1. What is the advantages of Polymorphism?

- Polymorphism provides many advantages that can improve the readability, maintainability, and efficiency of code, such as:

- Code Reusability: Polymorphism provides the reuse of code, as methods with the same name can be used in different classes.

- Flexibility and Dynamism: Polymorphism allows for more flexible and dynamic code, where the behaviour of an object can change at runtime depending on the context in which it is being used.

- Reduced Complexity: It can reduce the complexity of code by allowing the use of the same method name for related functionality, making the code easier to read and maintain.

- Simplified Coding: Polymorphism simplifies coding by reducing the number of methods and constructors that need to be written.

- Better Organization: Polymorphism allows for better organization of code by grouping related functionality in one class.

- Code Extensibility: Polymorphism enables code extensibility, as new subclasses can be created to extend the functionality of the superclass without modifying the existing code.

- Increased Efficiency: Compile-time polymorphism can lead to more efficient coding. The compiler can choose the appropriate method to call at compile time, based on the number, types, and order of arguments passed to it.

2. How is inheritance useful when using polymorphism in java?

- Inheritance is a powerful feature in Java. Java Inheritance lets one class acquire the properties and attributes of another class.
- Polymorphism in Java allows us to use these inherited properties to perform different tasks.
- Thus, allowing us to achieve the same action in many different ways

3. What are the differences between Polymorphism and Inheritance in Java?

Aspect	Inheritance	Polymorphism
Definition	Inheritance is a mechanism where a new class is derived from an existing	Polymorphism allows objects of different classes to be treated as objects of a common super class,

	class, inheriting its properties and methods.	primarily through the use of interfaces and abstract classes.
Purpose	Used to achieve reusability of code and establish a relationship between classes (parent-child relationship).	Used to achieve flexibility in code by allowing different classes to be treated as instances of the same class, particularly when their methods share the same name.
How It Works	The child class inherits attributes and behaviors (methods) from the parent class and can also have its own unique attributes and behaviors.	Involves methods that have the same name but may behave differently in different classes. The exact method that gets invoked is determined at runtime.
Types	Single inheritance, multiple inheritance, multilevel inheritance, hierarchical inheritance, hybrid inheritance.	Overloading (compile-time polymorphism) and overriding (runtime polymorphism).
Key Principle	“IS-A” relationship. For example, a Dog is an Animal.	“CAN-DO” relationship. For example, a Printer can print in different ways.
Implementation	Achieved through class definitions. In languages like Java, extends keyword is used.	Achieved through method overloading and overriding. Interfaces or abstract classes are often involved.
Usage Example	A class Car inherits from a class Vehicle. The Car will have all attributes and methods of Vehicle, plus its own unique attributes and methods.	A function draw could be implemented in multiple ways depending on whether it’s drawing a Circle, Square, or Triangle. Each shape will have its own implementation of draw.
Flexibility	It is static and defined at the time of class creation.	It is dynamic and can provide a more flexible interface for interactions between objects.
Limitation	Deep inheritance hierarchies can become complex and hard to manage.	If not properly managed, it can lead to confusion about which method is being called.