

String pro\_name;

//Product class constructor

Product(**int** pid, String n)

{

pro\_Id = pid;

pro\_name = n;

}

**public** **void** display()

{

System.out.print("Product Id = "+pro\_Id + "  " + " Product Name = "+pro\_name);

System.out.println();

}

}

**Output:**

Product Object 1:

Product Id = 23907 Product Name = Dell Laptop

Product Object 2:

Product Id = 91240 Product Name = HP 630

Product Object 3:

Product Id = 29823 Product Name = LG OLED TV

Product Object 4:

Product Id = 11908 Product Name = MI Note Pro Max 9

Product Object 5:

Product Id = 43590 Product Name = Kingston USB

## Task 01

## Design a scenario where you have an array of Student objects representing student records in a class. Each Student object contains properties such as studentId, name, and grades (an array of integers representing grades obtained in different subjects). Create a problem where you need to implement a method to calculate the average grade for each student and then find the student with the highest average grade. The objective is to provide insights into student performance and identify the top performer in the class.

## Task 02

## Design a scenario where you have an array of Book objects representing books available in a library. Each Book object contains properties such as title, author, and genre. Create a problem where you need to implement a method to recommend a list of books to a library user based on their preferred genre. The objective is to personalize the user experience by suggesting books that align with their interests.

## Task 03

## Design a scenario where you have a BankAccount class representing individual bank accounts. Each BankAccount object contains properties such as accountNumber, balance, and transactionHistory. Create a problem where you need to implement methods for depositing and withdrawing funds from the account. Ensure that once a transaction is made (deposit or withdrawal), the balance property is finalized using the final keyword to prevent further modification. The objective is to maintain data consistency and prevent accidental changes to the account balance after transactions.

## Task 04

## Design a scenario where you have a Student class representing students in a school. Each Student object contains properties such as studentId, name, and grades (an array of integers representing grades obtained in different subjects). Create a problem where you need to implement methods for adding grades to the student's record and calculating the average grade. Use the final keyword to ensure that once the grades are calculated, they cannot be modified. The objective is to maintain the integrity of student grades and prevent accidental changes after calculation.

## Task 05

## Design a scenario where you have an Item class representing items in an inventory system. Each Item object contains properties such as itemId, name, quantity, and price. Create a problem where you need to implement a static method updateInventory in the InventoryManager class to handle inventory updates. This method should take parameters such as the itemId and the quantity to be added or subtracted. Ensure that the updateInventory method updates the quantity of the specified item in the inventory correctly, and the changes are reflected across all instances of the Item class. The objective is to demonstrate the use of the static keyword to manage shared data across all instances of a class in an inventory management system.

## Task 06

## Design a scenario where you have a Booking class representing individual flight bookings in a flight booking system. Each Booking object contains properties such as bookingNumber, passengerName, flightNumber, seatNumber, and bookingStatus. Create a problem where you need to implement a static method cancelBooking in the BookingManager class to handle booking cancellations for a specific flight. This method should take parameters such as the bookingNumber to be canceled. Ensure that the cancelBooking method cancels the specified booking correctly and updates the bookingStatus across all instances of the Booking class for the corresponding flight. The objective is to demonstrate the use of the static keyword to manage shared data across all instances of a class in a flight booking system.