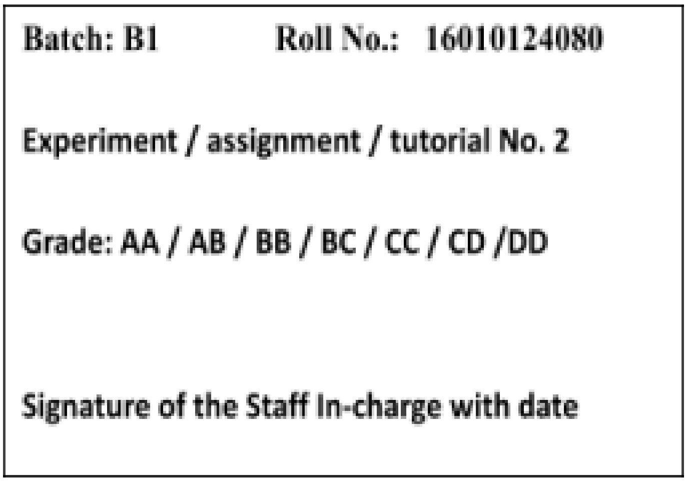


Batch: B1 Roll No.: 16010124080

Experiment / assignment / tutorial No. 2

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

TITLE : Control Statements

AIM:

Write a Java program to generate and show all Kaprekar numbers less than 1000. In number theory, a Kaprekar number for a given base is a non-negative integer, the representation of whose square in that base can be split into two parts that add up to the original number again. For instance, 45 is a Kaprekar number, because 452 = 2025 and 20 + 25 = 45.

Expected OUTCOME of Experiment:

CO1:Apply the features of object oriented programming languages. (C++ and Java)

CO2:Explore arrays, vectors, classes and objects in C++ and Java

Books/ Journals/ Websites referred:

1. E. Balagurusamy, “Programming with Java”, McGraw-Hill.
2. E. Balagurusamy, “Object Oriented Programming with C++”, McGraw-Hill.

Pre Lab/ Prior Concepts:

Java basic constructs (like if else statement, control structures, and data types Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages −

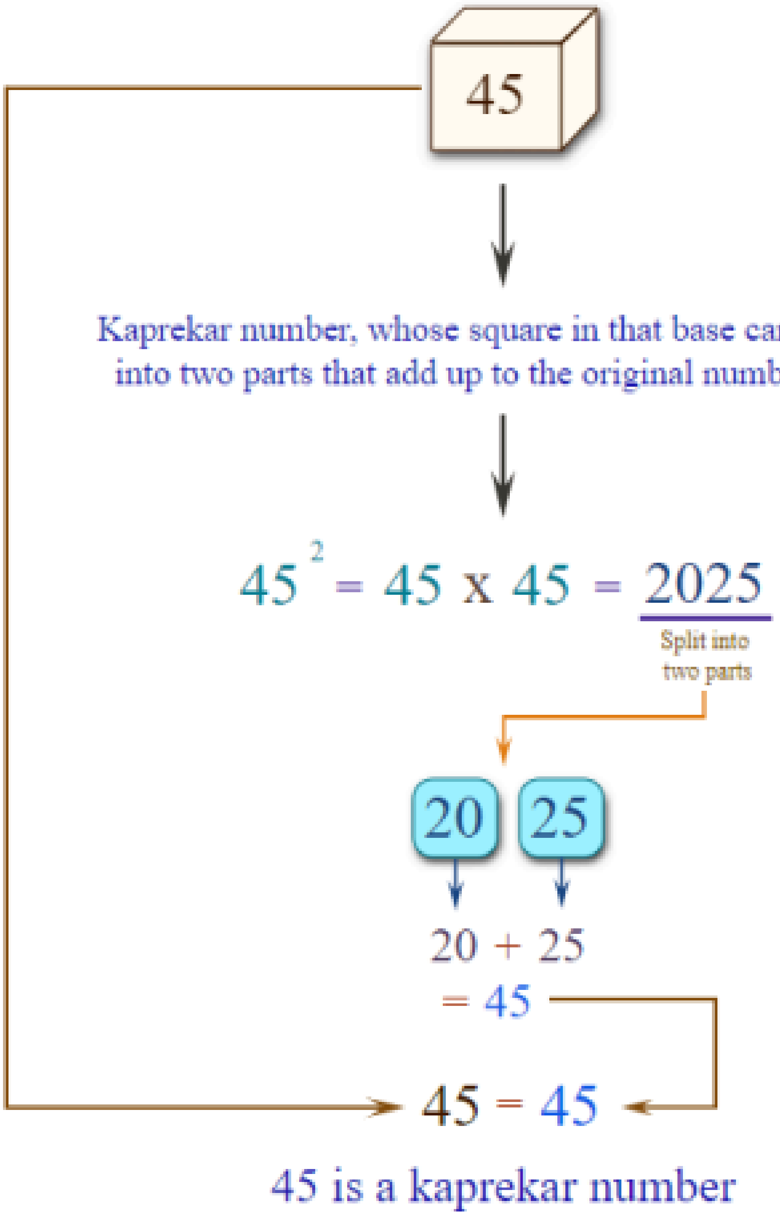
Sr.No.	Loop & Description
1	while loop Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
2	for loop Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.
3	do...while loop Like a while statement, except that it tests the condition at the end of the loop body.

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

Java supports the following control statements. Click the following links to check their details.

Sr.No.	Control Statement & Description
1	break statement Terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch.
2	continue statement Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.


45
Kaprekar number, whose square in that base can be split into two parts that add up to the original number again
45² = 45 x 45 = 2025
Split into two parts
20
25
20 + 25
= 45
45 = 45
45 is a kaprekar number

In number theory, a Kaprekar number for a given base is a non-negative integer, the representation of whose square in that base can be split into two parts that add up to the original number again. For instance, 45 is a Kaprekar number, because 452 = 2025 and 20 + 25 = 45.

Algorithm:

Start

Loop through numbers from 1 to 999

For each number, check if it is a Kaprekar number using the kaprekar function

In kaprekar(num):

- Calculate the square of the number
- Count the number of digits in the square using length_of_num function
- Calculate the partition point as 10 raised to (total digits - total digits divided by 2)
- Split the square into two parts:
    - right part is square modulo partition
    - left part is square divided by partition
- If the sum of left and right equals the original number, or if the number is 0, return 1 (true)
- Otherwise, return 0 (false)

If kaprekar returns 1, print the number

Repeat for the next number

End

Implementation details:

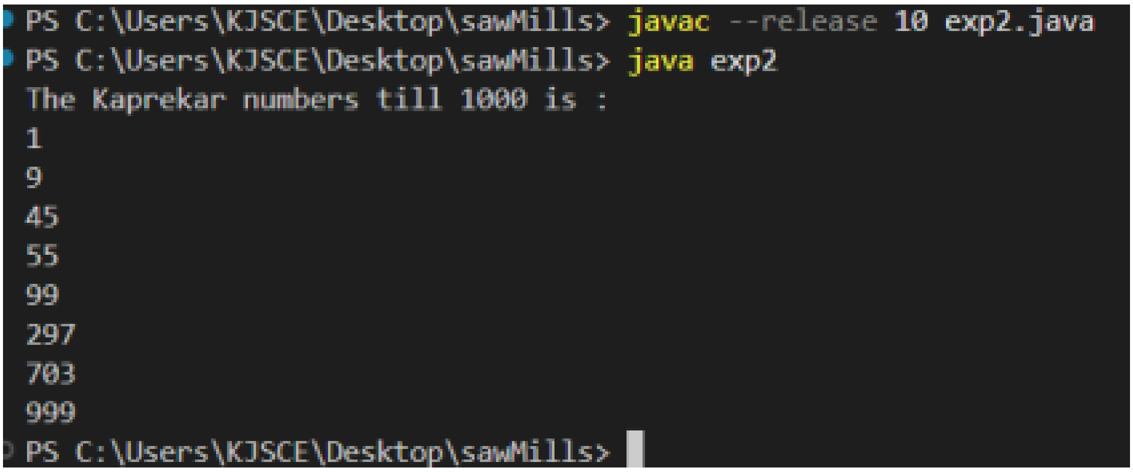
```
Shounak Dutta <shounak.d@somaiya.edu

import java.util.*;

public class exp2{

 public static int length_of_num(int num){

 int len=0;

 while(num!=0){

 num=num/10;

 len=len+1;

 }

 return (len);

 }

 public static int kaprekar(int num ){

 int left=0,right=0;

 int square_of_num= (int)Math.pow(num,2);

 int n= length_of_num(square_of_num);


 int partition = (int) Math.pow(10, n- n / 2);

right=square_of_num%partition;

 left= square_of_num/partition;
```

```
if (num==0){
return 1;
}
if (left+right==num){
return 1;
}
else {
return 0;
}
}
public static void main(String[] args){
System.out.println("The Kaprekar numbers till 1000 is :");
for (int i =1; i<1000;i++){
int ans=kaprekar(i);
if (ans==1){
System.out.println(i);
}
}
```

```
}
```

```
}
```

Output:



```
PS C:\Users\KJSCE\Desktop\sawMills> javac --release 10 exp2.java
PS C:\Users\KJSCE\Desktop\sawMills> java exp2
The Kaprekar numbers till 1000 is :
1
9
45
55
99
297
703
999
PS C:\Users\KJSCE\Desktop\sawMills>
```

Conclusion:

The algorithm for finding Kaprekar numbers efficiently identifies numbers whose square can be split into two parts that sum up to the original number. By squaring the number, determining the number of digits, and logically dividing the square into left and right parts, the algorithm checks for the Kaprekar condition without converting numbers to strings. This method is both mathematically sound and computationally

simple, making it an effective approach for identifying Kaprekar numbers within a given range.

Date: 01/08/25 Signature of faculty in-charge

<u>Post Lab Descriptive Questions:</u>

Q.1 Write a program to find the largest of three numbers using the if-else construct. import java.util.Scanner;

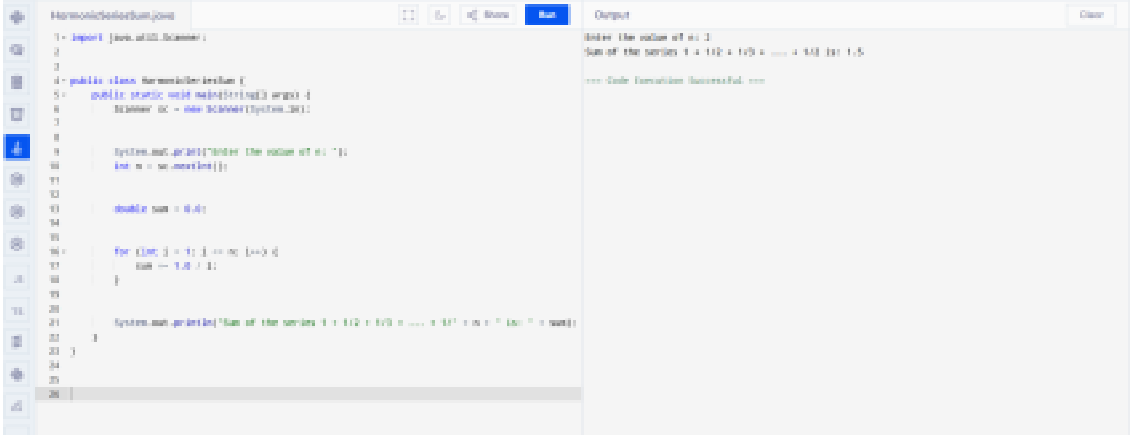
```
public class LargestOfThree {
 public static void main(String[] args) {
 Scanner sc = new Scanner(System.in);

 System.out.print("Enter first number: ");
 int a = sc.nextInt();
 System.out.print("Enter second number: ");
 int b = sc.nextInt();

 System.out.print("Enter third number: ");
 int c = sc.nextInt();

 int largest;
```

```
 if (a >= b && a >= c) {
 largest = a;
 } else if (b >= a && b >= c) {
 largest = b;
 } else {
 largest = c;
 }

 System.out.println("The largest number is: " + largest);
 }
}
```

Q.2 Write a program to determine the sum of the following series for a given value of n:1+½+⅓+....+1/n

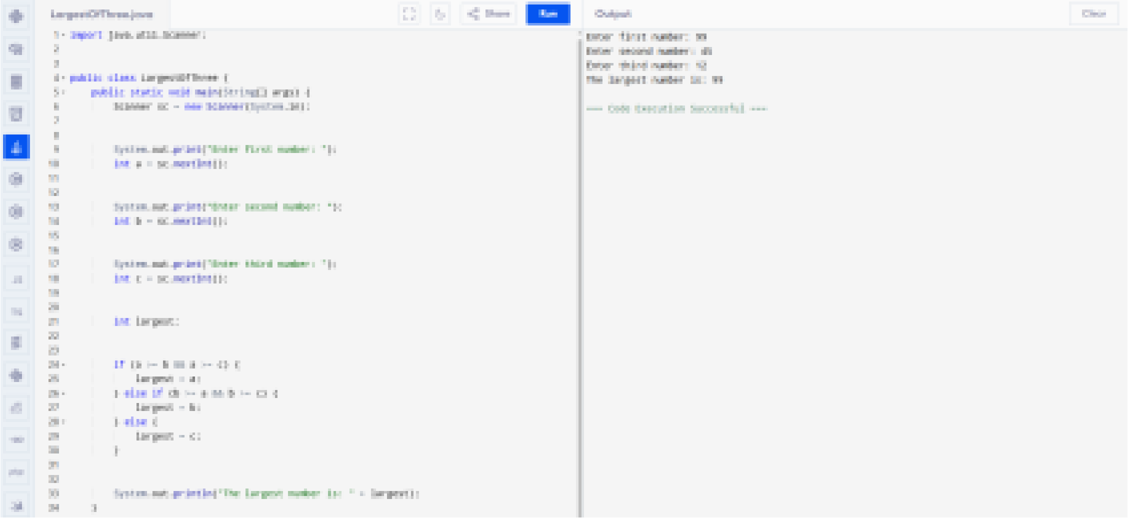
```
import java.util.Scanner;

public class HarmonicSeriesSum {
 public static void main(String[] args) {
 Scanner sc = new Scanner(System.in);

 System.out.print("Enter the value of n: ");
 int n = sc.nextInt();

 double sum = 0.0;
for (int i = 1; i <= n; i++) {
sum += 1.0 / i;
}

System.out.println("Sum of the series 1 + 1/2 + 1/3 + ... + 1/" + n + " is: " + sum);  }
}
```

Output:



```
import java.util.Scanner;


public class HarmonicSeriesSum {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);


        System.out.print("Enter the value of n: ");
        int n = sc.nextInt();


        double sum = 0.0;


        for (int i = 1; i <= n; i++) {
            sum += 1.0 / i;
        }


        System.out.println("Sum of the series 1 + 1/2 + 1/3 + ... + 1/" + n + " is: " + sum);
    }
}
```

```
Enter the value of n: 2
Sum of the series 1 + 1/2 + 1/3 + ... + 1/2 is: 1.5

=== Code Execution Successful ===
```



```
import java.util.Scanner;


public class LargestOfThree {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);


        System.out.print("Enter first number: ");
        int a = sc.nextInt();


        System.out.print("Enter second number: ");
        int b = sc.nextInt();


        System.out.print("Enter third number: ");
        int c = sc.nextInt();


        int largest;


        if (a >= b && a >= c) {
            largest = a;
        } else if (b >= a && b >= c) {
            largest = b;
        } else {
            largest = c;
        }


        System.out.println("The largest number is: " + largest);
    }
```

```
Enter first number: 99
Enter second number: 45
Enter third number: 12
The largest number is: 99

=== Code Execution Successful ===
```