Reading Assignment

Đinh Ngọc Cầm – 20226016 – ICT01

1. What is the advantages of Polymorphism?

- Code Reusability: Polymorphism allows methods to be written to operate on objects of a superclass, and these methods can be reused for objects of any subclass

- Flexibility and Extensibility: Polymorphism allows for flexible and extensible code design.

- Readability and Maintainability: Polymorphism enhances code readability by promoting a more generalized approach to programming.

- Dynamic Method Binding: Polymorphism in Java enables dynamic method binding, where the specific method to be invoked is determined at runtime based on the type of object.

- Encapsulation: Polymorphism helps in achieving encapsulation by hiding the implementation details of classes from the outside world.

- Interface Implementation: This encourages programming to interfaces rather than implementations, which leads to more modular and loosely coupled code.

- Method Overriding: Polymorphism allows subclasses to provide specific implementations of methods defined in their superclass.

- Run-Time Polymorphism: Also known as dynamic polymorphism, this feature allows the same method to behave differently based on the type of object it is called on.

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

- Code Reusability: Inheritance allows a subclass to inherit fields and methods from its superclass. When combined with polymorphism, this facilitates code reusability.

- Method Overriding: Inheritance enables subclasses to override methods defined in their superclass. This is a key aspect of polymorphism, as it allows objects of different classes to be treated uniformly through a common interface

- Dynamic Binding: Inheritance plays a crucial role in enabling this dynamic binding mechanism. When a method is invoked on a superclass reference that refers to a subclass object, the JVM resolves the method call to the overridden method in the subclass

- Interface Implementation: Inheritance is often used in conjunction with interfaces to achieve polymorphism.

- Code Organization and Abstraction: Inheritance promotes code organization and abstraction by facilitating the creation of hierarchical class structures

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 class, inheriting its properties and methods.	Polymorphism allows objects of different classes to be treated as objects of a common super class, 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	A function draw could be implemented in multiple ways depending on whether it’s drawing

	methods of Vehicle, plus its own unique attributes and methods.	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.