Assignment one of forty IT23044

```
import java.io.*;
import java.util.*;
public class Assignmentoneofforty {
public static void main (String[] args){
 try (Scanner scanner = new Scanner(new File("input.txt"));
      PrintWriter writer = new PrintWriter(new
                                File("output.txt"))){
  if (!scanner.hasNextLine) {
   System.out.println("Error: Input file is Empty");
    return;
   }
String line = scanner.nextLine().trim();
 if (line.isEmpty) {
  System.out.println("Error: Input file only contain whitespace");
   return;
  }
String[] number = line.split(",\\s*");
int highestnumber = Integer.MIN_VALUE;
```

```
for (String numStr : numbers){
    int num = Integer.parseInt(numStr.trim());
    if (num > highestnumber){
        highestnumber = num;
    }
}

long sum = (long) highestNumber * (highestnumber+1)/2;
writer.println("highest number =" + highestnumber + "\n
        sum = " + sum);
System.out.println ("Operation Successfull");
} catch (FileNotFoundException e){
    System.out.println(" File not found ");
}
catch (NumberFormatException e) {
System.out.println (" Invalid number formate ");
}
}
}
```

Assignment two of forty IT23044

- Differences between static and final fields and methods

Feature	static	Final
Defination	Belongs to the class, shared across instance	Cannot be modified once set or overridden
Access	Can be accessed via class name or instance	Accessed like other field method
Purpose	To store or manage class-level data	To make fields immutable or to prevent method overiding

※ What happens if you access a static field or method via an object instead of class name.

⇒ Java allows you to access static fields and methods using either an object or the class name. The preferred and recommended way is to use the class name because it makes it clear that the field or method is associated with the class itself not an instance of the class. No error will be happend.

Assignment three of forty. IT23044

```
import java.util.*;

public class Assignmentthree offorty {
 public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
  System.out.println("Enter the range's lowest value");
    int lowest = input.nextInt();
 System.out.println("Enter the rangeh highest value");
  int highest = input.nextInt();
 System.out.println("Factorion numbers in the range
                are: ");
  find(lowest, higest);

 public static void find(int lowest, int highest) {
  for(int n = lowest; n<= highest; n++) {
    int sum = 0;
    int number = n;
    while(number > 0)
      { int r = number%10;
        sum += factorial(r);
         number = number/10;
      }
```

```
if(sum==n){
    System.out.print(n);
    System.out.print(" ");
}
}
System.out.println("");
}
public static int factorial(int n){
    if (n==0){
        return 1;
    }
    int factorialsum=1;
    while (n>1)
    { factorialsum *= n;
        n--;
    }
    return factorialsum;
}
}
```

A3 Four of fourty IT23064

Difference between class, Local and Instance variable

Attribute	Class variable	Instance variable	Local variable
Declaration	static keyword used	Declared inside a class but outside methods	Declared insi methods
Access	Accessed via class name or object	Accessed via object reference	Accessed only within the method where declared.
Default value	Default value assigned (0 for int)	Default value assigned (null for reference)	No default value must be initializ
Lifetime	As long as the class is loaded into memory	As long as object is alive	During the method execution
Memory location	In the method area (class memory	Stored in the heap (for object instance)	Stored in the stack for method execution

Significance of "this" keyword,

① Refers to the current instance of the class.

② Help differentiate between instance V. and local V./parameter

③ Used for constructor chaining

④ Used to pass the current object to methods or other constructors.

Five of Fonty : IT23044

```
public class Assignmentfiveoffourty{
  public static void main(String[] args){
    int[] arrn = {10,12,14,15,16,18,20}
    int result = calculatesum(arrn);
    System.out.println("The sum of the array=" +result);
  }
  public static int calculatesum(int[] arrn){
      int sum=0;
    for(int num: arrn)
      { sum+ = anum
      }return sum;
  }
}
```

Six IT23044

Access modifiers are keywords which define the accessibility of a class and its members. It's used to control the visibility of classes, interfaces, variables, methods, constructors.

Types of modifiers:

Public → accessible from anywhere

Private → accessible within the same class.

protected → must be in subclass. accessible within the same package and by subclasses.

default → accessible only within the same package and not outside it. It is more restrictive than protected and less than private

※ Variable are written in the previous page.

Seven IT23044

```
import java.util.*;
public class Assignmentsevenofsifty{
 public static void main (String[] args){
  Scanner input = new Scanner (System.in);
  System.out.print("Enter coefficient a, b, and c");
  double a = input.nexDouble();
  double b = input.nextDouble();
  double c = input.nexDouble();
  double d = b*b - 4*a*c;
  if (d<0){
    System.out.println("No real roots");
  }
  else {double determinate = Math.sqrt(d);
        double x1 = (-b + determinate)/(2*a);
        double x2 = (-b - determinate)/(2*a);
        double result = Math.min(x1, x2);
       System.out.println("result=" + result);
  }
 }
}
```

Eight IT23044

```
import java.util.*;

public class Assignmenteightofdouty {
  public static void main(String[] args){
    Scanner input = new Scanner(System.in);
    System.out.println("Enter a string");
    String s = input.nextLine();
   Char[] characters = s.toCharArray;
   determinecharactentype(characters);
}

public static void determinechanactentype(chan[]
                                          chanacte
{  for (chan ch : character){
    if (Chairacter.isLetter(ch)){
      System.out.println(ch+"is a Letter");
      }
    else if ( Character.is Digit){
      System.out.println(ch" is a digit");
    }
    else it (Character.is Whitespace"){
      System.out.println("Whitespace detected");
      }
```

IT23044

```
else {
    System.out.println(ch+" is a special character");
    }
}
}
{
{
```

In this code "determinecharactertype (characters)"

this line is a array passing line

Nine IT23044

Method overriding (Inheritence context)

Method overriding is a feature of inheritence that allows a subclass to provide a specific implementa for a method that is already defined in it's superc

Method overriding:

- When a subclass provides its own implementation of a method that is already present in the superclass it is called method overriding.
- The method in the subclass should be same as the superclass (name, return type, parameter)
- When invoke the method on an instance of the subclass, the overridden version of the method will be executed

The super keyword is used to refer to the superclass member's (fields, methods, constructor)

super.methodname().

Issues with Overriding

- Constructor are not inherited by subclass. That cannot override a constructor.
- If a subclass constructor does not explicitly call a superclass constructor the default constructor of the superclass will be called automitacally.
- If the superclass does not have default constructor the subclass must explicitly call one of superclass constructor.

Difference between Static and non-Static Members

Feature	Static	Non-Static
Definition	Belong to the class shared among all instance	Belong to individual objects (instance) of the class
Access	Accessed without creating an object	Cannot accessed without object
Memory allocation	Allocated once when the class is loaded	Allocated each time an object is created
Example	static int count;	int Id;

* Pallindrom checker (Number & String)

```
import java.util.*;
public class Assignmentten {
 public static void main(String[ ] args)
   Scanner input = new Scanner(System.in);
   System.out.println("Enter an integen number");
   int n = input.nextInt ();
    input.nextLine ();
   System.out.println("Enter a string");
   String s= input.nextLine( );
   int rev=checkpallindrome(n)
  String revs = checkpal(s);
   if (n=rev){
       System.out.println("pallindrome");
       }
   elese { System.out.println("Not pallindrome"); }
 if (s.equals(revs)){
        System.out.println("pallindrome");
       }
```

```
else{
    System.out.println("Not pallindrome");
    }
}

public static int checkpallindrome(int n){
    int reverse=0
    while(n>0){
        int r= n%10;
        reverse=reverse*10+r;
        n=n/10;
    }
    return reverse;
}

public static String checkpal(String s){
    String ss= new StringBuilder(s).reverse().toString(
    return ss;
}
}
```