**Week-6 Lab**

**WAP to Assign different priorities to the 2 threads and observe the behavior.**

**package** week\_6**;**

class **ThreadPriorityDemo** *extends* *Thread* {

*public* *void* run() {

        System**.***out***.**println(getName() **+** " running with priority " **+** getPriority())**;**

**for** (*int* i **=** 0**;** i **<** 5**;** i**++**) {

            System**.***out***.**println(getName() **+** " count: " **+** i)**;**

        }

    }

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

        ThreadPriorityDemo t1 **=** **new** ThreadPriorityDemo()**;**

        ThreadPriorityDemo t2 **=** **new** ThreadPriorityDemo()**;**

        t1**.**setName("Low Priority Thread")**;**

        t2**.**setName("High Priority Thread")**;**

        t1**.**setPriority(Thread**.***MIN\_PRIORITY*)**;** *// 1*

        t2**.**setPriority(Thread**.***MAX\_PRIORITY*)**;** *// 10*

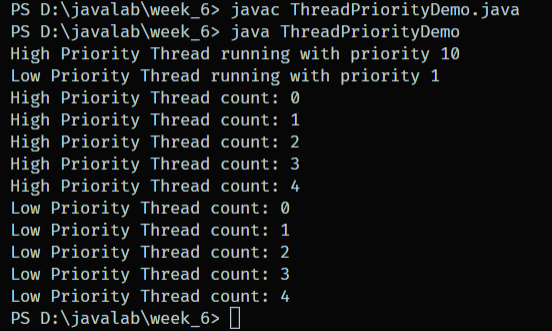
        t1**.**start()**;**

        t2**.**start()**;**

    }

}

**Output -**



**WAP to create a producer-consumer scenario using– the wait () and notify () methods for thread synchronization.**

class **Buffer** {

*private* *int* data**;**

*private* *boolean* available **=** false**;**

*public* *synchronized* *void* produce(*int* **value**) *throws* InterruptedException {

**while** (available) wait()**;**

        data **=** value**;**

        available **=** true**;**

        System**.***out***.**println("Produced: " **+** data)**;**

        notify()**;**

    }

*public* *synchronized* *int* consume() *throws* InterruptedException {

**while** (**!**available) wait()**;**

        available **=** false**;**

        System**.***out***.**println("Consumed: " **+** data)**;**

        notify()**;**

**return** data**;**

    }

}

*public* class **ProducerConsumerDemo** {

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

        Buffer buffer **=** **new** Buffer()**;**

        Thread producer **=** **new** Thread(() **->** {

**for** (*int* i **=** 0**;** i **<** 5**;** i**++**) {

**try** { buffer**.**produce(i)**;** } **catch** (InterruptedException **e**) { e**.**printStackTrace()**;** }

            }

        })**;**

        Thread consumer **=** **new** Thread(() **->** {

**for** (*int* i **=** 0**;** i **<** 5**;** i**++**) {

**try** { buffer**.**consume()**;** } **catch** (InterruptedException **e**) { e**.**printStackTrace()**;** }

            }

        })**;**

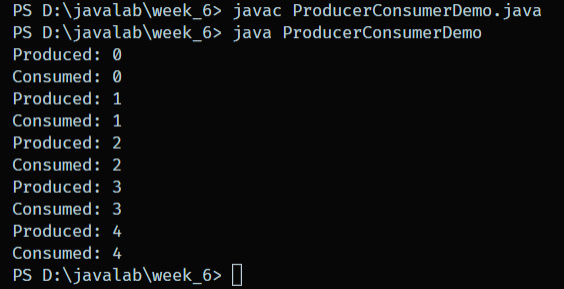
        producer**.**start()**;**

        consumer**.**start()**;**

    }

}

**Output -**

****

**WAP to implement deadlock in Java (Content Beyond Syllabus).**

*public* class **DeadlockDemo** {

*private* *final* Object lock1 **=** **new** Object()**;**

*private* *final* Object lock2 **=** **new** Object()**;**

*public* *void* method1() {

*synchronized* (lock1) {

            System**.***out***.**println("Thread 1: Holding lock1...")**;**

**try** { Thread**.**sleep(100)**;** } **catch** (InterruptedException **e**) {}

*synchronized* (lock2) {

                System**.***out***.**println("Thread 1: Holding lock2...")**;**

            }

        }

    }

*public* *void* method2() {

*synchronized* (lock2) {

            System**.***out***.**println("Thread 2: Holding lock2...")**;**

**try** { Thread**.**sleep(100)**;** } **catch** (InterruptedException **e**) {}

*synchronized* (lock1) {

                System**.***out***.**println("Thread 2: Holding lock1...")**;**

            }

        }

    }

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

        DeadlockDemo deadlock **=** **new** DeadlockDemo()**;**

        Thread t1 **=** **new** Thread(() **->** deadlock**.**method1())**;**

        Thread t2 **=** **new** Thread(() **->** deadlock**.**method2())**;**

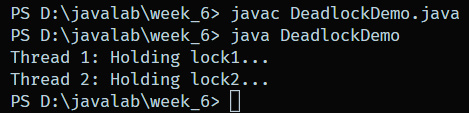
        t1**.**start()**;**

        t2**.**start()**;**

    }

}

**Output –**



**WAP to implement the following new features in Java. (a) Functional Interface (b)Lambda Expression: Write a Java program to implement a lambda expression to check if a given string is empty. (c) Method References: (d)Default and Static Method in Interface (e) Inner Class**

**package** week\_6**;**

@**FunctionalInterface**

interface **Greeting** {

*void* sayHello()**;**

}

interface **CheckEmpty** {

*boolean* isEmpty(String **s**)**;**

}

interface **MyInterface** {

*default* *void* defaultMethod() {

        System**.***out***.**println("Default method called")**;**

    }

*static* *void* staticMethod() {

        System**.***out***.**println("Static method called")**;**

    }

}

class **LambdaEmptyCheck** {

*public* *static* *void* test() {

        CheckEmpty check **=** s **->** s**.**isEmpty()**;**

        System**.***out***.**println(check**.**isEmpty(""))**;**    *// true*

        System**.***out***.**println(check**.**isEmpty("abc"))**;** *// false*

    }

}

class **MethodReferenceDemo** {

*public* *static* *void* printMessage() {

        System**.***out***.**println("Hello from method reference!")**;**

    }

*public* *static* *void* test() {

        Runnable r **=** MethodReferenceDemo**::**printMessage**;**

        r**.**run()**;**

    }

}

class **DefaultStaticDemo** *implements* *MyInterface* {

*public* *static* *void* test() {

        DefaultStaticDemo obj **=** **new** DefaultStaticDemo()**;**

        obj**.**defaultMethod()**;**

        MyInterface**.**staticMethod()**;**

    }

}

class **Outer** {

    class **Inner** {

*void* display() {

            System**.***out***.**println("Inner class method")**;**

        }

    }

*public* *void* test() {

        Outer**.**Inner inner **=** *this***.***new* Inner()**;**

        inner**.**display()**;**

    }

}

*// The main public class - file name must be Week6Demo.java*

*public* class **newFeatures** {

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

        System**.***out***.**println("LambdaEmptyCheck:")**;**

        LambdaEmptyCheck**.**test()**;**

        System**.***out***.**println("\nMethodReferenceDemo:")**;**

        MethodReferenceDemo**.**test()**;**

        System**.***out***.**println("\nDefaultStaticDemo:")**;**

        DefaultStaticDemo**.**test()**;**

        System**.***out***.**println("\nOuter class Inner:")**;**

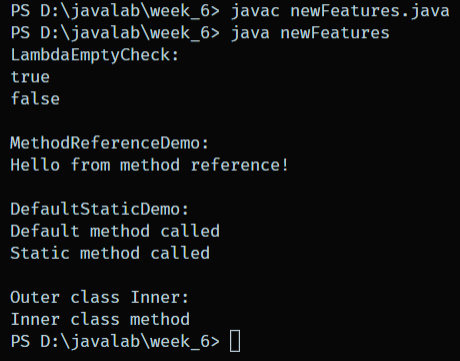
        Outer outer **=** **new** Outer()**;**

        outer**.**test()**;**

    }

}

**Output –**



**WAP to implement different types of Annotations in JAVA.**

**package** week\_6**;**

*import* java**.**lang**.**annotation**.***\****;**

@**Retention**(RetentionPolicy**.***RUNTIME*)

@**Target**(ElementType**.***METHOD*)

@*interface* **MyAnnotation** {

    String info() **default** "Default info"**;**

}

class **AnnotationDemo** {

    @**MyAnnotation**(info **=** "Test method annotation")

*public* *void* testMethod() {

        System**.***out***.**println("Inside testMethod")**;**

    }

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

        AnnotationDemo obj **=** **new** AnnotationDemo()**;**

        obj**.**testMethod()**;**

        MyAnnotation annotation **=** obj**.**getClass()

**.**getMethod("testMethod")

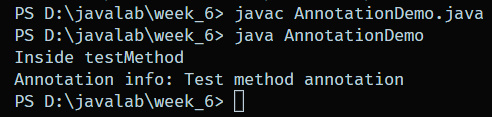
**.**getAnnotation(MyAnnotation**.***class*)**;**

        System**.***out***.**println("Annotation info: " **+** annotation**.**info())**;**

    }

}

**Output –**



**WAP to filter data by using streams.**

*import* java**.**util**.***\****;**

*import* java**.**util**.**stream**.***\****;**

*public* class **StreamFilterDemo** {

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

        List**<**String**>** list **=** Arrays**.**asList("apple"**,** "banana"**,** "avocado"**,** "blueberry")**;**

        List**<**String**>** filtered **=** list**.**stream()

**.**filter(s **->** s**.**startsWith("a"))

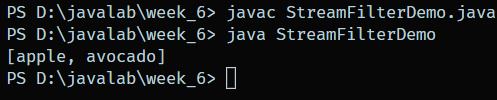
**.**collect(Collectors**.**toList())**;**

        System**.***out***.**println(filtered)**;** *// [apple, avocado]*

    }

}

**Output –**



**WAP to Traversing the array elements and to sum the elements using For-each loop.**

*public* class **sumArray** {

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

*int*[] arr **=** {1**,** 2**,** 3**,** 4**,** 5}**;**

*int* sum **=** 0**;**

**for** (*int* num **:** arr) {

            sum **+=** num**;**

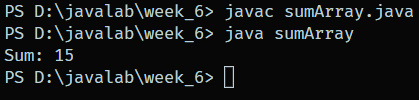
        }

        System**.***out***.**println("Sum: " **+** sum)**;**

    }

}

**Output –**



**WAP to implement Base64 Encoding and Decoding.**

*import* java**.**util**.**Base64**;**

*public* class **Base64Demo** {

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

        String original **=** "Hello World!"**;**

        String encoded **=** Base64**.**getEncoder()**.**encodeToString(original**.**getBytes())**;**

        System**.***out***.**println("Encoded: " **+** encoded)**;**

*byte*[] decodedBytes **=** Base64**.**getDecoder()**.**decode(encoded)**;**

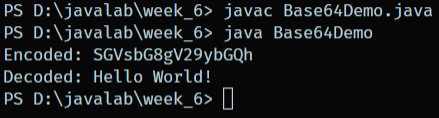
        String decoded **=** **new** String(decodedBytes)**;**

        System**.***out***.**println("Decoded: " **+** decoded)**;**

    }

}

**Output –**



**WAP to implement Local Variable Type Inference.**

*public* class **LocalVarInferenceDemo** {

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

        var message **=** "Hello, Local Variable Type Inference!"**;**

        System**.***out***.**println(message)**;**

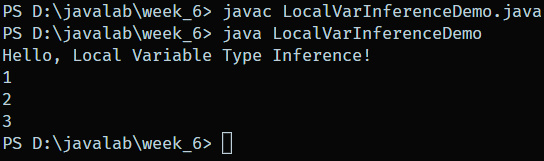
        var list **=** java**.***util***.***List***.**of(1**,** 2**,** 3)**;**

        list**.**forEach(System**.***out***::**println)**;**

    }

}

**Output –**



**WAP to implement Sealed Class.**

*abstract* *sealed* class **Vehicle** *permits* *Car***,** *Truck* {

*abstract* *void* drive()**;**

}

*final* class **Car** *extends* *Vehicle* {

*void* drive() {

        System**.***out***.**println("Driving a car.")**;**

    }

}

*final* class **Truck** *extends* *Vehicle* {

*void* drive() {

        System**.***out***.**println("Driving a truck.")**;**

    }

}

*public* class **SealedDemo** {

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

        Vehicle v1 **=** **new** Car()**;**

        Vehicle v2 **=** **new** Truck()**;**

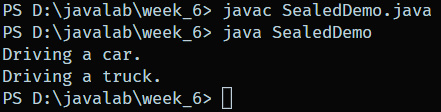
        v1**.**drive()**;**

        v2**.**drive()**;**

    }

}

**Output –**



**WAP to insert an element into the array list at the first position.**

*import* java**.**util**.**ArrayList**;**

*public* class **InsertFirstPosition** {

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

        ArrayList**<**String**>** list **=** **new** ArrayList**<>**()**;**

        list**.**add("B")**;**

        list**.**add("C")**;**

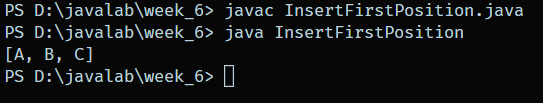
        list**.**add(0**,** "A")**;** *// Insert at first position*

        System**.***out***.**println(list)**;** *// [A, B, C]*

    }

}

**Output –**



**WAP to iterate a linked list in reverse order.**

*import* java**.**util**.**LinkedList**;**

*import* java**.**util**.**ListIterator**;**

*public* class **LinkedListReverse** {

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

        LinkedList**<**Integer**>** list **=** **new** LinkedList**<>**()**;**

        list**.**add(10)**;** list**.**add(20)**;** list**.**add(30)**;**

        ListIterator**<**Integer**>** iterator **=** list**.**listIterator(list**.**size())**;**

**while** (iterator**.**hasPrevious()) {

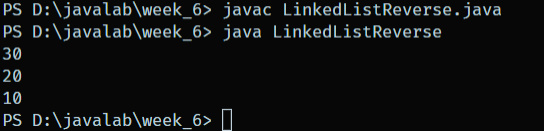
            System**.***out***.**println(iterator**.**previous())**;**

        }

    }

}

**Output –**



**WAP to append the specified element to the end of a hash set.**

*import* java**.**util**.**HashSet**;**

*public* class **HashSetAppend** {

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

        HashSet**<**String**>** set **=** **new** HashSet**<>**()**;**

        set**.**add("One")**;**

        set**.**add("Two")**;**

*// HashSet doesn't guarantee order but adding element appends logically*

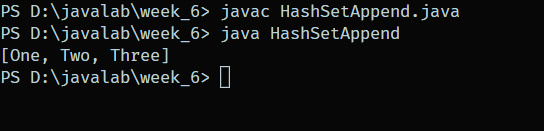
        set**.**add("Three")**;**

        System**.***out***.**println(set)**;**

    }

}

**Output –**



**WAP to add all the elements of a specified to another tree set. tree set.**

*import* java**.**util**.**TreeMap**;**

*public* class **TreeMapSearch** {

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

        TreeMap**<**String**,** Integer**>** map **=** **new** TreeMap**<>**()**;**

        map**.**put("X"**,** 100)**;**

        map**.**put("Y"**,** 200)**;**

**if** (map**.**containsValue(100)) {

            System**.***out***.**println("Value 100 found in TreeMap")**;**

        } **else** {

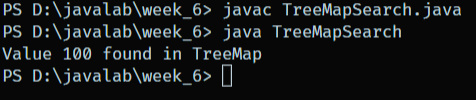
            System**.***out***.**println("Value not found")**;**

        }

    }

}

**Output –**



**WAP to count the number of key-value (size) mappings in a map.**

*import* java**.**util**.**HashMap**;**

*public* class **MapSizeDemo** {

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

        HashMap**<**String**,** Integer**>** map **=** **new** HashMap**<>**()**;**

        map**.**put("A"**,** 1)**;**

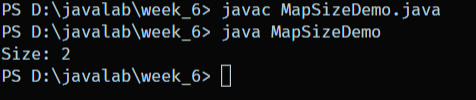
        map**.**put("B"**,** 2)**;**

        System**.***out***.**println("Size: " **+** map**.**size())**;**

    }

}

**Output –**



**WAP to search for a value in a Tree Map. WAP to Demonstrate Iterator**

*import* java**.**util**.**TreeSet**;**

*public* class **TreeSetAddAll** {

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

        TreeSet**<**Integer**>** set1 **=** **new** TreeSet**<>**()**;**

        set1**.**add(10)**;** set1**.**add(20)**;**

        TreeSet**<**Integer**>** set2 **=** **new** TreeSet**<>**()**;**

        set2**.**add(30)**;** set2**.**add(40)**;**

        set1**.**addAll(set2)**;**

        System**.***out***.**println(set1)**;** *// [10, 20, 30, 40]*

    }

}

**Output –**

