1. What are the advantages of Polymorphism?

Polymorphism offers numerous benefits for code readability, maintenance, and efficiency such as:

- **Flexibility and Dynamics**: It fosters flexible and dynamic code where an object's behaviour can change at runtime based on the context that it is used.
- **Extensibility**: It facilitates code extension by enabling the creation of new subclasses to expand the functionality of the superclass without the need of modifying existing code.
- **Reusability**: It enables code reuse since methods with the same name can be used in different classes.
- **Streamlining Development**: It simplifies coding by reducing the number of methods and constructors that need to be created.
- **Organisation Enhancement**: It aids in organising code by grouping related functionalities in one class.
- **Reducing Complexity**: Polymorphism simplifies code by using the same method name for related tasks, making the code easier to understand and maintain.
- **Efficiency Improvement**: Compile-time polymorphism can enhance efficiency by allowing the compiler to select the appropriate method to call based on the arguments passed to it.

2. How is Inheritance useful to achieve Polymorphism in Java?

In Java, Inheritance is a powerful feature that allows a class to inherit the properties and attributes of another class, whereas Polymorphism enables the utilisation of the inherited properties to execute different tasks. Therefore, through numerous different approaches, we can still accomplish the identical task.

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

- Definition: Inheritance is a mechanism where a class is derived from an already existing class and inherits its properties and methods, whereas Polymorphism enables objects of different classes to be treated as objects of a common super class, primarily through the use of interfaces and abstract classes
- Purpose: Inheritance is used to achieve the reusability of code and establish a relationship between classes(parent-child relationship), whereas Polymorphism is 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.
- Types:
    - Inheritance: Single inheritance, Multiple inheritance, Multilevel inheritance, Hierarchical inheritance, Hybrid inheritance.
    - Polymorphism: Overloading(compile-time Polymorphism) and Overriding (runtime Polymorphism).

- Usage:
    - Inheritance:
        + The child class inherits attributes and behaviours(methods) from the parent class and can also have its own unique attributes and behaviours.
        + “IS-A” relationship, a car is a vehicle, for example.
    - Polymorphism:
        + Involves methods that have the same name but may behave differently in different classes. The exact method that gets invoked is determined at runtime.
        + “CAN-DO” relationship, a design engineer can produce different house designs, for example.
- Implementation: Inheritance is achieved through class definitions, and extends keyword is used in languages like Java, whereas Polymorphism is achieved through 2 methods overloading and overriding, and interfaces or abstract classes are often involved.
- Flexibility: When it comes to Inheritance, it is static and defined at the time of class creation, whereas in Polymorphism it is dynamic and can provide a more flexible interface for interactions between objects.
- Limitation:
    - Inheritance: Deep inheritance hierarchies can become complex and hard to manage.
    - Polymorphism: It can lead to confusion about which method is being called if not managed properly.