1. Advantages of Polymorphism:

Polymorphism offers several benefits that enhance code readability, maintainability, and efficiency:

- Code Reusability: Methods with the same name can be reused across different classes, promoting code reusability.
- Flexibility and Dynamism: Polymorphism enables dynamic behavior where an object's behavior can change at runtime, depending on its context.
- Reduced Complexity: By using the same method name for related functionality, polymorphism reduces code complexity and enhances readability.
- Simplified Coding: Polymorphism simplifies coding by minimizing the number of methods and constructors required.
- Better Organization: It facilitates better code organization by consolidating related functionality within a single class.
- Code Extensibility: Polymorphism supports code extensibility by allowing new subclasses to extend the functionality of the superclass without modifying existing code.
- Increased Efficiency: Compile-time polymorphism can lead to more efficient coding, as the compiler can determine the appropriate method to call based on arguments passed.

2. Inheritance's Role in Polymorphism in Java:

- Inheritance allows a class to acquire properties and attributes from another class.
- Polymorphism leverages inherited properties to perform different tasks, achieving the same action in various ways.

3. Differences between Polymorphism and Inheritance in Java:

    a. Aspect:

- Inheritance: A new class is derived from an existing class, inheriting its properties and methods.
- Polymorphism: Objects of different classes can be treated as instances of a common superclass through interfaces and abstract classes.

    b. Purpose:

- Inheritance: Achieve code reusability and establish relationships between classes.
- Polymorphism: Provide flexibility by treating different classes as instances of the same class, particularly with methods sharing the same name.

    c. How It Works:

- Inheritance: Child class inherits attributes and behaviors from the parent class and may have its own unique attributes and behaviors.
- Polymorphism: Involves methods with the same name behaving differently in different classes; the invoked method is determined at runtime.

d. Types:

- Inheritance: Single, multiple, multilevel, hierarchical, hybrid inheritance.
- Polymorphism: Method overloading (compile-time polymorphism) and overriding (runtime polymorphism).

e. Key Principle:

- Inheritance: "IS-A" relationship. Example: A Dog is an Animal.
- Polymorphism: "CAN-DO" relationship. Example: A Printer can print in different ways.

f. Implementation:

- Inheritance: Achieved through class definitions, using the extends keyword in Java.
- Polymorphism: Achieved through method overloading and overriding, often involving interfaces or abstract classes.

g. Usage Example:

- Inheritance: A class Car inherits from a class Vehicle, having all Vehicle's attributes and methods plus its own.
- Polymorphism: A draw function implemented differently for drawing Circle, Square, or Triangle shapes.

h. Flexibility:

- Inheritance: Static and defined at class creation time.
- Polymorphism: Dynamic, providing a flexible interface for object interactions.

i. Limitation:

- Inheritance: Deep inheritance hierarchies can become complex and hard to manage.
- Polymorphism: May lead to confusion about which method is being called if not properly managed.