1. What is encapsulation in Java?
2. Binding data and functions together
3. Hiding implementation details
4. Both a and b
5. Only a
6. Which access modifier provides the **least** encapsulation?
7. public
8. private
9. protected
10. default
11. Which is true about encapsulation in Java?
12. It breaks OOP rules
13. It enables implementation hiding
14. It disables polymorphism
15. None of the above
16. What keyword is commonly used to encapsulate fields?
17. static
18. final
19. private
20. abstract
21. Why do we use getters and setters?
22. To break encapsulation
23. To allow direct field access
24. To control access to variables
25. To overload constructors
26. Encapsulation ensures:
27. Loose coupling
28. Strong coupling
29. No inheritance
30. Memory overuse
31. Which of the following best supports encapsulation?
32. abstract classes
33. public fields
34. private fields with public methods
35. final methods
36. What happens if you make a class’s fields public?
37. Increased encapsulation
38. Better security
39. Reduced encapsulation
40. No effect
41. Choose the correct encapsulation practice:
42. Declaring all fields public
43. Having both private fields and public setters
44. Avoiding constructors
45. Using only static methods
46. What is data hiding in the context of encapsulation?
47. Deleting data
48. Hiding internal variables from other classes
49. Encrypting fields
50. Making methods final
51. Which is the disadvantage of **no encapsulation**?
52. More flexibility
53. Unpredictable behavior
54. Increased performance
55. Better readability
56. What does this class violate?

public class User {

public String name;

public int age;

}

1. Polymorphism
2. Encapsulation
3. Abstraction
4. None
5. How can you modify the above class to enforce encapsulation?
6. Make fields protected
7. Make fields private and provide public getters/setters
8. Use static methods
9. Add final keyword
10. Which of these is **NOT** a benefit of encapsulation?
11. Security
12. Better maintainability
13. Data hiding
14. Slower performance
15. Encapsulation is implemented at:
16. Class level
17. Object level
18. Package level
19. None
20. A setter method typically:
21. Declares the class
22. Returns a value
23. Sets a value to a field
24. Encrypts a variable
25. Choose the best field declaration for encapsulation:
26. public int count
27. private int count
28. protected int count
29. int count
30. Why might you choose **not** to provide a setter?
31. To hide implementation logic
32. To allow only read access
33. To make the field final
34. To implement polymorphism
35. Which type of method **typically breaks** encapsulation?
36. Static methods
37. Public methods
38. Methods that return field references directly
39. Final methods
40. Can encapsulation be violated using reflection?
41. Yes
42. No
43. Only in abstract classes
44. Only for public fields
45. What Java keyword is critical to supporting encapsulation?
46. private
47. static
48. extends
49. implements
50. Which of these supports **read-only** encapsulated fields?
51. public void setName(String name)
52. public String getName()
53. private String name
54. Constructor injection
55. What’s the biggest risk of exposing class internals?
56. Faster execution
57. Better design
58. Violation of class contracts
59. Improved security
60. Why might encapsulation be considered a **security feature**?
61. It encrypts data
62. It uses final methods
63. It hides sensitive internal states from external modification
64. It uses abstract methods
65. When are fields usually set in an encapsulated class?
66. At runtime through public methods
67. At compile time
68. Using public fields
69. Automatically by JVM
70. Which keyword is used to inherit a class in Java?
71. implements
72. extends
73. inherits
74. superclass
75. Which concept is best implemented using inheritance?
76. Abstraction
77. Code reusability
78. Polymorphism
79. Encapsulation
80. What does the super keyword refer to?
81. Current class object
82. Child class
83. Parent class constructor or method
84. Static method
85. Can a subclass override a private method of its superclass?
86. Yes
87. No
88. Only if it’s final
89. Only if it’s abstract
90. If class B extends class A, which is true?
91. A is subclass, B is superclass
92. A is superclass, B is subclass
93. Both are interfaces
94. It has no impact
95. What is the default superclass for all classes in Java?
96. Object
97. String
98. Class
99. Base
100. Which of these is not inherited by subclasses?
101. public fields
102. static methods
103. private methods
104. protected methods
105. Can a class inherit from multiple classes in Java?
106. Yes
107. No
108. Only with interface
109. Only if all classes are abstract
110. Which keyword prevents a class from being inherited?
111. abstract
112. protected
113. static
114. final
115. What happens when a subclass declares a method with the same signature as a parent class?
116. It's ignored
117. It hides the parent’s method
118. It overrides the method
119. It throws compile error
120. Inheritance promotes:
121. Strong coupling
122. Code duplication
123. Code reuse
124. Compilation delay
125. Which inheritance type is **not** supported directly by Java?
126. Single
127. Multilevel
128. Multiple
129. Hierarchical
130. What is hierarchical inheritance?
131. One class inheriting multiple classes
132. Multiple classes inheriting from the same superclass
133. Class extending itself
134. Inheritance via interface only
135. Which method is used to call a superclass constructor?
136. this()
137. parent()
138. super()
139. init()
140. What is the consequence of improper use of inheritance?
141. Code becomes modular
142. Code becomes tightly coupled and fragile
143. Better readability
144. Faster runtime
145. Which of these is legal in a constructor of a subclass?
146. super()
147. super(arguments...)
148. this()
149. All of the above
150. Which is true about constructors in inheritance?
151. Child class constructors inherit parent constructors
152. Parent class constructors must be explicitly called
153. Parent constructors are automatically invoked
154. Java allows constructor inheritance
155. What is a practical use of super.method() in a subclass?
156. Hide the superclass method
157. Call a private method
158. Execute overridden logic from the superclass
159. Exit a constructor early
160. A field with protected access is visible to:
161. All classes everywhere
162. Only in the same class
163. Subclasses and package-level classes
164. Private only
165. Constructor chaining using super() allows:
166. Inheritance of final methods
167. Calling superclass constructors
168. Disabling polymorphism
169. Skipping object instantiation
170. Which class cannot be inherited?
171. One with only static methods
172. One marked final
173. One with a constructor
174. One marked abstract
175. How can a subclass invoke a hidden field of its superclass?
176. this.field
177. super.field
178. static.field
179. private.field
180. Which inheritance type is used in class C extends A, B?
181. Multilevel
182. Single
183. Multiple – not allowed
184. Interface-based
185. Abstract classes are used when:
186. Full implementation is provided
187. Object instantiation is needed
188. You want to enforce method contracts but leave implementation to subclasses
189. You want to disable polymorphism
190. What’s the most accurate statement about Java inheritance?
191. Multiple inheritance of classes is supported
192. Constructors are inherited
193. Static methods are polymorphic
194. Java uses single class inheritance but allows multiple interface implementation
195. What does polymorphism allow in Java?
196. Writing multiple classes without hierarchy
197. Defining multiple methods with same name
198. Using one interface for different types
199. Accessing private fields
200. Method overriding supports:
201. Compile-time polymorphism
202. Runtime polymorphism
203. Static polymorphism
204. Encapsulation only
205. Which of the following best demonstrates **runtime polymorphism**?
206. Method overloading
207. Method overriding
208. Constructor overloading
209. Static imports
210. Which class members are subject to polymorphism?
211. private methods
212. static methods
213. instance methods
214. constructors
215. Given:

class A { void show() { System.out.println("A"); } }

class B extends A { void show() { System.out.println("B"); } }

What is printed by:

A obj = new B(); obj.show();

1. A
2. B
3. Compilation error
4. Runtime exception
5. Polymorphism improves:
6. Memory usage
7. Reusability and flexibility
8. Compile speed
9. Variable visibility
10. Can we override a method with a different return type?
11. Yes, if it’s a covariant type
12. Yes, any return type
13. No
14. Only if method is static
15. Choose a valid overloaded method pair:
16. void print(int a) and void print(int a)
17. int print(int a) and String print(int a)
18. void print() and void print(int)
19. static void print() and final void print()
20. What does dynamic method dispatch mean?
21. Binding method at compile-time
22. Binding method at runtime based on object
23. Printing objects dynamically
24. Executing constructors
25. In polymorphism, the reference type determines:
26. The actual object behavior
27. The variable name
28. What members are accessible at compile-time
29. JVM output
30. Method overloading depends on:
31. Return type
32. Access modifier
33. Parameters
34. Object type
35. What is required to achieve polymorphism?
36. Same class
37. Overridable methods
38. Final classes
39. Static binding only
40. Choose the correct statement:
41. Overloaded methods must differ in access modifiers
42. Static methods can participate in runtime polymorphism
43. Overloaded methods can vary in parameter count
44. Constructors cannot be overloaded
45. What is **method signature** used for in polymorphism?
46. To create private fields
47. To distinguish overloaded methods
48. To hide superclass behavior
49. For constructor chaining
50. Interface-based polymorphism allows:
51. One method to act on multiple types
52. Single inheritance
53. Static methods
54. Abstract constructors
55. Which version of draw() is invoked here?

Shape s = new Circle();

s.draw();

1. Shape’s draw
2. Circle’s draw
3. Compilation error
4. Depends on compiler
5. Polymorphism ensures:
6. Tight coupling
7. Implementation hiding
8. Code generality and reuse
9. Object immutability
10. Java supports polymorphism primarily via:
11. Static method overloading
12. Instance variable hiding
13. Method overriding and interfaces
14. Default constructors
15. Which rule applies to overriding methods?
16. Parameters must change
17. Access level can be more restrictive
18. Return type can be subclass of original
19. Use private keyword
20. What is output?

class Animal {

void sound() { System.out.println("Generic"); }

}

class Dog extends Animal {

void sound() { System.out.println("Bark"); }

}

Animal a = new Dog();

a.sound();

1. Generic
2. Bark
3. Compilation error
4. Animal
5. Polymorphism is often used with:
6. Composition
7. Arrays of objects
8. Static members
9. Final classes
10. Which statement is **false** about polymorphism?
11. It’s only possible through inheritance
12. Interfaces can help achieve it
13. Overriding must match signature
14. JVM decides overridden method at runtime
15. Polymorphism allows objects to be:
16. Instantiated without class
17. Used interchangeably via a base reference
18. Accessed by static methods only
19. Declared final
20. Method overriding cannot be used with:
21. Abstract classes
22. Static methods
23. Dynamic dispatch
24. Virtual functions
25. What is the major difference between overloading and overriding?
26. Overloading requires same signature
27. Overriding is compile-time
28. Overloading happens in same class; overriding across hierarchy
29. Both are same in Java
30. What is the primary goal of encapsulation?
31. Reduce inheritance
32. Hide internal state and enforce abstraction via behavior
33. Improve runtime efficiency
34. Increase visibility of fields
35. Which of these classes most strongly violates encapsulation?

class Data {

public int count;

void process() { count++; }

}

1. Uses encapsulation correctly
2. Violates due to public field
3. Violates due to method design
4. Is abstract
5. What concept defends against unintended modification of object state?
6. Inheritance
7. Overloading
8. Data Hiding
9. Upcasting
10. A final field that’s private and has no setter behaves as:
11. Mutable
12. Unencapsulated
13. Read-only with full encapsulation
14. Method-local
15. Which technique ensures immutability in encapsulation?
16. Private fields + setters
17. Private final fields + constructor initialization
18. Public constructors + mutable fields
19. Protected variables
20. Which of the following is a **breach** of encapsulation?
21. Using private fields with getters
22. Returning a direct reference to a mutable object field
23. Hiding constructor logic
24. Declaring fields private final
25. What risk arises when returning a reference to an internal array?
26. Performance lag
27. Index misalignment
28. External code can mutate internal state
29. Compilation fails
30. To make a class **fully immutable**, which is NOT required?
31. Class must be final
32. All fields private final
33. Mutable fields must be deeply copied
34. Implement interface
35. Defensive copying helps maintain:
36. Coupling
37. Abstraction
38. Encapsulation
39. Code size
40. What best demonstrates strict encapsulation?
41. Public variables
42. private List<String> items; with unmodifiable accessors
43. Protected attributes
44. Static methods
45. Which combination best supports encapsulation and information hiding in Java?
46. public fields and private methods
47. private fields and public methods
48. protected fields and final methods
49. static fields and abstract methods
50. Which is true about setters in encapsulated designs?
51. They break encapsulation
52. They allow validation before updating fields
53. They’re required for every field
54. They always reduce security
55. What Java keyword is essential for implementing encapsulation effectively?
56. synchronized
57. final
58. private
59. default
60. What kind of class design breaks encapsulation and reduces modularity?
61. Immutable classes
62. Classes with global variables
63. Encapsulated beans
64. Utility classes
65. In a well-encapsulated class, how should a mutable field like List<T> be exposed?
66. With public List<T>
67. With public T[] toArray()
68. With Collections.unmodifiableList() wrapper
69. With final List<T>
70. Which annotation or pattern can **protect object invariants** in an encapsulated class?
71. @Final
72. Builder pattern
73. @Override
74. Singleton
75. A violation of encapsulation can cause:
76. Tightly controlled access
77. Predictable interfaces
78. Security holes and maintenance issues
79. Improved visibility
80. Choose the best pattern for sensitive field exposure:
81. Return the field directly
82. Use only private constructors
83. Return a clone of the field object
84. Avoid method access
85. Which principle supports the idea: “Expose behavior, not state”?
86. Inheritance
87. Polymorphism
88. Encapsulation
89. Data duplication

95. Why should internal mutable fields not be exposed directly?

1. It reduces compile time
2. It leads to accidental thread synchronization
3. It allows modification from outside violating object contracts
4. It helps in subclass inheritance
5. How does encapsulation relate to **modifiability**?
6. Changes to internal representation do not affect external code
7. Makes subclassing harder
8. Requires more memory
9. Reduces method overriding
10. What’s the **key encapsulation violation** here?

public class Score {

private int value;

public int getValue() { return value; }

public int setValue(int val) { return val; }

}

1. Value is not initialized
2. No default constructor
3. setValue doesn’t update field
4. getValue should be private
5. Encapsulation improves **modularity** by:
6. Locking down all fields
7. Centralizing access control
8. Increasing code length
9. Preventing code reuse
10. A setter that accepts invalid data and updates a field regardless breaks:
11. Inheritance
12. Encapsulation logic
13. Interface inheritance
14. Static binding
15. Which return type is preferred for exposing internal Map<K,V> with safety?
16. Map<K,V>
17. null
18. Collections.unmodifiableMap(map)
19. LinkedHashMap<K,V>

101.What happens if a superclass has a parameterized constructor and the subclass does not explicitly call it?

1. Compile-time error
2. It calls super() automatically
3. It compiles but fails at runtime
4. JVM auto-generates a call with default values

102.Which inheritance scenario would cause the Diamond Problem in Java (if not handled by interfaces)?

1. Multiple inheritance from classes
2. Multilevel inheritance
3. Hierarchical inheritance
4. Single inheritance

103.What is the role of the super keyword in method overriding?

1. Replace the parent method
2. Access overridden method of superclass
3. Deny access to the parent
4. Prevent subclassing

104.Consider:

class A { A() { System.out.print("A "); } }

class B extends A { B() { System.out.print("B "); } }

class C extends B { C() { System.out.print("C "); } }

What’s the output when new C(); is called?

1. C B A
2. A B C
3. C
4. Compilation error

105.Which of the following is true about the constructor execution order?

1. Subclass constructor is executed first
2. Superclass constructor is executed first
3. Constructors are executed randomly
4. No relation exists

106.Can you override a static method in Java?

1. Yes, always
2. No, because static methods belong to class not instance
3. Only if the method is public
4. Only inside nested classes

107.Given:

class Parent { final void display() {} }

class Child extends Parent { void display() {} }

What is the outcome?

1. Valid override
2. Compilation error
3. Runtime exception
4. Child’s method hides parent’s

108.What happens if both the subclass and superclass have static methods with the same signature?

1. It overrides
2. It hides the superclass method
3. Compilation error
4. JVM throws ambiguity

109.If a class is abstract, then:

1. You can’t inherit it
2. You must override all its methods
3. It cannot have a constructor
4. It cannot be instantiated

110.Why can't multiple inheritance with classes be implemented in Java?

1. Ambiguity of method resolution
2. JVM limitation
3. It increases compile time
4. Java does allow it

111.What does super() do when placed in the first line of a constructor?

1. Initializes the subclass
2. Calls superclass constructor
3. Calls overridden method
4. Calls static block

112.Given:

class A { int i = 10; }

class B extends A { int i = 20; void show() { System.out.println(super.i); } }

What will new B().show(); output?

1. 10
2. 20
3. Compilation error
4. 0

113.Which feature helps Java simulate multiple inheritance?

1. final classes
2. interfaces
3. static imports
4. generics

114.Which one cannot be overridden?

1. private methods
2. protected methods
3. abstract methods
4. public methods

115.Java supports constructor chaining through:

1. final methods
2. Interfaces
3. super() and this()
4. Overloaded methods

116.Which class hierarchy is valid?

1. abstract → interface → concrete
2. interface → abstract → concrete
3. interface → interface → concrete
4. concrete → abstract → interface

117.What happens if an abstract class doesn’t have any abstract methods?

1. Compilation error
2. Can’t create instance
3. Still valid
4. Must declare an interface

118.A subclass constructor always calls:

1. super() or another constructor in the same class
2. Only super()
3. Only this()
4. JVM handles it silently

119.Given:

interface A { default void print() { System.out.print("A"); } }

interface B { default void print() { System.out.print("B"); } }

class C implements A, B { public void print() { A.super.print(); } }

What is the output of new C().print();?

1. A
2. B
3. Compilation Error
4. A B

120.Which of the following can’t be inherited?

1. final method
2. static method
3. constructor
4. all the above

121.What concept allows multiple levels of class inheritance in Java?

1. Composition
2. Chaining
3. Multilevel Inheritance
4. Aggregation

122.Can abstract classes have constructors?

1. No
2. Only private ones
3. Yes
4. Only if they’re called with super()

123.What’s inherited when a class implements an interface?

1. Code
2. Declarations
3. Object
4. Nothing

124.Which design flaw is often a result of excessive inheritance?

1. Cohesion
2. Low complexity
3. Fragile base class problem
4. High cohesion

125.In case of method overriding, which method is executed at runtime?

1. Superclass method
2. Subclass method
3. Compiler’s choice
4. JVM chooses randomly

126.What is achieved through dynamic polymorphism in Java?

1. Compile-time method binding
2. Method overloading
3. Runtime method overriding
4. Object cloning

127.Which of these is not a valid example of polymorphism?

1. Method overriding
2. Method overloading
3. Variable hiding
4. Interface implementation

128.Given:

class A { void show() { System.out.println("A"); } }

class B extends A { void show() { System.out.println("B"); } }

A obj = new B(); obj.show();

Output?

1. A
2. B
3. Compilation error
4. Runtime exception

129.Polymorphism supports:

1. Method abstraction only
2. Static binding only
3. Both compile-time and runtime binding
4. No binding at all

130.What does overriding mean in Java?

1. Same method name and different signatures
2. Defining a superclass method inside subclass with same signature
3. Overloading static blocks
4. Multiple constructors

131.Can you override a method with a more restrictive access modifier?

1. Yes
2. No
3. Only if method is static
4. Only in interfaces

132.Choose the one true about overridden methods:

1. Return types must match exactly
2. Return types can be covariant
3. Parameters must differ
4. Overriding happens only for static methods

133.Consider:

class A { void test() { System.out.println("A"); } }

class B extends A { void test(int x) { System.out.println("B"); } }

new B().test();

Output?

1. A
2. B
3. Compilation error
4. No output

134.Which rule must method overloading follow?

1. Same method name, same signature
2. Same name, different parameter list
3. Must return same type
4. Must change access level

135.Can a constructor be overloaded?

1. No
2. Yes
3. Only in subclasses
4. Not with final

136.Which version of the method is called when a superclass reference holds a subclass object?

1. Superclass method always
2. Subclass method if overridden
3. It depends on JVM
4. Both are called

137.What is covariant return type?

1. Changing method name on override
2. Using superclass return in subclass
3. Return type of subclass method is subtype of overridden method
4. Multiple return types

138.Polymorphism enables:

1. Faster compilation
2. Interface redefinition
3. Uniform access to multiple object types
4. Abstract class inheritance only

139.When are overloaded methods resolved?

1. Compile-time
2. Runtime
3. Dynamically
4. Never

140.Which of the following violates polymorphism?

1. Multiple methods with same name, different params
2. Two unrelated classes with same method
3. Downcasting with invalid object
4. Overriding public methods

141.What happens if you overload a method with identical parameters but different return types?

1. Compilation succeeds
2. Method is overridden
3. Compilation error
4. JVM resolves using return type

142.Why can we achieve polymorphism with interfaces?

1. Interfaces support static methods
2. Interface types can refer to different implementing objects
3. Interfaces don’t support abstraction
4. It depends on constructors

143.What’s the danger of unchecked downcasting?

1. Runtime crash due to ClassCastException
2. Compile-time error
3. Slower program
4. Syntax error

144.What’s the output?

class A { int print() { return 1; } }

class B extends A { long print() { return 2; } }

1. 1
2. 2
3. Compile error
4. Runtime error

145.Which statement is true about static methods and polymorphism?

1. They can be overridden
2. They exhibit dynamic behavior
3. They don’t participate in polymorphism
4. JVM dispatches based on object type

146.What best describes method hiding in Java?

1. Subclass redefines static method of superclass
2. Subclass hides instance method
3. Interface hides implementing method
4. Abstract method override

147.When calling a static method via reference variable of superclass pointing to subclass object:

1. Superclass method executes
2. Subclass method executes
3. Compile error
4. JVM chooses

148.Why is method overriding considered a form of polymorphism?

1. Code duplication
2. Allows method behavior to vary based on object type
3. It allows static resolution
4. It improves execution speed

149.Given:

interface Shape { void draw(); }

class Circle implements Shape { public void draw() { System.out.print("Circle"); } }

How can polymorphism be demonstrated?

1. new Shape()
2. Shape s = new Circle(); s.draw();
3. Circle.draw();
4. Not possible

150.What does instanceof operator help with in polymorphism?

1. Resolve overloaded methods
2. Avoid casting
3. Safely check object’s type at runtime
4. Override abstract methods

151.Which OOP concept allows object behavior to change based on context?

1. Inheritance
2. Overloading
3. Polymorphism
4. Encapsulation

152.Why is constructor overloading useful?

1. It enables finalization
2. It hides methods
3. It allows different ways to instantiate objects
4. It adds security

153.Which modifier restricts inheritance of a class?

1. static
2. final
3. abstract
4. protected

154.What is true about constructors in inheritance?

1. They are inherited directly
2. Must call super()
3. They’re always abstract
4. Can’t be overloaded

155.What does super.variable access?

1. Subclass copy
2. Static block
3. Superclass field
4. Default constructor

156.What is encapsulation primarily used for?

1. Code reuse
2. Data hiding and modularity
3. Performance enhancement
4. Object replication

157.Which of the following supports partial implementation?

1. Abstract class
2. Interface
3. Final class
4. Static method

158.Which of these is false about polymorphism?

1. Enables flexibility
2. Always compile-time
3. Supports multiple implementations
4. Enables method overriding

159.What breaks polymorphism in an overridden method?

1. Matching signature
2. Same return type
3. More restrictive access
4. Superclass call

160.What happens when an interface defines a method but the class doesn’t implement it?

1. Compilation error
2. Runtime error
3. JVM auto-defines
4. Method is hidden

161.Which class structure violates encapsulation?

1. private fields + public getters
2. public fields accessed directly
3. final fields set via constructor
4. fields exposed via interface

162.Can an interface contain constructors?

1. Yes
2. No
3. Only default
4. Depends on version

163.How does overriding differ from overloading?

1. Overriding requires inheritance
2. Overloading needs same return
3. Overriding only works on private methods
4. Overloading supports runtime dispatch

164.What does a method override require?

1. Same name, parameters, and accessible modifier
2. Different access
3. Static context
4. Interface link

165.How does encapsulation enhance security?

1. Protects constructors
2. Exposes data globally
3. Restricts direct field access
4. Duplicates objects

166.When do we use abstract classes over interfaces?

1. When multiple inheritance is needed
2. To provide base functionality
3. Only for runtime polymorphism
4. When hiding constructors

167.Choose the correct statement:

1. Static methods support overriding
2. Private methods are polymorphic
3. Subclass can override a superclass method with broader access
4. Final methods are overridable

168.Which polymorphic behavior fails at runtime?

1. Upcasting
2. Overriding
3. Invalid downcasting
4. Interface binding

169.Which principle aligns best with “design to interface, not implementation”?

1. Polymorphism
2. Encapsulation
3. Abstraction
4. Static typing

170.What happens when a subclass re-declares a field already present in parent?

1. Parent field overrides
2. Subclass hides parent field
3. Compilation fails
4. JVM ignores both

171.What enables method call resolution at runtime?

1. Overloading
2. Overriding
3. Static binding
4. Anonymous blocks

172.Encapsulation is broken by:

1. Public fields
2. Getter returning mutable objects
3. Final methods
4. Private fields with no access

173.What happens if you extend a final class?

1. Inherits methods
2. Gets compiled
3. Compilation fails
4. Skips constructors

174.What determines accessibility of inherited members?

1. Method name
2. Access modifier
3. Class name
4. Parameter count

175.How can polymorphism reduce coupling?

1. Forces subclassing
2. Supports interfaces and abstraction
3. Uses static inheritance
4. Avoids constructor chains

176.Which design principle is violated when a subclass modifies its parent’s internal logic via public/protected fields?

1. Abstraction
2. Loose coupling
3. Encapsulation
4. Polymorphism

177.Consider:

class A { void display() { System.out.print("A"); } }

class B extends A { void display() { System.out.print("B"); } }

A obj = new B();

What principle is this line demonstrating?

1. Static binding
2. Interface inheritance
3. Polymorphism (upcasting)
4. No OOP principle

178.What will this code print?

interface I { default void show() { System.out.print("I"); } }

class X implements I { public void show() { System.out.print("X"); } }

class Y extends X { public void show() { super.show(); System.out.print("Y"); } }

new Y().show();

1. XY
2. IY
3. XIY
4. IXY

179.Can a subclass access a private method of its superclass through inheritance?

1. Yes, always
2. No, not directly
3. Only if using super
4. Only for final classes

180.If both subclass and superclass define a field with same name, which one is accessed via reference of superclass?

1. Subclass field
2. Compilation error
3. Superclass field
4. Undefined behavior

181.What pattern describes an interface with no methods or fields?

1. Abstract interface
2. Empty contract
3. Marker interface
4. Inheritance-only

182.What access level should a class use to fully restrict access outside the package?

1. public
2. private
3. protected
4. default

183.Which of the following demonstrates both encapsulation and polymorphism?

class Book { private String title; public void print() { System.out.println(title); } }

1. Yes
2. No
3. Only encapsulation
4. Only inheritance

184.Which Java construct allows a single method name to behave differently based on parameter types?

1. Overriding
2. Overloading
3. Encapsulation
4. Abstract class

185.What is one benefit of polymorphism in large-scale applications?

1. Reduces interface complexity
2. Improves memory usage
3. Eliminates inheritance needs
4. Enables extensible and maintainable code

186.When is a method selected at compile time rather than runtime?

1. Overriding
2. Overloading
3. Interface implementation
4. Finalization

187.Can constructors be polymorphic?

1. Yes, via overloading
2. Yes, via overriding
3. No
4. Only for interfaces

188.What happens when a subclass doesn’t call its parent constructor explicitly?

1. Compiler inserts super()
2. Program crashes
3. Constructor is skipped
4. JVM throws error

189.How can a class achieve both inheritance and flexibility from multiple sources in Java?

1. Multiple extends
2. Combining abstract class and interface
3. Runtime imports
4. Static inheritance

190.What happens when a method returns a type more specific than the one declared in superclass?

1. Compile error
2. It’s allowed if return types are covariant
3. Run-time exception
4. Method is hidden

191.What rule applies when a subclass overrides a method and throws a checked exception not declared in the superclass method?

1. Compilation passes
2. Compile-time error
3. Runtime error
4. JVM decides

192.Consider:

class Vehicle { public void start() throws IOException {} }

class Car extends Vehicle { public void start() throws Exception {} }

What’s the outcome?

1. Compiles and runs
2. Compilation error
3. Runtime exception
4. Inheritance ignored

193.In which OOP principle do you define an interface and use multiple implementations interchangeably?

1. Inheritance
2. Abstraction
3. Polymorphism
4. Coupling

194.Why are protected fields dangerous in large class hierarchies?

1. Promote modularity
2. Encourage tight coupling and fragile base class
3. Prevent access from subclasses
4. Better than private

195.Given:

class A { private int x = 5; public int getX() { return x; } }

class B extends A { private int x = 10; }

What does new B().getX(); return?

1. 5
2. 10
3. 0
4. Compilation fails

196.What is the primary advantage of encapsulating fields and exposing behavior via methods?

1. Performance
2. Loose coupling and change control
3. Access speed
4. More memory

197.Which one is true about abstract classes and interfaces?

1. Abstract classes can’t have constructors
2. Interfaces can’t contain constants
3. Abstract classes can include implemented methods
4. Interfaces support multiple inheritance via extends

198.Which scenario best demonstrates interface-based polymorphism?

1. Using different constructors in same class
2. Overriding methods with different access
3. Assigning subclass object to interface reference
4. Calling superclass static method

199.Which design pattern promotes encapsulation and hides implementation details behind a public API?

1. Factory
2. Singleton
3. Adapter
4. Facade

200.Which of the following violates the Liskov Substitution Principle?

1. Subclass enhances base method
2. Subclass returns narrower type
3. Subclass throws broader exception
4. Subclass changes method contract behavior

**Answer Key:**

**1.c 2.a 3.b 4.c 5.c 6.a 7.c 8.c 9.b 10.b**

**11.b 12.b 13.b 14.d 15.a 16.c 17.b 18.b 19.c 20.a**

**21.a 22.b 23.c 24.c 25.a 26.b 27.b 28.c 29.b 30.b**

**31.a 32.c 33.b 34.d 35.c 36.c 37.c 38.b 39.c 40.b**

**41.d 42.c 43.c 44.c 45.b 46.b 47.b 48.c 49.c 50.d**

**51.c 52.b 53.b 54.c 55.b 56.b 57.a 58.c 59.b 60.c**

**61.c 62.b 63.c 64.b 65.a 66.b 67.c 68.c 69.c 70.b**

**71.b 72.a 73.b 74.b 75.c 76.b 77.b 78.c 79.c 80.b**

**81.b 82.c 83.d 84.c 85.b 86.b 87.b 88.c 89.b 90.c**

**91.b 92.c 93.c 94.c 95.c 96.a 97.c 98.b 99.b 100.c**

**101.a 102.a 103.b 104.b 105.b 106.b 107.b 108.b 109.d 110.a**

**111.b 112.a 113.b 114.a 115.c 116.b 117.c 118.a 119.a 120.d**

**121.c 122.c 123.b 124.c 125.b 126.c 127.c 128.b 129.c 130.b**

**131.b 132.b 133.a 134.b 135.b 136.b 137.c 138.c 139.a 140.c**

**141.c 142.b 143.a 144.c 145.c 146.a 147.a 148.b 149.b 150.c**

**151.c 152.c 153.b 154.b 155.c 156.b 157.a 158.b 159.c 160.a**

**161.b 162.b 163.a 164.a 165.c 166.b 167.c 168.c 169.c 170.b**

**171.b 172.b 173.c 174.b 175.b 176.c 177.c 178.a 179.b 180.c**

**181.c 182.d 183.a 184.b 185.d 186.b 187.a 188.a 189.b 190.b**

**191.b 192.b 193.c 194.b 195.a 196.b 197.c 198.c 199.d 200.d**