① IT-22023 (Re-add)

DAY: _____

TOPIC NAME: _____ TIME: DATE: / /

```
import java.io.*;
import java.util.*;
Public class enam1 {
    public static void main(String[] args) {
    String input = "input.txt";
    String output = "output.txt";
    Scanner read = new Scanner (new File (input));
    Printwriter write = new Printwriter (new File (output));
    read.useDelimiter(",");
    List<Integer> numbers = new Array List<>();
    while (read.hasNextLine()) {
        numbers.add (read.nextInt()); }
    if(numbers.isEmpty()) {
        write.println("No number found");
        read.close();
        write.close();    return; }
int maxnum = Collections.max(numbers);
write.println("Maximum Number :"+maxnum);
    for(int x :numbers) { int sum = (n*(n+1))/2;
                write.println(sum+",");
                                    }
    write.close();
    read.close();
    }
}
```

Detailed Differences Between static and final in java.

static	final
1. A member (field or method) marked as static belongs to the class rather than any specific instance	1. A member (field or method) marked as final belongs to instance rather than any specific class
2. Applies to variables, methods, blocks, nested classes.	2. variables, methods classes.
3. Shared by all instances of the class. A single copy exits in memory	3. once assigned, its value cannot be changed. if its an object reference the reference cannot be change.
4. Can be called without creating an object (class.meth)	4. cannot be overridden in subclasses but can be inherited.
5. Stored in the method area.	5. Stored in the heap of stack
6. static int count = 0;	6. Final int max = 100;

IT-22023

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Code:

```
class staticEnample {
    Static int count = 0;
     Static void display () {
     System.out.println("static method
          called");
    } }

public class main{
    public static void main(String[] args) {
    system.out.println(static Enample.count);
    Static Enample.display();
                    // works
    Static Enample obj = new static Enample();
     System.out.println(obj.count);

}
}
```

both works

output:

0
static method called
0

(3) IT-22023 (Re-add)

TOPIC NAME: _____ DAY: _____

TIME: DATE: / /

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import java.io.*;
Public class exam2 {
    Public static void main(String[] args) {
 int start, end;
    Scanner sc = new Scanner(System.in);
    start = sc.nextInt();
    end = sc.nextInt();
for (int j = start; j <= end; j++) {
    int num = j;
    int numo = num;
    List<Integer> arr = new ArrayList<>();
    while (num != 0) {
        int n = num % 10;
            num = num/10;   add.add(n);
        }
int sum=0;  for (int x : arr) {
                int mul = 1;
            for (int i=1; i<= n; i++) {
                mul = mul * i; }   sum = sum + mul; }
    If (sum == numo) { System.out.Print(numo + ", ");
}}}}
```

(4) IT-22023

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Class variable, local variable, instance variable differ primarily in their scope, lifetime, and usage within a class:

1. class variables:
   - Declared with the static keyword
   - Belong to the class, not instances of the class
   - Shared by all objeet of the class, meaning changes to the class variable affect all instances.
   - can be accessed using the class name or through an instance

Example: static int count;

2. Instance variable
   - Declared without the static keyword.
   - Belong to a specific instanse of the class.

IT-22023

TOPIC NAME: _____ DAY: _____

TIME: DATE: / /

- Each object of the class has its own copy of the instance variable
- can be accessed using the object reference.

Example: int age;

3. Local variables:
   - Declared within a method, constructor or block.
   - Exist only during the execution of the method or block where they are declared.
   - Do not retain their values between methods calls.
   - Cannot be accessed outside the method, constructor, block.

Example: int sum=0;

IT-22023

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Significance of the this keyword:

- The this keyword is a reference to the current object (the instance of the class) within a non-static method or constructor.
- It is used to distinguish between instance variable and local variable when they have the same name.

```
= class car {
    string color;
        car (string color) {
            this.color = color;
        }
    }
```

(5) IT-22023 (re-add).

TOPIC NAME: _____ DAY: _____

TIME: DATE: / /

```
Public class exam1{
    Public static void main (String[] args)
{   int[] array = {1,2,4,2,1,5,6,9};
    Innerexam - obj = new Innerexam();
    int sum = obj.sumArray (array);
    System.out.Println ("Sum = "+sum);
}}
Public class Innerexam{
    Public in sumArray (int[] arr){
    int sum=0;
      for (int x : arr){
        sum = sum+x;
      }
      return sum;
  }
}
```

(6) IT-22023 (Re-add)

TOPIC NAME: _____ DAY: _____

TIME: DATE: / /

Access Modifiers in Java:

Access modifiers in java define the visibility (accessibility) of classes, methods and variables. They specify who can access a particular class, method, or variable. The four main access modifiers in java are:

1. public:
   - Accessibility: Accessible from any other class, no matter what package the belongs to.
   - Usage: used when we want a class, method, or variable to be accessible from anywhere in the program.

= public class myclass {
    public int number;
}

⑥ IT-22023 (Re-add)

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Access Modifiers in Java:

Access modifiers in java define the visibility (aceessibility) of classes, methods and variables. They specify who can access a particular class, method, or variable. The four main access modifiers in java are:

1. public:
   - Accessibility: Accessible from any other class, no matter what package the belongs to.
   - Osage: used when we want a class, method, or variable to be accessible from anywhere in the program.

```
= public class myclass {
    public int number;
  }
```

⑥ IT-22023 (Re-add)

TOPIC NAME: _____ DAY: _____

TIME: DATE: / /

Access Modifiers in Java:

Access modifiers in java define the visibility (accessibility) of classes, methods and variables. They specify who can access a particular class, method, or variable. The four main access modifiers in java are:

1. public:
   - Accessibility: Accessible from any other class, no matter what package the belongs to.
   - Usage: used when we want a class, method, or variable to be accessible from anywhere in the program.

```
= public class myclass {
    public int number;
  }
```

IT-22023

TOPIC NAME: _____ DAY: _____

TIME: DATE: / /

2. Private:

- Accessibility: Accessible only within the same class. Other classes cannot access private members, even if they belong to the same package.
- Usage: Used when you want to restrict access to class members for encapsulation purposes, usually to ensure data protection.

```
= public class MyClass {
    private int numbers;
    public void setNumbers(int num)
{ numbers = num;
  }
  }
```

IT-22023

TOPIC NAME: _____ DAY: _____

TIME: DATE: / /

3. Protected:

Accessibility: Accessible within the same package and by subclasses (even if they are in a different package)

usage: used when we want a member to be available to subclasses but not accessible to every class.

```
= public class myclass{
      protected int number;
  }
=
```

Types of variable in java:

1. Instance variable

. These variable are declared within a class but outside any method or constructor.

- Each object of the class has its own copy of the instance variable

Example:

```
public class person {
    string name;
}
```

2. class variable:

- These variable are declared with the static keyword.
- They are shared among all instances of the class. There is only one copy of a static variable, it can be accessed directly by the class name or an object.

IT-22023

TOPIC NAME: _____ DAY: _____

TIME: DATE: / /

= Public class Counter {

static int count = 0;

}

3. Local variable:

• These variable are declared within a method or block of code.

• They are only accessible within the method or block in which they are declared and are created when the method is invoked, distroyed when the method completes.

= public class Example {

public void addNumbers () {

int sum = 0;

sum = 5 + 10;

sout (sm); } }

IT-22023

TOPIC NAME: _____ DAY: _____

TIME: DATE: / /

Parameter:

- These variables are used in method declaration to accept values passed when calling the method.

Public int add (int a, int b)

=

(7) IT-22023 (Re-add).

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

```
import java.util.Scanner;
Public class exam1{
    Public static void main(String[] args){
    Scanner sc= new Scanner(System.in);
      int a=sc.nextInt();
     int b=sc.nextInt();
     int c=sc.nextInt();

 if((b*b-4*a*c)<0){
          System.out.println("No real root");
    }
 else{ double x1=(-b+Math.sqrt(b*b-4*a*c))/(2*a);
     double x2=(-b-Math.sqrt(b*b-4*a*c))/(2*a);
     System.out.printf("%f\n",Math.min(x1,x2));
    }
    }
}
```

(8) IT-22023 (Re-add).

TOPIC NAME: _____ DAY: _____

TIME: DATE: / /

```
impont java.util.Scanner;
Public class exam1 {
   Public static void main(String[] args) {
   Scanner sc=new Scanner(system.in);
 character input =sc.nent().charAt(0);

 if(character.isDigit(input)){
        system.out.Println("Its a Digit");
    }
else if(Character.iswhitespace(input)){
   System.out.Println("Its a whitespace");
}
else { system.out.Println("Its a letter");
   }
   }
```

IT-22023

TOPIC NAME: _____ DAY: _____

TIME: DATE: / /

In java we can pass an array to a function by simply specifying the array type in the methods parameter.

```
Public class ArrayEx{
    Public void method(int[] arr){
        |
        |
        |
    }
```

main

```
ArrayEx ss = new ArrayEx();
ss.method(Array);
```

(5) IT-22023 (Re-add).

TOPIC NAME: _____ DAY: _____

TIME: DATE: / /

Method overriding occurs when a subclass provides a specific implementation of a method that is already defined in its superclass. This allows a subclass to modify the behavior of inherited methods.

Key rules of method overriding

1. The method in the subclass must have the same name, return type, and parameters as the method in the superclass.
2. The method in the subclass cannot have a weaker access modifier.
3. The overriding method cannot throw a broader exception than the method in the superclass.

IT-22023

TOPIC NAME: _____ DAY: _____

TIME: DATE: / /

9. Dynamic method Dispatch: when a method is overriden, Java decides which version of the method to execute at run time, based on the actual object type.

The 'super' keyword is used to call the method of a superclass from a subclass, which is useful when you want to extend or modify the inherited behavior instead of completely replacing it.

```
class Animal{
    void sound() {
        system.out.println("
    Animal make sounds");
    }}
class Dog extends Animal {
        @override
```

IT-22023

TOPIC NAME: _____ DAY: _____

TIME: DATE: / /

```
void sound (){
    super. sound ();
system ·out. println ("Dog Barks");
}}
public class Main{
    Public static void main(
    String[] args){
        Dog d = new Dog();
        d. sound ();
    }
}
```

Output:

Animals make sounds
Dog Barks

- Here super. sound () calls the

sound() method of Animal, and then the Dog class adds its own behavior.

⇒ Issues when overriding Methods

. constructors cannot be overrideen

⇒ constructors are not inherited, so they cannot be overridden. However a subclass constructor can explicitly call a superclass constructor using super().

. Constructor call order.

⇒ when creating an object of a subclass, the constructor of the superclass is called first.

⇒ If a superclass does not have a no-argument construc tor

TOPIC NAME: _____ DAY: _____

TIME: DATE: / /

the subclass must explicitly call a parameterized constructor using `super()`:

```
// class parent {
    parent() {
System.out.println("Parent
 constructor");
    }}
class child extends parent {
    child() {
        super();
system.out.println("child
constructor");
    }} public class Main {
Public static void main(String[] args)
{ child c = new child();
    }}                    //
```

IT-22023

TOPIC NAME: _____ DAY: _____

TIME: DATE: / /

Potential Issues with overriding:

. Incorrect 'super()' uses:

⇒ If the superclass has a parameterized constructor but the subclass does not explicitly call it, a compilation error occurs.

⇒ • Final methods cannot be overridden: If a method is declared as a final in the superclass, it cannot be overridden.

. Private methods Are not inherited:

⇒ Private methods in a superclass are not visible in the subclass, so they cannot be overriden.

(10) IT-22023

TOPIC NAME: _____ DAY: _____

TIME: DATE: / /

Difference Between Static and non-Static Members

Static member	non-static member.
① Belong to the class rather than object	1. Belong to a specific object of the class.
② Can be accessed using the class name	② Can only be accessed through an object.
③ Allocated once in a memory and shared across all objects	③ Each object gets its own copy.
④ Classname.method() or object.method()	④ object.method()
⑤ Changeing a static variable affects all objects	⑤ Changing a non-static variable affects only that object.

IT-22023

TOPIC NAME: _____ DAY: _____

TIME: DATE: / /

```
import java.util.Scanner;
Public class exam1{
  Public static void main(String[]
                          args){
Scanner sc=new Scanner(System.in);
String s=sc.NextLine();
int length=s.length();
if(length%2==0){ int a=0,b=length-1;
while(a<b){ if(s.charAt(a)!=s.charAt(b)){
System.out.println("Not a palindrom");
    return;} a++; b--;
  }} else{ int a=0, b=length-1;
while(a<=b){ if(s.charAt(a)==
s.charAt(b)){
        System.out.println("Not a
palindrom"); return;} a++;
                        b--;
}}
system.out.Println("Its a Pallindrom");
}}
```

(11) IT-22023 (Re-add)

TOPIC NAME: _____ DAY: _____

TIME: DATE: / /

Abstraction: Abstraction is the process of hiding implementation details and showing only essential features. In Java abstraction is achieved using abstract classes and interface.

- Focuses on what an object does not how it does.
- Achieved using athe 'abstract' keyword.
- Abstract methods must be implemented by subclasses.

```
// abstract class vehicle{
      abstract void start();
   }
class car entends vehicle {
    void start() {
    System.out.println("Car starts
    with a key.");
    }}.
//
```

IT-22023

TOPIC NAME: _____ DAY: _____

TIME: DATE: / /

Encapsulation: Encapsulation is the process of wrapping data (variables) and methods into a single unit (class) and restricting direct access to it.

- Data is hidden using 'private' variable
- Accessor (getter) and mutator (setter) method are used for controlled access.

```
// class person {
        private string name;

    public void setName(string name) {
            this.name = name; }
    public string getName() {
            return name;
                    }}

public class Main {
        psvm() {
    person p = new Person();
        p.setName("Alice");
sout(p.getName()); }}
```

⇒

TOPIC NAME : IT-22023 DAY :

TIME : DATE : / /

Abstract class	Interface
• A class that can have abstract and concrete methods.	• A blue print that only contains abstract methods
• can have both abstract and non abstract methods.	• only abstract methods
• Can have instance variable	• variables are public static and Final
• A class can extend only one abstract class	• A class can implement multiple interfaces.
• Used when classes share common behavior	• Used to achive full abstraction and multiple inheritance.

(12) IT-22023

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

```
import java.util.*;
Public class Mainclass{
    Public Static void main(String[] args){
    Sumclass work1 = new Sumclass();
    work1.print Result();
    system.out.Printf("%.2f", work1.calculatesum());
    DivisonMultipleClass work2 = new DivisonMultiple-
    -class();
    Scanner sc = new Scanner(system.in);
System.out.Println("Enter 2 number to calculate
gcd ");
        int x = sc.nent Int();
        int y = sc.nent Int();
        work2.print Result();
System.out.println("Enter 2 number to calculate
    lcm");
    x = sc.nent Int();
    y = sc.nent Int();
System.out.Println("LCM => " + work2.LCM(x,y));
```

IT-22023

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

```
NumberConversionClass wonk3 = new NumberCon-
-versionClass ();
        x = sc.nextInt();
    System.out.println(wonk3.decTobinary(n));
        x = sc.nentInt ();
    System.out.println(wonk3.decTOoct(n));
        x = sc.nextInt ();
    System.out.println(wonk3.decTo hex(n));

CustomPrintClass wonk4 = new CustomPrintClass();
        String s = sc.nentLine();
    System.out.println(wonk4.prn());
}
}
```

=

IT-22023

TOPIC NAME: _____ DAY: _____

TIME: DATE: / /

```
Public class NumberConversionClass {
    Public String decToBinary (int num){
    return Integer.toBinaryString(num);
    }
Public String decToOct (int num){
    return Integer.toOctalString(num);
}
Public String decToHex(int num){
    return Integer.toHexString(num).to
        .UpperCase();
    }}
```

```
class BaseClass {
    void printResult() {
    System.out.Print("your answer:");
    }
}
```

IT-22023

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

```
// Public class SumClass extends BaseClass{
    double sum=0;
    Public double calculateSum(){
        for(double i=1; i>=0.1; i=i-0.1){
        sum=sum+i;
        return sum}
    }}
//
//
public class DivisorMulClass extends
        BaseClass{
    Public int GCD(int a, int b){
    while(b!=0){ int temp=b;
        b=a%b;
        a=temp;}
    return a;
    }
Public int LCM(int a, int b){
    return (a*b)/GCD(a,b);}}
//
```

17-2023

TOPIC NAME: _____ DAY: _____

TIME: DATE: / /

```
//public class customprintclass {
    Public string pro(string s){
        s="###"+s+"###";
    return s;
    }
}
//
```

14 IT-22023 (Re-add)

TOPIC NAME: _____ DAY: _____

TIME: DATE: / /

In Java BigInteger class is used to handle very large numbers that exceed the limit of primitive ~~byte~~ types like 'int', 'long'. It provides methods for Arithmetic operations, bit manipulation, GCD ...

```
// import java.util.Scanner;
   Public class mainClass{
 public static void main(String[] args)
{ Scanner sc=new Scanner(system.in);
    int n=sc.nextInt();
Baseclass bs=new Baseclass();
system.out.println("Factorial of "+n+".
    is =>."+bs.fact(n));  }}
```

TOPIC NAME : IT-22023 DAY : TIME : DATE : / /

```
import java.math.BigInteger;
Public class Baseclass{
 Public BigInteger fact(int n){
 BigInteger ff=BigInteger.ONE;
  for(int i=1; i<=n; i++){
     ff=ff.multiply(BigInteger.valueOf
                          (i));
    }
  return ff;
  }}
```

//

(16) IT-22023

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Polymorphism in Java is the ability of an object to take many forms. It allows a single interface to represent different types, typically achived through method overriding and method overloading. It closely related to dynamic method dispatch,

```
// class Animal {
        void makeSound () {
    System.out.println (" Animal make
            sound ");
                }}
class Dog entends Animal{
    @Override
        Void makeSound () {
    system . out . Println ("Dog barks");
}
 }
```

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

```
class eat entends Animal {
    @ovennide
    void makeSound(·) {
        System·out·Println("cat meows");
    }
}

Public class ·polymorphism{
    public stutic void main(String[]
                            args){
        Animal myAnimal1 =new Dog();
        Animal myAnimal2 = new Cat();

myAnimal1.makeSound ();
myAnimal2·makeSound();
    }
}
```

//

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Performance Impact and Trade-offs

Polymorphism introduces a slight performance overhead due to dynamic method dispatch. The trade-of is between flexibility and efficiency.

Pros: Enhance code reusability, maintainability, and scalability by allowing different behaviors via a common interface.

Cons: Increase method call overhead due to runtime lookup and prevents certain compiler optimization that static method calls allow.

(17) IT-22023

TOPIC NAME: _____ DAY: _____

TIME: DATE: / /

Both ArrayList and LinkedList implements the List interface in java, but they differ in internal implementation, Performance, and use cases.

1. Internal Structure

- ArrayList: uses a dynamic array. Elements are stored in contiguous memory locations.
- LinkedList: uses a doubly linked list. Each element contains data and references to the previous and next elements.

- Time complexities for Operations.

Operations	ArrayList	LinkedList
Access	O(1)	O(n)
Insertion at End	O(1)	O(1)
" " Beg.	O(n)	O(1)
" " Mid	O(n)	O(n)
Del. " End	O(1)	O(1)
Del " Beg	O(n)	O(1)

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

When to use which?

Use ArrayList when:

- First random access O(1) is required
- More reads than inserts/deletes, especially in the middle
- Memory efficiency is important

Use LinkedList when:

- Frequent insertions/deletions at the beginning or middle
- No need for fast random access.
- Memory overhead is not a concern.

Impact on Large Datasets-

- ArrayList performs better for large datasets with frequent Random access.

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

- LinkedList can handle large datasets better when there are frequent insertions/deletions at arbitrary positions but suffers from cache locality issues due to scattered memory allocation.

(40) IT-22023

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

```
import java.util.Scanner;
Public class mainclass {
    Public Static void main(String [] args) {
    Scanner sc = new Scanner (system.in);
     int  n = sc.nentInt ();
     int  y = sc.nent Int ();
     System.out.Println("SUM => " + (n+y));
     system.out.println("Diff => " + (n-y));
     System.out.Println("Product " + (n*y));
     if (y != 0)
        system.out.println("Quotiant =>
                                " + (n/y));
     else,
        System.out.Println("Invalid ");
    }
}
```

(29) IT-22023

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

```
import java.io.File;
import java.io.printWriter;
import java.util.Scanner;

Public class mainclass{

Public static void main(String[] args){
    Scanner sc = new Scanner (system.in);
    String input = "input.txt";
    string output = "Output.tnt";

Scanner read = new Scanner (new
                    File(input));

Printwriten write= new printwriten
                (new File(output));
read.UseDeliniten (",");
        int mani = Integer.MIN-VALUE;
    while(read.hasNentLine()){
        if(read.nentInt()>mani)
            mani = read.nentInt();
}
system.out.println ("mani");
    }}
```

(31) IT-22023

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

```
import java.util.Date;
import java.text.SimpleDateFormate;
Public Class MainClass {
    Public Static Void main(String []
                args) {
        Date now = new Date();
SimpleDateFormat f = new SimpleDate Formal
        ("yyyy-mm-dd HH:mm:ss");
    System.out.println(f.format(now));
}
```

IT-22023.

(32)

TOPIC NAME: _____ DAY: _____

TIME: DATE: / /

```
Public class counterclass {
    Private static int instanceCount=0;

    Public counterclass () {
        instanceCount++;
        if (instanceCount > 50)
            instanceCount=0;
    }
Public static int getInstanceCount() {
    return instanceCount;
}
Public static void main (String [] args) {
    for (int i=0 ; i<55; i++)
        new counterclass();
    System.Out.Println("Object " + (i+1) +
        ", Instance Count : " +
        counterclass.getInstanceCount());
}
}
```

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Encapsulation, Abstraction, polymorphism inheritance.

Encapsulation:

Encapsulation is the concept of bundling data and methods that operate on the data into a single unit and restricting direct access to them. It is achieved using private access modifiers and Providing public getter and setter method to control access.

```
// class student {
        Private string name;
Public string getName() {
        return name; }
Public void setName (String name) {
        this.name = name;
} ;                    //
```

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Abstraction:

Abstraction hides implementation details and only enposes necessary functionalities. It is achived using abstract classes and interfaces.

```
// abstract class vehicle{
    abstract void start();
    }

class car entends vehicle{
    @Override
    void start(){
System.out.println("cars starts
    with a key");

}
}
```

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Polymorphism

Polymorphism means "many forms" allowing a method to behave differently based on the object. It is categorized into compile time and runtime.

Complie time → method overloading

Run time → method overriding

// method overloading

```
class mathoperation {
int add (int a, int b) {return a+b;}
double add (double a, double b)
{return a+b;}.
```

//

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

```
// method overriding
    class Animal {
        void sound() {
        System.out.Println("Animals
            make sound");
        }
    }

class cat entends Animal {
    @override
    void sound() {
        System.out.Println("meows");
    }
    }
//
```

Here cat override the sound() method from Animal.

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Inheritance

Inheritance allows one class (child/subclass) to acquire properties and methods of another class (parent/superclass), promoting code reusability.

```
// class Parent{
    void show(){
System.out.println("This is
Parent class");
    }}
class child extends parent{
    void display(){
      System.out.println
("This is child class");
    }}
```

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Abstract class vs Interface

Both abstract class and interfaces are used for abstraction in Java, but they serve different purposes.

* Abstract class: An abstract class is a class that cannot be instantiated and can have both abstract and non-abstract methods.

key Features:

- can have both abstract and concrete methods.
- can have constructors.
- can have instance variables
- can have access modifiers.

```
// abstract class vehicle {
        String brand = "Toyota";

    abstract void start(); // abstract method.
```

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

```
void stop() {  // concrete method
    System.out.println("Vehicle
        stopped");
}}
```

```
class Car extends Vehicle {
    @Override
    void start() {
    Sout("Car starts with a
        key");
}
}
```

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Interface

An interface is a fully abstract type that only contains method signatures

- Can only have abstract methods.

```
// interface Animal {
        void makeSound();
    }
class Dog implements Animal {
    @Override
    public void makeSound() {
    System.out.println("Dog barks");
    }
}.
```

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Multithreading in Java.

Multithreading allows multiple parts of a program to run concurrently, improving efficiently.

Multi theading is useful when we need to perform multiple tasks at the same time, to improve speed and efficiency. It especially helpful when dealing with:

- cpu Intensive Tasks
- I/O operations
- Real time Applications.

d Single-core CPU: JVM switches between threads very fast, giving illiution of parallel execution.

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Multicore-cpu: Threads actually run in parallel on different cpu core.

multithreading can be done in two ways.

① Extending thread class

② Implement Runnable Interface.

```
//class MyThread extends Thread {
    @override
    public void run() {
    for loop (1-5)
    }
}
public class Thread Example {
    public class void main(String[]
    args) {
My thread t1 = new MyThread();
    t1.start(); } } //
```

TOPIC NAME : DAY :
TIME : DATE : / /

Implement the Runnable Interface

```
// class MyRunnable implements
                Runnable {
@Override
  public void run() {
    for (int i = 1; i <= 5; i++)
    System.out.println("Runnable " + i);
  }
}

public class RunnableExample {
      psvm()
    MyRunnable task = new MyRunnable();
    Thread t1 = new Thread(Task);
    t.start();
```

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Synchronization

Synchronization is a technique through which we can control multiple threads among the number of threads, only one thread will enter inside the synchronized area..

The main purpose of synchronization is to overcome the problem of multithreading when multiple thread are trying to access same resource at a time.

Synchronization is broadly classified in to two catagory.

→ method level synchronization.
→ block level synchronization.

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

```
Public synchronised void run() {
    // resource
    }
t1.start();
t2.start();
t3.start();
```

Deadlock: Deadlock is a situation in multithreading where two or more threads are blocked forever, waiting for each other to release a resource. This typically occurs when two or more threads have circular dependencies on a set of locks.

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

A deadlock in Java occurs when two or more threads are waiting for each other to release a resource, but neither can proceed because they are blocked indefinitely.

Example:

. Thread 1 lock Resource A and wait for Resource B.

Exception Handling in Java

E.H.I.J allows a program to handle runtime errors gracefully instead of crashing. It is done using try, catch, finally, throw, throws.

Types of Exceptions:

① checked Exceptions.

② Unchecked Exceptions

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

- IO Exception:
- SQL Exception
- NullPointerException
- Arithemitic Exception
- out of Memory Error
- stack Over Flow.

```
// try {
      int result = 10/0;
      System.out.println(result);
   }
  catch (ArithemicException e){
      System.out.println
         ("Cannot divide by 0");
   }
//
```

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Java Garbage Collection Mechanism.

GC in Java automatically removes unused objects from memory, preventing memory leaks. The JVM handles this process.

- When an object is no longer reachable, it becomes eligible for GC.
- The GC process reclaims memory by destroying these objects.
- Java does not guarantee immediate garbage collection, only when needed.