POLYMORPHISM IN JAVA

1.What is polymorphism in Java?

Polymorphism allows objects of different classes to be treated as objects of their common superclass, enabling them to respond to the same method invocations in different ways.

2.What are the two main types of polymorphism in Java?

Method overriding: Subclass method with the same name and signature as a superclass method, providing specific implementation for the subclass.
Method overloading: Multiple methods in the same class with the same name but different parameter lists, allowing different ways to call the method with different arguments.

3.What are the key requirements for method overriding?

Same name and signature (return type, name, parameter types)
Non-private superclass method
Non-final superclass method
Subclass method accessibility must be the same or broader than the superclass method.

4.How does dynamic binding work in polymorphism?

At runtime, the actual object's type determines which method implementation is invoked, not the reference variable type used.

5.What is the benefit of using polymorphism?

More flexible and maintainable code, promotes code reusability, allows for generic algorithms that work with different object types.

6.Can you explain the difference between static and dynamic binding?

Static binding: Compiler determines the method to call based on the reference variable type at compile time.
Dynamic binding: JVM determines the method to call based on the actual object's type at runtime.

7.What is method overloading useful for?

Providing different functionalities based on the provided arguments.
Enhancing code readability and clarity.

8.Can you explain the instanceof operator and its relation to polymorphism?

Used to check if an object is an instance of a specific class or interface.
Helps in casting reference variables to specific types safely during runtime.

9.What are the limitations of polymorphism?

Potential performance overhead due to dynamic dispatch.
Increased complexity if not used carefully.

10.Can you provide an example of polymorphism using inheritance and interfaces?

Example: Shape class with subclasses Square, Circle, and Triangle, all implementing a draw() method, each drawing their respective shapes differently.

11.Describe a scenario where polymorphism might lead to unexpected behavior. How would you debug it?

Incorrect method overriding or overloading definitions can lead to unintended calls. Use debugging tools, instanceof checks, and careful review of method implementations.

12.What are some best practices for using polymorphism

effectively?

Clear and concise method names.
Consistent use of inheritance and interfaces.
Careful design of overloaded methods.
Thorough testing of polymorphic code.

13.How can polymorphism be used to improve code design principles like cohesion and loose coupling?

Code becomes more focused on functionalities rather than specific implementations.
Dependence on specific classes is reduced, promoting reusability and maintainability.

14.How can polymorphism be used in conjunction with generics in Java?

Generic methods and collections can operate on different types of objects while maintaining type safety.

15.Can you discuss the role of polymorphism in design patterns like the Strategy pattern and the Template Method pattern?

These patterns leverage polymorphism to dynamically choose different algorithms or implementations based on context.