ASSIGNMENT - 8

Q1. Write a java program that will create the reference of the thread and display the details of the reference thread. (It should display class, name of thread, priority, group name).

```
class Main {
   public static void main(String[] args) {
Thread thread = Thread.currentThread();
      System.out.println("Class: " + thread.getClass());
      System.out.println("Name: " + thread.getName());
      System.out.println("Priority: " + thread.getPriority());
      System.out.println("Group name: " + thread.getThreadGroup().getName());
  } }
```

OUTPUT:

```
PS C:\Users\Padmalya Meher> javac main.java
PS C:\Users\Padmalya Meher> java main.java
Class: class java.lang.Thread
Name: main
Priority: 5
Group name: main
```

Q2. Write a java program that will create a thread and set the thread name, display the thread name, get the thread id, check the thread is currently alive or not.

```
import java.util.Scanner;  class
Main {
   public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
      Thread thread = new Thread();
      System.out.print("Enter the Thread name to set: ");
thread.setName(sc.next());
      System.out.println("Thread ID: " + thread.threadId());
      System.out.println("Thread is alive?: " + thread.isAlive());
  } }
```

OUTPUT :

```
PS C:\Users\Padmalya Meher>  & 'C:\Program Files\Java\jdk-25\b
C:\Users\Padmalya Meher\AppData\Local\Temp\vscodesws_c99fb\jdt
Enter the Thread name to set: Thread1
Thread ID: 35
Thread is alive?: false
```

Q3. Write a java program that will create one thread [using Runnable interface]. The main thread will read a number and the newly created thread checks the number is Armstrong number or not.

```
import java.util.Scanner;  class Armstrong
implements Runnable {      private int
num;   public Armstrong(int num) {
this.num = num;
   }
   public void run() {         int length =
(int)Math.log10(num) + 1;         int armstrong = 0, temp
= num;         while (temp != 0) {            armstrong +=
(int)Math.pow(temp % 10, length);            temp /= 10;
      }
      if (num == armstrong) {
         System.out.println(num + " is an armstrong number");
      }
 else {
         System.out.println(num + " is not an armstrong number");
      }
 } }
class Main {
   public static void main(String[] args) {         Scanner
sc = new Scanner(System.in);
System.out.print("Enter a number: ");         new
Thread(new Armstrong(sc.nextInt())).start();
   }
}
```

OUTPUT :

```
Enter a number: 2808
2808 is not an armstrong number
```

Q4. Write a java program that will create one thread [using extends]. The main thread will read a number and check the number is prime or composite and the same time the new thread will check the number palindrome or not.

```
import java.util.Scanner;  class
Palindrome extends Thread {
private int num;      public
```

```
Palindrome(int num) {
this.num = num;
   }
   public void run() {         int reverse = 0,
temp = num;         while (temp != 0) {
reverse = reverse * 10 + temp % 10;
temp /= 10;
      }
      if (num == reverse) {
         System.out.println(num + " is an palindrome number");
      } else {
         System.out.println(num + " is not an palindrome number");
      }
 } }
class Main {      public static void
checkPrime(int num) {         boolean
isPrime = true;         for (int i = 2; i <= num /
2; i++) {            if (num % i == 0) {
isPrime = false;               break;
         } }
      if (isPrime) {
         System.out.println(num + " is a prime number");
      }
 else {
         System.out.println(num + " is not a prime number");
      }
   }
   public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.print("Enter a number: ");
int num = sc.nextInt();         new
Palindrome(num).start();
checkPrime(num);
   }
```

}

OUTPUT :

```
PS C:\Users\Padmalya Meher> javac Pallindrome.java
PS C:\Users\Padmalya Meher> java Main
Enter a number: 2467
2467 is a prime number
2467 is not an palindrome number
```

Q5. Write a java program that will create one child thread. The child thread has to display all odd numbers between start and end, and the main thread will display all the even numbers between start and end.

```
import java.util.Scanner;  class
Odd extends Thread {      private
int start, end;      public Odd(int
start, int end) {         this.start =
start;         this.end = end;
   }
   public void run() {
      System.out.print("Odd numbers between " + start + " and " + end + " are:");
for (int i = start; i <= end; i++) {             if (i % 2 == 1) {
            System.out.print(" " + i);
         } }
      System.out.println();
   }
}
class Main {      public static void printEven(int
start, int end) {         System.out.print("Even
numbers between " + start + " and " + end + "
are:");         for (int i = start; i <= end; i++) {
if (i % 2 == 0) {
            System.out.print(" " + i);
         } }
      System.out.println();
   }
   public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.print("Enter two numbers: ");
```

```
int start = sc.nextInt();         int end =
sc.nextInt();
      Thread thread = new Odd(start, end);
thread.start();
      try {
thread.join();
      }
catch (InterruptedException e) {
         System.out.println("Thread was interrupted");
      }
      printEven(start, end);
  } }
```

OUTPUT :

```
PS C:\Users\Padmalya Meher> javac Odd.java
PS C:\Users\Padmalya Meher> java Main
Enter two numbers: 34 67
Odd numbers between 34 and 67 are: 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67
Even numbers between 34 and 67 are: 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66
```

Q6. Write a java program to create two threads. First thread should find the square of the number, second thread should find the sum of the digits of the number.

```
import java.util.Scanner;  class
Square extends Thread {
private int num;      public
Square(int num) {
this.num = num;
   }
   public void run() {
      System.out.println("Square of " + num + " is: " + num * num);
   }
}
class DigitSum extends Thread {
private int num;      public
DigitSum(int num) {
this.num = num;
   }
  public void run() {         int
sum = 0, temp = num;
```

```
while (temp != 0) {
sum += temp % 10;
temp /= 10;
      }
      System.out.println("Sum of the digits of " + num + " is: " + sum);
  } }
class Main {
  public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.print("Enter a numbers: ");
int num = sc.nextInt();         new
Square(num).start();         new
DigitSum(num).start();
  }
}
```

OUTPUT :

```
PS C:\Users\Padmalya Meher> javac Square.java
PS C:\Users\Padmalya Meher> java Main
Enter a numbers: 38
Sum of the digits of 38 is: 11
Square of 38 is: 1444
```

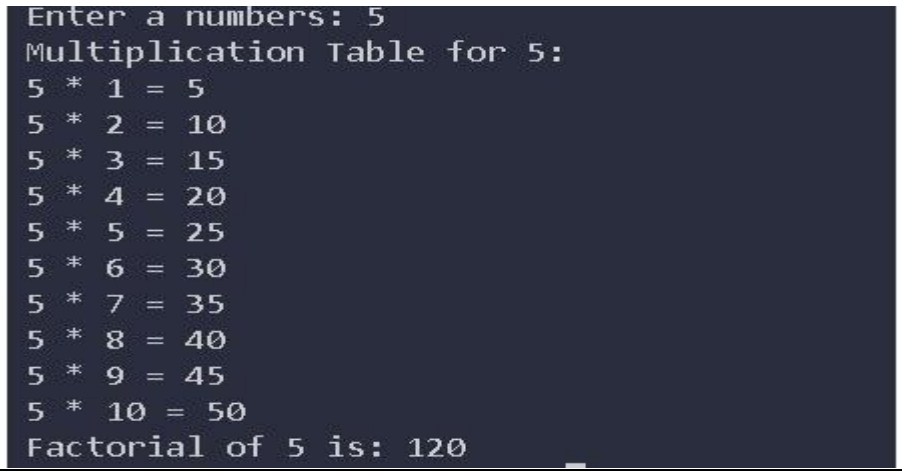
Q7. Write a java program that will create two threads. The main thread will read a number and one thread will print the multiplication table of the entered number and the same time the other thread will find the factorial of the entered number.

Note: Main thread has to wait until other two threads have completed their task.

```
import java.util.Scanner;
class MultiplicationTable extends Thread {
private int num;     public
MultiplicationTable(int num) {
this.num = num;
  }
  public void run() {
     System.out.println("Multiplication Table for " + num + ":");
for (int i = 1; i <= 10; i++) {
        System.out.println(num + " * " + i + " = " + num * i
 } }
```

```
}
class Factorial extends Thread {
private int num;     public
Factorial(int num) {
this.num = num;
   }
   public void run() throws IllegalArgumentException {
int fact = 1;          if (num < 0) {
         throw new IllegalArgumentException("Factorial of negative number is undefined");
      }  for (int i = 2; i <= num; i++) {
fact *= i;
      }
      System.out.println("Factorial of " + num + " is: " + fact);
   }
}
class Main {
   public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.print("Enter a numbers: ");         int
num = sc.nextInt();
      Thread thread1 = new MultiplicationTable(num);
thread1.start();   try {
         thread1.join();
      } catch (InterruptedException e) {
         System.out.println("Thread was interrupted");
      }
      Thread thread2 = new Factorial(num);
thread2.start();
      try {
         thread2.join();
      } catch (InterruptedException e) {
         System.out.println("Thread was interrupted");
 }
   } }
```

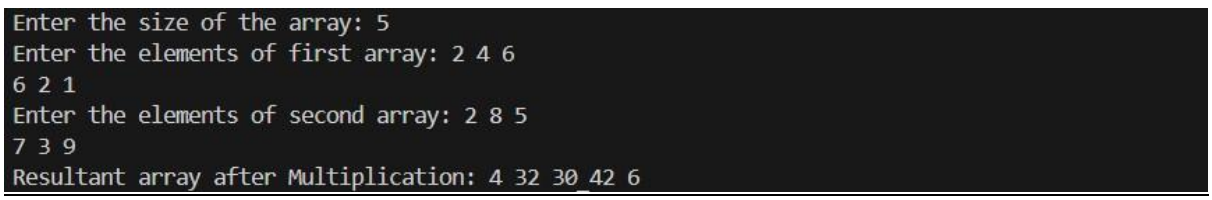
OUTPUT :



Q8. Write a java program that will create two threads. Set the priority to each thread and display it.

```
import java.util.Scanner;  class Main {
   public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
      Thread thread1 = new Thread();
      Thread thread2 = new Thread();
      System.out.println("Enter the Priority of First Thread: ");
thread1.setPriority(sc.nextInt());
      System.out.println("Enter the Priority of Second Thread: ");
thread2.setPriority(sc.nextInt());
      System.out.println("First Thread Priority: " + thread1.getPriority());
      System.out.println("Second Thread Priority: " + thread2.getPriority());
   }
}
```

OUTPUT :



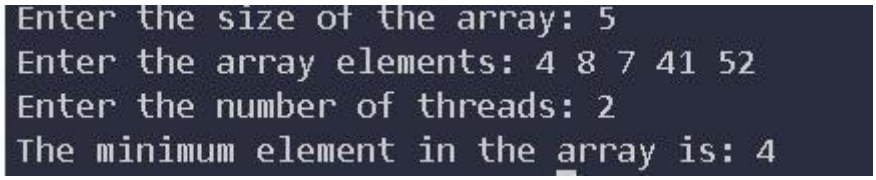
Q9. Write a java program that will compute product of two 1D arrays using multithreading. The program should read two 1D arrays of same size from the user. First thread should multiply the corresponding elements present in the odd index position and second thread should multiply the corresponding elements present in the even index position. Main thread should display the result.

```
import java.util.Scanner;  class
Multiply extends Thread {
```

```
   int[] lhs, rhs, res;      String type;    public Multiply(int[] lhs, int[] rhs, int[] res, String type)
throws IllegalArgumentException {         if (!type.equals("even") && !type.equals("odd")) {
          throw new IllegalArgumentException("Invalid type argument. Can only be (\"even\" or          \"odd\")");
       }
       this.lhs = lhs;
this.rhs = rhs;         this.res
= res;         this.type =
type;
   }
   public void run() {         for (int i = type.equals("even") ? 0 : 1; i
< lhs.length; i += 2) {             res[i] = lhs[i] * rhs[i];
       }
   } }
class Main {
   public static int[] createArray(int size, String name) {
Scanner sc = new Scanner(System.in);
       int[] array = new int[size];
       System.out.print("Enter the elements of " + name + " array: ");  for
       (int i = 0; i < size; i++) {
          array[i] = sc.nextInt();
       }
       return array;
   }
   public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
       System.out.print("Enter the size of the array: ");
int size = sc.nextInt();         int[] lhs = createArray(size,
"first");         int[] rhs = createArray(size, "second");
int[] res = new int[size];
       Thread thread1 = new Multiply(lhs, rhs, res, "even");
Thread thread2 = new Multiply(lhs, rhs, res, "odd");
thread1.start();         thread2.start();    try {
 thread1.join();
          thread2.join();
```

```
        }
catch (InterruptedException e) {
            System.out.println("Thread was interrupted");
        }
        System.out.print("Resultant array after Multiplication:");
for (int value : res) {
            System.out.print(" " + value);
        }
        System.out.println();
    }
}
```

OUTPUT :



Q10. Write a simple Java thread program to compute the sum of n natural numbers. The program should read the number of threads m and value of n from the user. Each of the threads should add its share of assigned number to a global variable. When all the threads are done, the global variable should contain the result. The program should use a Synchronized block to make sure that only one thread is updating the global variable at a given time.

```
import java.util.Scanner;  class SumOfNaturalNumber
implements Runnable {
    private int n, i = 1, sum = 0;     public SumOfNaturalNumber(int n) throws
IllegalArgumentException {         if (n <= 0) {
            throw new IllegalArgumentException("Enter valid Natural Number. n: " + n);
        }
        this.n = n;
    }
    public int getSum() {
return sum;
    }
    public void run() {
while (i <= n) {
synchronized (this) {
sum += i++;
```

```
        }
    }
  }
}
class Main {
   public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
      System.out.print("Enter the number to calculate the sum: ");
int n = sc.nextInt();
      System.out.print("Enter the number of threads: ");
      int m = sc.nextInt();
      SumOfNaturalNumber sumOfNaturalNumber = new SumOfNaturalNumber(n);
      Thread[] threads = new Thread[m];
      for (int i = 0; i < m; i++) {
         threads[i] = new Thread(sumOfNaturalNumber);
      }
for (int i = 0; i < m; i++) {
threads[i].start();
      }
      for (int i = 0; i < m; i++) {
         try {
threads[i].join();
         } catch (InterruptedException e) {
            System.out.println("Thread was interrupted");
         } }
      System.out.println("Sum of the " + n + " Natural Numbers are: " + sumOfNaturalNumber.getSum());
   }
}
```

OUTPUT :

```
Enter the number to calculate the sum: 2808
Enter the number of threads: 3
Sum of the 2808 Natural Numbers are: 3943836
```

Q11. Write a Java thread program to search the minimum number in a given array. The program should read the number of elements in the array, number of threads to be created and the array elements from the user. Each thread should find minimum element in an assigned block of

elements and compare to global minimum element. When all the threads are done, the global variable should contain the minimum element. It should use a Synchronized block to make sure that only one thread is updating the global minimum variable at any given time.

```
import java.util.Scanner;  class MinimumElement extends Thread {      private static int min =
Integer.MAX_VALUE;      private int[] array;      private int start, end;      public
MinimumElement(int[] array, int start, int end) throws IllegalArgumentException {
      if (start < 0 || start > array.length || end < 0 || end > array.length) {
         throw new IllegalArgumentException("Invalid index of start or end. start: " + start + " and end: " + end);
}
this.array = array;
this.start = start;
this.end = end;
   }
   private static synchronized void setMin(int value) {
if (value < min) {             min = value;
      }
}
   public static int getMin() {
return min;    }     public
void run() { int localMin =
min;
      for (int i = start; i < end; i++) {
if (array[i] < localMin) {
localMin = array[i];
         } }
      setMin(localMin);
   }
}
class Main {
   public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
      System.out.print("Enter the size of the array: ");
int size = sc.nextInt();         int[] array = new int[size];
      System.out.print("Enter the array elements: ");
```

```
        for (int i = 0; i < size; i++) {
array[i] = sc.nextInt();
        }
        System.out.print("Enter the number of threads: ");
int m = sc.nextInt();

       int blockSize = size / m;
       Thread[] threads = new Thread[m];
        for (int i = 0; i < m; i++) {
threads[i] = new MinimumElement(array, i * blockSize, i == m - 1 ? size : (i + 1) + blockSize);
        }
        for (int i = 0; i < m; i++) {
threads[i].start();
        }
        for (int i = 0; i < m; i++) {
           try {
threads[i].join();
           }
 catch (InterruptedException e) {
               System.out.println("Thread was interrupted");
           } }
        System.out.println("The minimum element in the array is: " + MinimumElement.getMin());
    }
}
```

OPUTPUT :

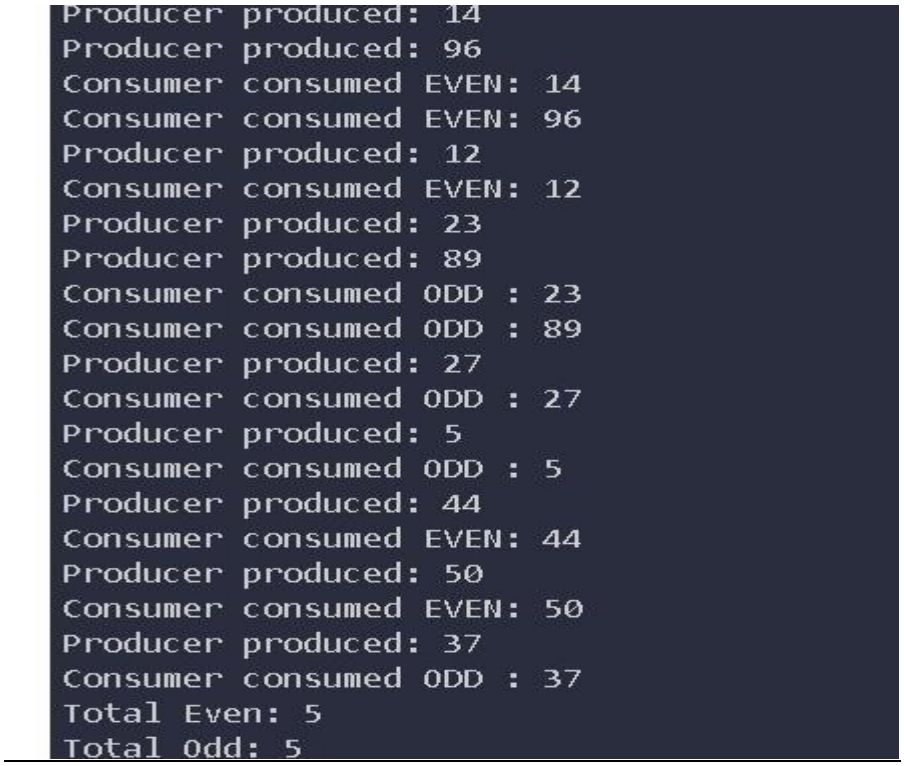
```
PS C:\Users\Padmalya Meher>  & 'C:\Program Files\Java\jdk-25\bi
XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Padmaly
sws_06023\jdt_ws\jdt.ls-java-project\bin' 'Main'
Enter the size of the array: 6
Enter the array elements: 23 6 41 38 52 75
Enter the number of threads: 4
The minimum element in the array is: 6
```

Q12. Write a java program in which main thread should create two child threads (Producer and Consumer). First child thread (Producer) should produce ten random integers between 1 to 100 and the second child thread (Consumer) should check whether the generated number is even or odd. At the end the second child thread (Consumer) should print total number of even numbers received. Both the threads should wait and notify each other wherever necessary. import

```
java.util.Random;  class Buffer {      private int value;      private boolean hasValue = false;      public
synchronized void produce(int product) {          while (hasValue) {
          try {
wait();
          }
 catch (InterruptedException e) {
              System.out.println("Producer Thread was interrupted");
          } }
       value = product;
hasValue = true;
       System.out.println("Producer produced: " + product);          notify();
   }
   public synchronized int consume() {
while (!hasValue) {
          try {
wait();
          } catch (InterruptedException e) {
              System.out.println("Consumer Thread was interrupted");
          }
       }
       hasValue = false;
notify();         return
value;
   }
 }
class Producer extends Thread {
   Buffer buffer;
   Producer(Buffer buffer) {this.buffer = buffer;}      public
void run() {
       Random random = new Random();
       for (int i = 0; i < 10; i++) {
buffer.produce(random.nextInt(100) + 1);
       }
```

```
   } }
class Consumer extends Thread {
   Buffer buffer;
Consumer(Buffer buffer) {this.buffer = buffer;}
public void run() {         int even = 0, odd = 0;
for (int i = 0; i < 10; i++) {              int num =
buffer.consume();             if (num % 2 == 0) {
             System.out.println("Consumer consumed EVEN: " + num);
even++;
          }
else {
             System.out.println("Consumer consumed ODD : " + num);
odd++;
          }
 }
       System.out.println("Total Even: " + even);
       System.out.println("Total Odd: " + odd);
   }
}
class Main {
   public static void main(String[] args) {
Buffer buffer = new Buffer();         new
Producer(buffer).start();         new
Consumer(buffer).start();
   }
}
```

OUTPUT :



NAME – PADMALAYA MEHER

ROLL NO. – 01 (B1)

DATE OE EXP. – 31/10/2025