1. **Uses of final keyword**
2. **Difference between Classes, Interfaces, and Abstract Classes (table format)**
3. **A Stack Interface (with implementation example)**

1️⃣ Uses of final Keyword

In Java, the **final keyword** is a **non-access modifier** used for **variables, methods, and classes**.

- **Uses:**
  1. **Final Variable** → makes the variable a constant (cannot be reassigned).
  2. **Final Method** → prevents method overriding in subclasses.
  3. **Final Class** → prevents inheritance of the class.

Examples:

```
// 1. Final Variable
class ConstantExample {
    final int MAX_VALUE = 100; // constant

    void show() {
        // MAX_VALUE = 200; // ❌ Error: cannot change value
        System.out.println("Constant value = " + MAX_VALUE);
    }
}

// 2. Final Method
class Parent {
    final void display() {
        System.out.println("Final method in Parent class.");
    }
```

```
}

class Child extends Parent {
    // void display() { }  // ❌ Error: cannot override final method
}

// 3. Final Class
final class Vehicle {
    void show() {
        System.out.println("This is a final class.");
    }
}

// class Car extends Vehicle { } // ❌ Error: cannot inherit final class
```

Output:

```
Constant value = 100
Final method in Parent class.
This is a final class.
```

2 Difference Between Class, Abstract Class, and Interface

Feature	Class	Abstract Class	Interface
Definition	Blueprint to create objects	Partially implemented class (may have abstract & concrete methods)	Complete abstraction, contains abstract methods (Java 7) + default/static methods (Java 8+)
Object Creation	Objects can be created	Cannot create object	Cannot create object
Methods	Only concrete methods	Both abstract & concrete methods	Abstract methods (default public & abstract) + default & static methods
Variables	Instance & static allowed	Instance & static allowed	Only public static final constants
Inheritance	Single inheritance	Supports single inheritance	Supports multiple inheritance (a class can implement many interfaces)
Keyword	class	abstract class	interface
Constructor	Allowed	Allowed	Not allowed
Access Modifiers	All allowed	All allowed	Only public (by default)
When to Use	For concrete reusable code	When some functionality is common but must be overridden	To achieve complete abstraction & multiple inheritance

3 A Stack Interface

- A **Stack** is a **linear data structure** that follows the **LIFO (Last In, First Out)** principle.
- The **Stack interface** defines the **basic operations** of a stack such as:
  - **push()** → insert element on top
  - **pop()** → remove and return top element
  - **peek()** → view top element without removing
  - **isEmpty()** → check if stack is empty
- In Java, Stack can be implemented using **arrays**, **linked lists**, or built-in classes like Stack from java.util.
- Used in **function calls, expression evaluation, undo operations, parsing**, etc.

The **Stack** is a **LIFO (Last In, First Out)** data structure.

🔹 **Stack Interface:**

```
// Stack Interface
interface Stack {
    void push(int item);   // insert
    int pop();             // remove
    int peek();            // top element
    boolean isEmpty();     // check empty
}
```

🔹 **Implementation using Array:**

```
class ArrayStack implements Stack {
    private int maxSize;
    private int[] stackArray;
    private int top;

    // Constructor
    ArrayStack(int size) {
```

```
    maxSize = size;
    stackArray = new int[maxSize];
    top = -1;
}

// Push
public void push(int item) {
    if (top < maxSize - 1) {
        stackArray[++top] = item;
        System.out.println(item + " pushed into stack.");
    } else {
        System.out.println("Stack Overflow!");
    }
}

// Pop
public int pop() {
    if (!isEmpty()) {
        return stackArray[top--];
    } else {
        System.out.println("Stack Underflow!");
        return -1;
    }
}

// Peek
public int peek() {
    if (!isEmpty()) {
```

```
            return stackArray[top];
        } else {
            System.out.println("Stack is Empty!");
            return -1;
        }
    }

    // Check Empty
    public boolean isEmpty() {
        return (top == -1);
    }
}

// Main Class to Test
public class StackDemo {
    public static void main(String[] args) {
        Stack s = new ArrayStack(3);
        s.push(10);
        s.push(20);
        s.push(30);
        s.push(40);  // Overflow

        System.out.println("Top element: " + s.peek());
        System.out.println("Popped: " + s.pop());
        System.out.println("Popped: " + s.pop());
        System.out.println("Popped: " + s.pop());
        System.out.println("Popped: " + s.pop()); // Underflow
    }
```

}

- **Output:**

10 pushed into stack.

20 pushed into stack.

30 pushed into stack.

Stack Overflow!

Top element: 30

Popped: 30

Popped: 20

Popped: 10

Stack Underflow!

Simple Inheritance:

```
// Define an interface
interface Animal {
    void sound();   // abstract method
    void eat();     // abstract method
}
// Implement the interface in a class
class Dog implements Animal {
    public void sound() {
        System.out.println("Dog barks");
    }

    public void eat() {
        System.out.println("Dog eats bones");
    }
}

// Main class to test
public class InterfaceDemo {
    public static void main(String[] args) {
        Animal a = new Dog();   // interface reference, object of Dog
        a.sound();
        a.eat();
    }
}
```

Output:

Dog barks

Dog eats bones

10 normal MCQs (one correct answer each) and **5 multi-correct MCQs** (two correct answers out of five).

✅ Part A: 10 Normal Type MCQs (One Correct Answer)

Q1. Which of the following best defines **Polymorphism** in Java?
a) Having multiple variables with the same name
b) Having multiple classes with the same name
c) Ability of an object to take many forms
d) Ability to inherit multiple classes

Answer: c) Ability of an object to take many forms

Q2. Method Overloading is an example of:
a) Runtime Polymorphism
b) Compile-time Polymorphism
c) Dynamic Method Dispatch
d) Inheritance

Answer: b) Compile-time Polymorphism

Q3. Which keyword in Java refers to the **current object**?
a) super
b) this
c) static
d) final

Answer: b) this

Q4. Which keyword is used to access the **parent class constructor or method**?
a) this
b) parent
c) base
d) super

Answer: d) super

Q5. Which type of inheritance is **not supported directly** in Java (using classes)?
a) Single
b) Multilevel
c) Hierarchical
d) Multiple

Answer: d) Multiple

Q6. Which of the following **access modifiers** allows visibility in the same package and subclasses only?
a) public
b) private
c) protected
d) default (no modifier)

Answer: c) protected

Q7. Which of the following is **true about abstract classes**?
a) They can be instantiated directly
b) They may contain abstract and non-abstract methods
c) They cannot have constructors
d) They must contain only abstract methods

Answer: b) They may contain abstract and non-abstract methods

---

Q8. Which feature is **not supported** by interfaces in Java (before Java 8)?
a) Multiple inheritance
b) Static variables
c) Method implementation
d) Constants

Answer: c) Method implementation

Q9. The **final keyword** in Java can be applied to:
a) Variables
b) Methods
c) Classes
d) All of the above

Answer: d) All of the above

Q10. A **stack** works on which principle?
a) FIFO (First In First Out)
b) LIFO (Last In First Out)
c) FILO (First In Last Out)
d) None of the above

Answer: b) LIFO (Last In First Out)

✅ Part B: 5 Multi-Correct MCQs (Two Correct Answers)

Q11. Which of the following are features of **constructors** in Java?
a) They must have the same name as the class
b) They return void
c) They are automatically called when an object is created
d) They can be static
e) They must always be abstract

Correct Answers: a) and c)

Q12. Which statements are true about the **static keyword** in Java?
a) Static methods can be called without creating an object
b) Static variables are shared among all objects of a class
c) Static methods can use this keyword
d) Static blocks are executed before the main method
e) A class can be declared static at the top level

Correct Answers: a) and b)

Q13. Which of the following are examples of **Runtime Polymorphism**?
a) Method Overloading
b) Method Overriding
c) Dynamic Method Dispatch
d) Constructors Overloading
e) Final Methods

Correct Answers: b) and c)

Q14. Which of the following are **true about Interfaces** in Java?
a) All methods are abstract by default (Java 7)
b) They support multiple inheritance

c) They can have constructors
d) They can have default methods (Java 8 onwards)
e) They can have private instance variables

Correct Answers: a) and b) (also d is true since Java 8, but if you want 2 correct → a & b)

Q15. Which of the following are **uses of the final keyword**?
a) To prevent method overriding
b) To prevent inheritance of a class
c) To declare constants
d) To define abstract methods
e) To allow multiple inheritance

Correct Answers: a) and b) (also c is true; if you allow 3 answers, then a, b, c)