Q1. In Java, interfaces can have default method since Java 8. How does this change the way interfaces are used compared to abstract class. Can an abstract class implement an interface, and if so how does this effect inheritance and method resolution when both the abstract class and interface provide conflicting method implementation?

Ans:

Default method interface:

- Interface can now have default method
- It make interface more powerful because they can share behavior with classes like abstract class

Abstract class can implement an interface:

- Yes abstract class can implement an interface
- If the interface has methods (default or abstract) the abstract class can
  * Provide its own implementation for the method
  * leave the method for its subclass to implement

What happen with conflict:

If both abstract class and interface provide method with the same name

* The abstract class method takes the priority
* This is because method in the class hierarchy override interface method

Code:

```
interface MyInterface {
    default void show() {
    System.out.println("Interface default
     method.");          }
                    }
abstract class Myabstractclass implement Myinterf
    { public void show() {
     System.out.println(" Abstract class method
     }
}
class Myclass extends MyAbstract class {}
 Public class Main {
    Public static Main {
  Public static void main (String[] args) {
     My class.obj = new Myclass ();
      obj.Show (); }
```

22. difference between Hashmap Treemap Linked Hashmap . In terdata structures time complexities common operation . Which you would prefer?

Hashmap:

Structure: uses a has table to store key-value pair
Order : No order (completedly random)
Time : fast for insert, lookup, delet (O(1))
Usage : use when you need space and dont care about the order.

Treemap:

Structure: uses a tree (Red-Black) to store key value
Order : sorted by key (smallest to largest)
Time : slower than Hashmap (O(log n))
Usage : When you need sorted key

linked Hashmap:

Structure: uses a hash-table + linked list to store
order : keep the insertion or access order
time : same speed as Hash map (O1)
usage : use when you need to main tain the order of element

Difference in Order:

Hashmap : No order

Treemap : Sorted order

Linked Hashmap: Insertion order or access order

Which to use:

Hashmap : When speed matter

Treemap : When you need sorted key

Linked Hashmap: When you need preserved order of element

23. In Java Static building occur at compile time and dynamic building occur at runtime. How does this relate to polymorphism. Provide example where static building and dynamic building might produce different behavior. Discuss how this effect performance and method resolution and method resolution in heritance hierarchies.

Performace and Method resolution in inheritance Hierarchies:

Static building:

Performace: faster because it resolved at compile time

Method resolution: The method is resolved based on the reference type

Dynamic building:

Performance: Slightly slower because it resolved at runtime

Method resolution: The method is resolved based on the actual object

In inheritance hierarchies dynamic bilding is used for overriden method allowing runtime polymorphism, where the same method can exhibit different behaviors depending on the actual class of the object

Static Binding vs Dynamic Binding in simple term

Static Binding

- Happen at compile time
- used for: Private, final and static method
- Method resolution: The method that will be called is determined by the reference type
- Performance: faster

Dynamic Binding

- Happen at runtime
- Used for: Overridden method in subclass
- Method Resolution: The method that will be called is determined by the actual objet type at runtime

Performance: Slightly slower

Polymorphism and Binding

Polymorphism: A single method can have different behaviors based on the object calling it

Static Binding: Fixed method call, determined by the reference type

Dynamic Binding: Method call depends on the actual object determined at runtime

Code:

```
class Animal {
  Public static void makesound () {
    System.out.Println("Animal sound");
  }
      Public void sleep () {
        System.out.Println ("Animal sleeps");
  }
}
Class dog extends Animal {
    Public static void makesound () {
      System.out.Println("Dog barks");
    }
    Public void sleep() {
      System.out.Println ("Dog sleeps");
} }
Public class TestBinding {
  Public static void main (String[] args)
    Animal a = new Dog();
        a.makesound ();
        a.sleep ();
} }
```

Animal sound
Dog sleeps

24. Advantage of Execution Service framework manally managing thread in Java. How does the submit() method differ from execute() in execution service, and what are the the potential benefit of using callable object instead of Runnable. Provide example

Why we Execution Service instead of Threads?

- Easier to Manage: Execution service thread for you, so you dont have to create or image them manually
- Efficiency: Execution service reuses thread which is faster than creating new ones every time
- Cleanen Shutdown: It automatically handles stopping threads properly after the tasks finish

Submit() Vs execute() in Execution Service

- execute();

* Run a Runnable task
* use when you dont need any result

submit():

* Runs a Runnable or callable task
* Returns a future object that can give a result later
* Use when you need a result or want to check if the task finished

Runnable vs Callable

Runnable

* No result no exception handeling
* Good for simple task

Callable

* Can return a result
* Can throw exceptions

code:

```
import java.util.concurrent.*;
public class Example {
  Public static void main (String [] args) {
  Execution Service execution = Execution.
    newfixed Thread Pool (2);
  Runnable task1 = () -> system.out.Println(
  "Task 1");
  Callable <String> task2 = () -> "Task 2 done";
  execution.execute (task1)
 future <strings> result = execution.submit(
    task 2);
 try {
   system.out.Print ln (result.get());
  }
  Catch (Exception e) {
    e.Print Stack Trace();
  }
  execution.shutdown();
}
}
```

25 What are the issue need to be considered during handaling the exception. Write a Program to calcalate area of a Circle with set radius method that throws an exception if the radius is negetive

Handaling Exception:

We want to calcalate the area of circle but the radius should not negetive. we will throw an exeception

Simple Program to calculate the Area

```
class Circle {
    private double radius;
    public void setRadius (Double radius) {
        if (radius<0) {
            throw new IllegalArgumentException
            (" Radius cant be negetive");
        } this.radius = radius;
    }
public class CircleArea{
    public static void main (String[] args) {
        Circle circle = new. circle();
```

```
try {
    circle.setRadius(-5);
    System.out.println("Area of circle: " + circle
        .getArea());
    }
    catch (IllegalArgumentException e) {
        System.out.println("Error" + e.getMessage());
    }
}
}
```

Explanation:

- setRadius(): check if the radius is negetive
- getArea(); calculate the Area of the circle using $\pi \times r^2$
- In the main method, we tryto set anegetive radius (-5), which cause the exception

Output:

Error: Radius cant be negetive

26 How many ways we have to create a Thread write two single program create a simple Thread using Runnable Interface and extending Thread class.

In Java there are two main way to create a thread

1. By implementing the runnable interface
2. By extending the Thread class

1. Using the Runnable Interface

```
class MyRunnable implements Runnable{
    Public void run(){
        System.out.Println("Thread running using
        Runnable interface");
    }
}

Public class RunnableExample{
Public static void main(String[] arg)
{ My Runnable myRunnable = new MyRunnable();
    Thread thread = new Thread (myRunnable);
    thread.start();
}
}
```

Using the Thread class

```
class MyThread extends Thread {
    Public void run() {
    System.out.println("Thread running Thread class");
}
}
Public class Thread Example {
    Public static void main (String[] args) {
        MyThread thread = new MyThread();
        thread.start();
}
}
```

Key Difference:

Runnable Interface: when you want to achive multiple inheritance, as it avoid the restriction of extending only one class

Thread class: Simpler to implement but limits extending classes because Java supports single inheritance

29. Write a Java Program to read the highest value of a series from a file and writting the sum into another file. Use Scanner and PrintWriter class.

Code:

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

Public class File Processing {

Public static void main (String [] args){
try {
Scanner Scanner = new Scanner (new File ("
input.txt"));
    int highest = Integer.MIN_VALUE;
    int sum=0;

    while (Scanner.hasNextInt());
      sum += num;
     If (num>highest) {
        highest =num;
}
}
    Scanner.close();

  PrintWriter written = new PrintWritten (
  "Output.txt");
```

```
writer.println("Highest Value: " + highest);
    writer.println("sum" + sum);
    writer.close();
  System.out.println("Done! results are in '
    output.txt' ");
  } catch (Exception e) {
    System.out.println("Error" + e.getMessage());
  }
 }
}
```

Explanation

Input File (input.txt)

Contain the numbers (5,10,15,20)

logic

- The program reads each each numbers using Scanner.
- Calculates the sum and tracks the highest value

Output:

Highest value : 20
Sum : 50

30 write a java code of an array size(n>20) and another array of size n/10 then determine the divison and reminden after diving the first array by the second array.

```
import java.util.Scanner;
Public class Array Division{
Public static void main(String[] args)
{ Scanner scanner= new scanner (system.in);
System.out.print("Enter the size of the
first array (n>20);");
   int n = scanner.nextInt();
if (n<=20)
{ System.out.println("n must be greater
   than 20!");
return;
}
int [] array1 = net int[n]
System.out.println("Enter " +nt" element
   of the first array:");
     array[i] = scanner.nextInt();
}
int m= n/10;
int [] array2= next int[m];
System.out.println("Enter" +m"+"
   element for the second array");
```

```
For (int i=0; i<n i++)
{ array2[i] = scanner.nextInt();
if (array2[i] == 0){
System.out.println (" Division by zero is
not allowed !");
return;
}
}

System.out.println ("Results (ceiling Division):");
for (int i=0 ; i<m; i++)
{ int divident = array1[i];
int divison = array2[i];
int quotient =(int) Math.ceil ((double)
dividend / divison);
int remainden = dividend % divison;
System.out.println ("Array[" +i+ "] / Array2
["+i+"] = " + quotient +" "(Remainden: +
remainden +")" );
}
scanner.close();
}
```

Input 30

Input: 10, 20, 30 300

Input 5, 10, 15

output: 2
2
2