Assignment one of fonty IT23044

```
import java.io.*;
import java.util.*;
public class Assignmentoneoffonty {
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;
```

IT23044

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

long sum = (long) highestNumber *(highestnumen+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 overidding

※ 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;

}
```

IT 23044

```
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;
}
}
```

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 inside 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 initialized
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./parame

③ Used for constructor chaining

④ Used to pass the current object to methods or at constructor.

Five of Forty : 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 Assignmenteightofdonty {
  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();
    determinecharactertype(characters);
  }

public static void determinecharactertype(char[] characters
{ for (char ch : characters) {
    if (Character.isLetter(ch)){
      System.out.println(ch + " is a Letter");
    }
    else if (Character.isDigit(ch)){
      System.out.println(ch " is a digit");
    }
    else if (Character.isWhitespace(ch)){
      System.out.println("Whitespace detected");
    }
```

IT23064

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

In this code "determinecharactertype (characters)"

this line is a array parsing line

Nine IT23044

Method overriding (Inheritence context)

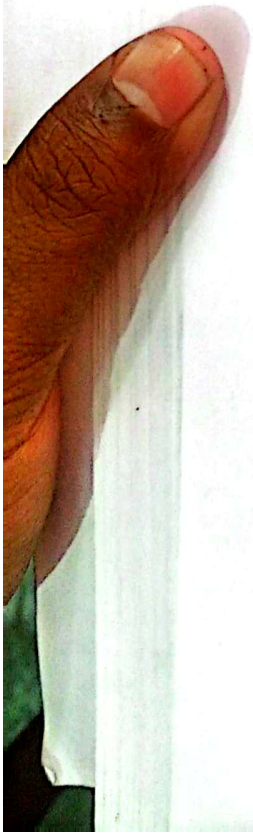
Method overriding is a feature of inheritence that allows a subclass to provide a specific implemented for a method that is already defined in it's superc.

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 automatically.
- If the superclass does not have default constructor the subclass must explicitly call one of superclass constructor.

10 IT23064

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");
      }
   else { 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;
}
}
```

11 IT23044

Abstraction is a method of hiding the unwanted info. Whereas encapsulation is a method to hide the data in a single entity or unit along with a method to protect information from outside.

Difference between abstract class and interface

Feature	Abstract class	Interface (Pure Abs. class)
Definition	A class that contains at least one pure ~~virtual~~ virtual function but can also have normal member functions and attribute.	Contains only pure virtual function and no data members.
Function type	pure virtual, normal virtual, concrete function	Only pure virtual function (all function must be overridden
Variable	Can have members	Cannot have member (only constant if needed)
Access Specifier	Can have private, protected and public	By default all methods are public
Usage	When some implementation is shared among subclass	when multiple classes need to follow a strict contract
Multiple inheritence	Support single and multiple inheritence	Used for multiple inheritence
Construction	✓	✗

12 IT23044

```
public class Assignmen12{
 public static void main(String[] args){
  Sumclass sumobj = new Sumclass();
  double sumresult= sumobj.computesum();
  sumobj.printresult("Sum of series :"+sumresult);
  DivisonMultipleClass gcdlcmobj = new DivisorMultipleClass();
   int n1 = 24, n2 = 36;
   int gcd = gcdlcmobj.gcd(n1, n2);
   int lcm = gcdlcmobj.lcm(n1, n2);
   gcdlcmobj.printresult("gcd = "+gcd);
   gcdlcmobj.printresult("Lcm = "+lcm);
   Numberconversionclass converte = new Numberconversionclass();
   int number= 29;
  converter.printresult("Binary of "+number+" is = " +
                        converter.toBinary(number));
  converter.printresult("Hexadecimal of "+number+" is = "+
                        converter.toHexdec(number));
  converter.printresult("Octal of "+number+" is = " +
                        converter.toOctal(number));
```

```
Customprintclass customprint = new Customprintclass();
Customprint.pr("Programme Execution completed");
}
}
class Baseclass {
void printresult(String result){
System.out.println(result);
}
}
class sumclass extends Baseclass{
double computesum() {
double sum = 0
for (double i = 1.0; i >= 0.1 i -= 0.1) {
sum += i;
}
return sum;
}
}
```

```
class DivisorMultipleclass extends Baseclass{
    int gcd(int a, int b){
     return (b=0)? a : gcd(b, a%.b); }

    int lcm (int a, int b) {
        return (a*b) / gcd(a,b);
    }
}

class Numberconversionclass extends Baseclass {
    String toBinary(int number) {
     return Integer.toBinaryString (number);
    }
    String toHex (int number){
     return Integer.toHexString (number); }
    String toOctal (int num) {
     return Integer.toOctalStnint(number),
    }

class customprintclass extends Baseclass {
 void pr(String message){
    System.out.println(" " +message{;
    } }
```

25 IT23064

Issues to Consider While Handling Exception

① Correct Exception Type: Lik (ArithmaticException, IO Exception, NULLPointerException) rather than that Exception.

② Proper placement of try-catch Blocks

③ Resource management: Use finally

④ Don't Suppress Exception: Avoid empty catch blocks.

⑤ Meaningful Error message;

⑥ Custom Exception:

* Calculat Area of a Circle

```
public class Assignment 25 {
  public static void main (String[] args) {
    Circle circle = new Circle ();
  try { circle.setRadius (-5);
        double area = circle.calculateArea();
        System.out.println("Area = " + area);
  }
  catch (InvalidRadiusException e) {
```

```
        System.out.println("Exception Caught" + e.getmessage
    }
    finall {
        System.out.println("Programme Execution Completed")
    }
}
class InvalidRadiusException extends Exception {
    public InvalidRadiusException(String message){
        super(message);
    }}
class Circle {
    private double radius;
    public void setRadius(double radius) throws
    InvalidRadiusExeception { if (radius < 0) {
            throw new InvalidRadiusException("
            Redis cannot negative");
    }
    this.radius = radius;
public double calculateArea() {
    return Math.PI * radius * radius; }}
```

26 IT23044

In java there are two ways to create a thread

① By extending the Thread class

- Override the run method in a subclass of Thread
- Create an instance and call start();

② By implementing Runnable Interface

- Implement Runnable and override the run() method.
- Pass the instance of the class to a Thread object and call start

```
public class ThreadExample{
public static void main(String[] args){
MyThread thread = new MyThread();
  thread.start();
}}

class MyThread extends Thread{
  public void run(){
  for(int i=0; i<=5; i++){
    System.out.println("Thread is running :"+i);
```

```
try { Thread.sleep(500); }
catch (InterruptedException e){
  System.out.println("Thread interrupted :" + e.me
                                          ssage());
}}}
```

※ By Runnable Interface

```
public class RunnableExample {
  public static void main(String[] args){
    MyRunnable myRunnable = new MyRunnable();
    Thread thread = new Thread(myRunnable);
      thread.start();}}}
class MyRunnable implements Runnable{
  public void run(){
    for(int i=1; i<=5; i++){
    System.out.println("Runnable thread running" + i);
    try { Thread.sleep(500);}
    catch{InterruptedException e)}
}}}}System.out.println("Thread interrupted " + e.get
```

34 IT23044

String S_1 = "This is ICT 2107 Java";

String S_2 = new String ("This is ICT 2107 Java");

String S_3 = "This is ICT 2107 Java";

S_1.equals(S_2) → true [.equals() compares content of String]

S_1 == S_2 → False [== checks memory reference]

S_1 is stored in the string pool while S_2 is created as a new object in the Heap, so they have different references.

S_1 = S_3 → true [→ Since S_3 is also String literal it point to the same memory location as S_1 in the String Pool.];

```
public class CounterTest {
    public static void main (String[] args){
        for(int i=1; i<=55 ; i++){
            new Counterclass ();
            System.out.println("Object"+i+"created. Instan
            count :" + Counterclass.getInstanceCount ());
    }}}

class Counterclass{
    private static int instanceCount = 0;
    public CounterClass (){
        instanceCount ++
        if (instancecount>50) {
            instanceCount = 0 ;}
    }
    public static int getInstanceCount (){
        return instanceCount;
    }}
```

33 IT23044

```
public class ExtremeFinder {
    public static void main (String[] args) {
    int x = findExtreme("Smallest", 5, 2, 9, 1);
    int y = findExtreme("largest, 8, 3, 10, 4);
    System.out.println("Smallest number:"+x);
    System.out.println("Largest number:" + y);
}
public static int findExtreme (String type, int... numbers){
    if (number.length==0){
        throw new IllegalArgumentException("At least one number
            must be provided");
        }
    int result = number[0]
for (int num : numbers) {
    if (type.equalsIgnoreCase("smallest") && num<result){
        result = num;}
    else if (type.equalsIgnorecase("largest") && num>result{
        result = num;}
    }
    return result;
}
```

* When to use an Astnact Clan Over an Intenface

1. Partial Implementation is Needed
2. State (Fields) needs to be shaned
3. Better code Organization
4. Encapsulation of Behavior

* Intenface Uses

1. Multiple Inteniitence in requiined
2. Behavion Contracts
3. Loosely Coupled Design
4. Integnation with thind panty libnanyes

.0 21-3044

Polymorphism: is the ability of an object to take many forms. It allows a single interface to be used for different types, enabling code reusability and flexibility

① Compile-time polymorphism (Method overriding) Resolves method calls at compile time.

② Run time polymorphism → Used dynamic method dispatch to resolve method calls at runtime. This occurs when a subclass overrides a method from its superclass, and the method to be executed is determined by runtime type of the object.

```
public class PolymorphismExample{
    public static void main(String[] args){
        Animal myAnimal;
        myAnimal = new Dog();
        myAnimal.makeSound();
        myAnimal = new Cat();
        myAnimal.makeSound();
    }
}
```

```
class Animal {
    void makeSound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    @Override
    void makeSound() {
        System.out.println("Dog barks");
    }
}

class Cat extend Animal {
    @Override
    void makeSound() {
        System.out.println("Cat meows");
    }
}
```

It's polimorphism because makeSound behaves differently depending on Dog or Cat class

Impact of Polymorphism on Performance:

Performance trade off

Approach	Performance Impact	Use case
Polymorphism (Method Overriding)	Slightly slower (due to dynamic dispatch)	When flexibility and scalability are needed
Direct Method Calls	Faster (resolved at compile time)	When performance is critical
Final Methods (final void method)	Faster	When you don't need to methods overridden

Tarade off Between Polymorphism and Specific Method Calls:

① Advantage of Polymorphism

- Increase code reusability (One interface, multiple implementation.)
- Support extensibility (easy to add new behaviours)
- Helps in maintaining and scaling large projects
- Reduces tight coupling, making code more flexible

Dis Advantage:

- Slight performance overhead due to runtime method resolution
- Can make code harder to debug
- Incorrect design may lead to unintend behavior.

17 IT 23044

Feature	Array list	Linklist
Underlaying data Structeered	Resizable array	Doubly linked list
Access time (get index)	O(1) [Direct accen via index]	O(n) Traversal required
Insention at End	O(1) [Resizing may need]	O(1) [Resizing don't nee
Insention in Middle (add index, E e)	O(n) [Shifting required]	O(n) [Traversal required
Deletion at End [remove Last ()]	O(1)	O(1) [Adjust pointe
Deletion Begining	O(n) [Shifting needed]	O(1) Adjust pointers
Delition middle	O(n)	O(n) Traversal needed
Memory Overhead	Lower (store only element)	Higher (store elements + pointers)
Iteration Perdormence	farter	slower
Seerch perdormence	O(n)	O(n)

Use Arraylist when

① Need fast random access (get index)

② List is mostly read-heavy, with few insertion/removal

③ Memory efficiency is a concern (less overhead)

④ Iteration speed fast

Use Linklist when

① When need frequent insertion and deletion at the begining/middle

② You are dealing with large datasets where shifting is expensive

③ The list size fluctuates frequently and you want to avoid resizing overhead

Impact on Performance for Large Datasets:

- Arraylist is better for large datasets when random access and iteration are needed.
- LinkedList is better when frequent insertion/deletion occurs, but only if memory usage is not a concern
- Linked List has poor cache locality, making iteration slower as compared to Arraylist.

18 IT23044

```
import java.util.* Random;
public class CustomRandomGenerator {
private static final int[] predefinedArray = { 101, 20
                                307, 401, 509};
private static final int maxValue = 1000;
public static int [ ] myRand (int n){
 int[ ] randomNumbers = new int [n];
 long currentTime = System.currentTimeMillis();
 Random rand = new Random (currentTime);
for (int i=0; i<n; i++){
 int index = rand.nextInt(Predefine Array.length);
 randomNumbers [i] = int ((currentTime * predefineArray
                    [index]) % maxValue); }
 return randomNumbers;
}
public static int myRand() {
long currentTime = System.nanoTime();
int index =(int)(currentTime % predefinedArray.l
```

```
return (int) ((currentTime * predefinedArray[index]) %
        maxValue);
public static void main(String[] args) {
System.out.println("Generating 5 random numbers:");
int[] nums1 = myRand(5);
for (int num: nums1){
  System.out.print(num+" ");
  }
System.out.println("Generating a single random number");
  System.out.print(myRand());
  }
}
```

19 IT23044

Multithreading in Java

Multithreading allows multiple parts of the prog to run concurrently, improving performance and resource utilization. It is manged using th Thread class or Runnable interface and java provides built in synchronization mechanism to ha thread safety.

Feature	Thread class	Runnable Interface
Inheritence	Extends Thread class	Implements Runnable
Code reusability	Less reusable	More reusable
Over head	Each thread object has its own instance	Multiple threads can sha the same object
Implementation	Override run() method	Implement run() method

✱ Potential Issue with Multithreading

a. Race Condition → Occur when multiple threads acces shared resources without proper synchronizatio

b. Dead lock → Happens when two or more threads are waiting for each other to release locks, resulting in a permanent halt

c. Starvation: A low prionity thread never gets CPU time because high. prionity threads keep executing.

d. Live clock: Threads keep responding to each other's state but make no progress.

The synchronized keyword (Thread Safety)

20 J123044

Exception Handling in Java is a mechanism that allows a program to gracefully handle runtime errors and prevent crashes. It uses the try-catch-finally construct to detect and handle exceptions.

Feature	Checked Exceptions	Unchecked Exception
Inheritence	Subclass of Exception (except RuntimeException)	Subclass of RuntimeException
Compile time Handling	Must be handled using try-catch or throws	No mandatory handling
Examples	IOException, SQLException, InterruptedException	NullPointerException, AnithmaticException ArrayIndexOutofBound Exception
Usages	Used for recoverable conditions (e.g. file not found)	Used for programm errors (eg. null poin access)

Creating and Throwing Custom Exceptions:

In java, you can create custom exception by extending the Exception class (for checked exceptions) or RuntimeException (for unchecked exception).

```
public class TestChecked{
public static void main(String[] args){
  try{ validate(16);
  } catch(CustomCheckedException e){
  System.out.println("Exception:" + e.getMessage());
  }}
```

~~public class TestChecked~~

```
static void validate(int age) throws CustomChecked
exception{ if(age<18){
throw new CustomCheckedException("Age must be 18
or above"); }}
```

class CustomCheckedException extends Except

public CustomCheckedException(String messag

super(message);

{ }

Role of throw and throws

Keyword	Purpose	Usage
throw	Used to explicitly throw an exception	throw new Illeg Argument Except ("Invalid Input
throws	Declares that a method may throw an exceptio	void method throws exceptio

class Demo {

public static void main(String[] args){

try { checkNumber(-5);

} catch(IllegalArgumentException e) {

System.out.println("Exception caught: "+

e.getMessage()); }