1. Explain the importance of inheritance in Java?

Point	**Simple Explanation**
Code Reusability	Write common code once in the parent class and use it in all child classes.
Method Overriding	Child class can change how a parent method works.
Easy to Maintain	If you update the parent class, the changes apply to all child classes too.
Polymorphism	Same method name can do different things in different classes.
Class Hierarchy	Helps group similar classes (like Animal → Dog, Cat, etc.) for better structure.

2. What is Super keyword in Java? Illustrate with an example

The super keyword is used in Java to access members (variables, methods, or constructors) of a parent class from a child class.

super() → calls the parent class constructor
super.variable → accesses a variable from the parent class
super.method() → calls a method from the parent class
Case1:

```
// Superclass A
class A {
    A() {
        System.out.println("A class Constructor");
    }
}

// Subclass B extending A
class B extends A {
    B() {
        super(); // Calls A's constructor
        System.out.println("B class Constructor");
    }
}

// Main class
class Test {
```

```
    public static void main(String[] args) {
        B b = new B();
    }
}
```

Ouput:

```
A class Constructor
B class Constructor
```

Case2:

```
// Superclass A
class A {
    int i;
}

// Subclass B extending A
class B extends A {
    int i; // This hides A's 'i'

    B(int a, int b) {
        super.i = a; // Access A's i
        i = b;       // This class's i
    }

    void show() {
        System.out.println("i in superclass: " + super.i);
        System.out.println("i in subclass: " + i);
    }
}

// Main class
class UseSuper {
    public static void main(String[] args) {
        B subOb = new B(1, 2);
        subOb.show();
    }
}
```

Output::

```
i in superclass: 1
i in subclass: 2
```

14. Demonstrate multi-level hierarchy of classes in java
 3. Create a multi-level hierarchy for inheritance in Java

```
// Parent class A
class A {
    void funcA() {
        System.out.println("This is class A");
    }
}

// B is child of A
class B extends A {
    void funcB() {
        System.out.println("This is class B");
    }
}

// C is child of B (and grandchild of A)
class C extends B {
    void funcC() {
        System.out.println("This is class C");
    }
}

// Main class
public class Demo {
    public static void main(String[] args) {
        C obj = new C(); // Object of lowest class in the hierarchy

        obj.funcA(); // Inherited from A
        obj.funcB(); // Inherited from B
        obj.funcC(); // Defined in C
    }
}
```

Output:

```
This is class A
This is class B
This is class C
```

When a class inherits from a class which itself inherits from another class, it forms a chain — this is called multilevel inheritance.

Example Chain:
A → B → C

4. Explain the working of constructors for inheritance in Java

```
class A {
    A() {
        System.out.println("Constructor of A");
    }
}

class B extends A {
    B() {
        System.out.println("Constructor of B");
    }
}

class C extends B {
    C() {
        System.out.println("Constructor of C");
    }
}

public class Demo {
    public static void main(String[] args) {
        C obj = new C(); // Creating object of subclass
    }
}
```

Ouput::

```
Constructor of A
Constructor of B
Constructor of C
```

12. Write a Java program to demonstrate dynamic binding using method overriding

10. Write a Java Program to demonstrate why we need method overriding

11. Write a Java program to illustrate method overriding in Java

5. Demonstrate method overriding in java

Method Overriding is when a subclass provides a specific implementation of a method that is already defined in its superclass.
Method name, return type, and parameters must be the same.
It allows runtime polymorphism in Java.

```
class A {
    void callme() {
        System.out.println("Inside A's callme method");
```

```
    }
}

class B extends A {
    void callme() {
        System.out.println("Inside B's callme method");
    }
}

class C extends A {
    void callme() {
        System.out.println("Inside C's callme method");
    }
}

public class Dispatch {
    public static void main(String[] args) {
        A a = new A();
        B b = new B();
        C c = new C();
        A r; // reference of type A

        r = a; r.callme(); // Calls A's version
        r = b; r.callme(); // Calls B's version
        r = c; r.callme(); // Calls C's version
    }
}

Ouptut::

Inside A's callme method
Inside B's callme method
Inside C's callme method


 6. What are the key features of dynamic method dispatch

class A {
    void callme() {
        System.out.println("Inside A's callme method");
    }
}

class B extends A {
    void callme() {
```

```
        System.out.println("Inside B's callme method");
    }
}

class C extends A {
    void callme() {
        System.out.println("Inside C's callme method");
    }
}

public class Dispatch {
    public static void main(String[] args) {
        A a = new A();
        B b = new B();
        C c = new C();
        A r; // reference of type A

        r = a; r.callme(); // Calls A's version
        r = b; r.callme(); // Calls B's version
        r = c; r.callme(); // Calls C's version
    }
}

Ouptut::

Inside A's callme method
Inside B's callme method
Inside C's callme method

Java chooses which version of a method to run at runtime, based on the object,
not the reference

In java terms::

A r;
r = new B();  // B has its own version of the method
r.callme();   // Java runs B's method, not A's

r = new C();  // Now it points to C
r.callme();   // Java runs C's method, not A's

| **Feature**                | **Simple Explanation**
                            |
| ————————————————————— |
————————————————————————————————————————————————————————————————————— |
```

Run-time Polymorphism	Java decides which method to run **while the program is running**.
Superclass Reference	A parent class variable can hold a child class object.
Overridden Method Call	The method in the **child class** runs, not the one in the parent.
Needs Inheritance	Works only when a **child class overrides** a method from the parent class.
Flexible Code	You can write code that works for **many types of objects**.

7. Write a short note on abstract class in Java

Sometimes, all the data of one class are not required to be inherited to the sub class. In such cases, abstract is used with the class.

```
// A simple demonstration of abstract class
abstract class A {
    abstract void callme(); // abstract method

    void callmetoo() {
        System.out.println("This is a concrete method.");
    }
}

class B extends A {
    void callme() {
        System.out.println("B's implementation of callme.");
    }
}

public class Demo {
    public static void main(String[] args) {
        B b = new B();
        b.callme();      // calls overridden method
        b.callmetoo();   // calls concrete method from abstract class
    }
}
```

Output::

```
B's implementation of callme.
This is a concrete method.
```

Abstract = I'll do it later.

8. Identify the key features of object class in Java

Method	**What it Does (Simple Explanation)**
`Object.clone()`	Makes a copy of an object.
`Object.equals(Object obj)`	Checks if two objects are equal.
`Object.finalize()`	Called before the object is deleted by the garbage collector.
`Object.getClass()`	Tells the class type of the object at runtime.
`Object.hashCode()`	Returns a unique number (ID) for the object.
`Object.notify()`	Wakes up one thread waiting on the object.
`Object.notifyAll()`	Wakes up all threads waiting on the object.
`Object.toString()`	Returns a string that describes the object.
`Object.wait()`	Makes a thread wait until notified.
`Object.wait(long ms)`	Waits for a specific time before continuing.
`Object.wait(long ms, int ns)`	Waits for a more precise time (milliseconds and nanoseconds).

16. What are the key differences between multilevel inheritance and single-level inheritance

Aspect	**Single-Level Inheritance**	**Multilevel Inheritance**
Definition	A class inherits directly from one parent class.	A class inherits from a class, which itself is a child of another class.
Hierarchy Depth	Only one level of inheritance.	Involves more than one level of inheritance.
Structure	`Class A → Class B`	`Class A → Class B → Class C`
Complexity	Simple and easier to understand.	More complex due to multiple levels.

| **Example** | `class B extends A {}` | `class C extends B {}, class B extends A {}` |
| **Use Case** | When only one layer of base functionality is needed. | When functionality builds in steps across multiple levels. |

18. Write a Java program to show the method hiding in Java

Definition: In Java, if a static method is defined in both a superclass and subclass with the same name and signature, the method in the subclass hides the one in the superclass (this is not overriding because static methods are resolved at compile time).

```
class A {
    static void display() {
        System.out.println("Static method in Class A");
    }
}

class B extends A {
    static void display() {
        System.out.println("Static method in Class B");
    }
}

public class Test {
    public static void main(String[] args) {
        A obj = new B();  // Reference is of A
        obj.display();    // Calls A's display (method hiding, not overriding)
    }
}
```

Output:
Static method in Class A

19. Write a Java program to show the variable hiding in Java

Definition: If a subclass declares a variable with the same name as a variable in the superclass, the variable in the subclass hides the variable in the superclass

```
class A {
    int num = 10;
}

class B extends A {
```

```
    int num = 20;  // hides A's num
}

public class Test {
    public static void main(String[] args) {
        B obj = new B();
        System.out.println("B's num: " + obj.num);        // prints 20
        System.out.println("A's num: " + ((A)obj).num);   // prints 10
    }
}

Output:
B's num: 20
A's num: 10
```

21. Explain the advantages and potential drawbacks of using multilevel inheritance in Java.

advantages::

Feature	Simple Explanation
Code Reusability	Code written in a base class can be reused by all its subclasses.
Organized Structure	Maintains a logical class hierarchy (A → B → C).
Easy to Extend	You can easily add new features by adding a subclass.
Method Overriding	Allows subclasses to customize or update parent class behavior.

drawbacks::

Issue	Simple Explanation
Complexity	Too many levels of inheritance can make the code hard to understand.
Tight Coupling	A small change in the base class might affect all its child classes.
Hard to Debug	It can be difficult to trace errors in deep

inheritance chains. |
| No Multiple Inheritance | Java does not allow inheriting from multiple classes directly. |

22. Explain the three modes of inheritances

Mode of Inheritance	**Description**	**Example**
Single Inheritance	A subclass inherits from a single superclass.	`class B extends A {}`
Multilevel Inheritance	A class inherits from a class, which itself inherits from another class.	`class C extends B`, `class B extends A`
Hierarchical Inheritance	Multiple classes inherit from the same parent class.	`class B extends A`, `class C extends A`

25. Show how run time polymorphism is performed in Java using dynamic method dispatch
Runtime Polymorphism means that the method to be executed is determined at runtime, not at compile time.

In your code:
The reference r is of type A (superclass).
At runtime, r points to an object of class A, B, or C.
Based on the actual object, the correct overridden method is called.

26. Create a class called employee, derive a new class called programmer from employee. Comment on the order in which constructors are called when an object of employee is created

```
class Employee {
    float salary = 40000;

    // Constructor of Employee
    Employee() {
        System.out.println("Employee constructor called");
    }
}

class Programmer extends Employee {
    int bonus = 10000;
```

```
    // Constructor of Programmer
    Programmer() {
        System.out.println("Programmer constructor called");
    }

    public static void main(String[] args) {
        Programmer p = new Programmer(); // Object created
        System.out.println("Salary: " + p.salary);
        System.out.println("Bonus: " + p.bonus);
    }
}

Output::
Employee constructor called
Programmer constructor called
Salary: 40000.0
Bonus: 10000
```

23. What are the implications of using a final method in terms of inheritance and method overriding?
9. What is Final keyword in Java with respect to inheritance? Illustrate with an example
27. Describe a scenario where you would use final keyword in a multilevel inheritance structure

In Java, the final keyword is used to restrict inheritance or modification:
final class → cannot be extended.
final method → cannot be overridden.
final variable → value cannot be changed (constant).

```
class A {
    final void show() {
        System.out.println("This is a final method from class A");
    }
}

class B extends A {
    // Cannot override show() here because it is final in A
    // void show() {
    //     System.out.println("Trying to override in B"); // ❌ Error
    // }
}
```

```
class C extends B {
    // Still can't override show() here
    // void show() {
    //     System.out.println("Trying to override in C"); // ❌ Error
    // }
}

public class FinalExample {
    public static void main(String[] args) {
        C obj = new C();
        obj.show(); // Calls show() from class A
    }
}
```

🔒 1. Prevents Overriding

A subclass cannot provide its own version of a final method.

Helps preserve the original behavior of the method defined in the superclass.