CHAPTER 5

POLYMORPHISM

Contents

Key Objectives

Learning objectives of this chapter are to introduce the concept of polymorphism in OOP and to demonstrate how it is implemented in Java. By the end of the chapter, student should understand how objects of different classes can be treated through a common parent class, how method overriding enables dynamic behavior, and how polymorphism promotes flexibility and code reuse in program design. To reinforce these concepts, the chapter finishes with practical activities that allow student to apply what he has learned.

5.1 What is Polymorphism

Recapitulation 5.1

Polymorphism is one of the core concepts in OOP that allows objects to behave differently based on their specific class type. The word polymorphism means having many forms, and it comes from the Greek words poly (many) and morph (forms), this means one entity can take many forms. In Java, polymorphism allows the same method or object to behave differently based on the context.

5.2 Types of Polymorphism

Recapitulation 5.2

Java primarily supports two types of polymorphism:

1. ***Compile-Time Polymorphism*** *: This occurs when the compiler determines which method to call during the compilation phase. It is achieved through:*
   - ***Method Overloading****: Defining multiple methods in the same class with the same name but different parameters (number, type, or order).*
   - ***Operator Overloading****: For example, Java allows the + operator to perform both numeric addition and string concatenation.*
2. ***Run-Time Polymorphism*** *: This occurs when the Java Virtual Machine determines which method to call at runtime based on the actual object's type, not the reference type. It is achieved through:*
   - ***Method Overriding****: A subclass provides a specific implementation of a method that is already defined in its superclass, using the same name, return type, and parameters (method signature). This relies on inheritance and a superclass reference variable pointing to a subclass object (upcasting).*

5.3 Benefit of Polymorphism

Recapitulation 5.3

Polymorphism has many benefits which are listed below:

- ***Code Reusability**: Polymorphism allows the same method or class to be used with different types of objects, which makes the code more reusable.*
- ***Flexibility**: Polymorphism enables object of different classes to be treated as objects of a common superclass, which provides flexibility in method execution and object interaction.*
- ***Abstraction**: It allows the use of abstract classes or interfaces, enabling working with general types (like a superclass or interface) instead of concrete types (like specific subclasses), thus simplifying the interaction with objects.*
- ***Dynamic Behavior**: With polymorphism, the appropriate method to call can be selected at runtime, giving the program dynamic behavior based on the actual object type rather than the reference type, which enhances flexibility.*

5.4 Polymorphism by Inheritance

Recapitulation 5.4

- *Polymorphism by subtyping (or inheritance) is a key concept in OOP allowing derived (child) classes to share a common interface with their base (parent) class while specializing their own behavior, notably through method overriding.*
- *When a variable is declared to be of class S, the variable can be a reference to an instance of S or any of its subclasses. The inverse is not valid. Let's consider the class Pet and its subclasses Cat and Dog, the following code is valid:*

```
1    Pet petA = new Dog( );
2    Pet petB = new Cat( );
```

However, the following code is not valide:

```
1    Dog DogA = new Pet( );
```

- *The fact that the same variable can be referring to an instance of a different class results in polymorphism.*

- *The following two output statements will produce different results, depending on whether p is a Dog or a Cat:*

```
Pet p;
p = new Dog( );
System.out.println(p.speak( ));
p = new Cat( );
System.out.println(p.speak( ));
```

The speak method is called a polymorphic method.

- *If a variable is declared of type S and is referring to an instance of a subclass of S, then we must typecast the variable to the subclass when calling noninherited methods of the subclass. For example, the fetch method is defined in the Dog class only. So the followind code is invalid:*

```
Pet p;
p = new Dog( );
System.out.println(p.fetch( ));
```

To make this code valid, we need to typecast p to Dog, as follow:

```
Pet p;
p = new Dog( );
System.out.println( ((Dog)p).fetch( ) );
```

5.5 Keyword *instanceof*

Recapitulation 5.5

The `instanceof` *keyword in Java is a binary operator used to check whether an object is an instance of a specific class or implements a particular interface. It returns* `true` *if the object is an instance of the specified type and* `false` *otherwise. Its syntax is as follows:*

```
objectName instanceof className
```

`a instanceof B` *returns* `true` *if:*

- *the variable a stores a reference to an object of type B;*
- *the variable a stores a reference to an object whose class inherits from B;*

- *The variable a stores a reference to an object that implements interface B.*

Otherwise, it returns `false`*. If* `<reference-variable>` *is* `null`*,* `instanceof` *always returns* `false`*. Let's consider the class Vehicle and its subclass car and truck. The following Listing 5.1 shows an example of using the* `instanceof` *operator. The code begins by creating instances of different classes (Vehicle, Truck, and Car) and placing them in the array T. The code then displays whether each element of T is an instance of the Vehicle, Truck, or Car class. The code concludes by counting the number of trucks in T.*

```
Vehicle v = new Vehicle ();
Vehicle v1 = new Truck ();
Vehicle v2 = new Car ();
Car car1 = new Car ();
Truck truck1 = new Truck ();

Vehicle [] T = {v,v1 ,v2 ,car1 , truck1 };

for (int i=0;i<T. length ;i++)
 System.out.print (T[i] instanceof Vehicle +" ");
 // true true true true true

for (int i=0;i<T. length ;i++)
 System.out.print (T[i] instanceof Car +" ");
 // false false true true false

for (int i=0;i<T. length ;i++)
 System.out.print (T[i] instanceof Truck +" ");
 // false true false false true

int k =0;
for (int i=0;i<T. length ;i++)
 if(T[i] instanceof Truck ) k++;
System .out. println (" Number of trucks = "+k);// 2
```

Listing 5.1: Example of using instancof operator

5.6 Quick Check

Quick Check 5.1 *Polymorphism in Java means:*

① *One class having many constructors*

② *One method having many implementations*

③ *One variable having many values*

④ *One object having many references*

Quick Check 5.2 *Runtime polymorphism is achieved using:*

① *Method overloading*

② *Method overriding*

③ *Constructors*

④ *Static methods*

Quick Check 5.3 *What will be the output of the following code?*

① *A*

② *B*

③ *Compile-time error*

④ *Runtime error*

```
class A {
  void show() { System.out.println("A"); }
}
class B extends A {
  void show() { System.out.println("B"); }
}
public class Test {
  public static void main(String[] args) {
    A obj = new B();
    obj.show();
  }}
```

Quick Check 5.4 *Which one of the following statements is valid?*

```
Pet p = new Cat();
Cat c = new Pet();
```

Quick Check 5.5 *Is the following code valid?*

```
Pet p = new Dog();
System.out.println(p.fetch( ));
```

Quick Check 5.6 *Suppose Truck and Motorcycle are subclasses of Vehicle. Which of these declarations are invalid?*

```
Truck t = new Vehicle();
Vehicle v = new Truck();
Motorcycle m1 = new Vehicle();
Motorcycle m2 = new Truck();
```

Quick Check 5.7 *What is the purpose of the* `instanceof` *operator? Can we use it in polymorphism to check object type?*

Quick Check 5.8 *What will be the output of the following code?*

```
class A {
  static void show() { System.out.println("A"); }
}

class B extends A {
  static void show() { System.out.println("B"); }
}

public class Test {
  public static void main(String[] args) {
    A obj = new B();
```

```
        obj.show();
      }
  }
```

Quick Check 5.9 *What will be the output of the following code?*

```
class Vehicle {
  void start() { System.out.println("Vehicle starts");}
}

class Car extends Vehicle {
  void start() { System.out.println("Car starts"); }
}

class Bike extends Vehicle {
  void start() { System.out.println("Bike starts"); }
}

public class Test {
  public static void main(String[] args) {
    Vehicle[] v = { new Car(), new Bike() };
    for (i=0;i<v.length; i++) { v[i].start(); }
  }
}

```

Quick Check 5.10 *What will be the output of the following code?*

```
    class A {
      int x = 10;
    }

    class B extends A {
      int x = 20;
    }
```

```

    public class Test {
      public static void main(String[] args) {
        A obj = new B();
        System.out.println(obj.x);
      }
    }
```

Quick Check 5.11 *What will be the output of the following code?*

```
    class A {
      void display() { System.out.println("A"); }
    }

    class B extends A {
      void display() {
        super.display();
        System.out.println("B");
      }
    }

    public class Test {
      public static void main(String[] args) {
        A obj = new B();
        obj.display();
      }
    }
```

Quick Check 5.12 *Overloading vs Overriding*

What will be the output of the following code? Why?

```
class A {
  void show(int x) { System.out.println("A int"); }
}

```

```
class B extends A {
  void show(double x) { System.out.println("B double"); }
}

public class Test {
  public static void main(String[] args) {
    A obj = new B();
    obj.show(10);
  }
}
```

5.7 Activities

Exercise 5.1 *Resuming Exercise 4.2. Check whether the statements in the following code are valid and justify your answers:*

```

    Pet pet1 = new Dog(); Pet pet2 = new RedFish();   Pet pet3 = new Chamellon();
    Dog myDog = new Pet();

    Pet p;
    p = new Dog(); p.speak();
    p = new ServiceDog(); p.speak();

    Pet p4;  p4 = new RedFish( );   System.out.println(p4.Hungry());

    Pet p5; p5 = new Dog( ); System.out.println( ((RedFish)p5).Hungry( ) );
```

Exercise 5.2 *Resuming Exercise 4.2.*

① *Add a class named yellowSnake inherited from the class pet and set the color of the YellowSnake to be always Yellow (whatever, the color must be Yellow).*

② *Create a Box class that is characterized by a color and contains a Pet. The Box class has two methods: addPets to place a Pet in the Box and display to display the color of the box and*

the Pet in the box, as well as the type of Pet (dog, chameleon, ...).

③ *In a main program, create an array-list (or a table) of Pets. Each element of the list must be Dog, RedFish, Chameleon, or yellowSnake. To fill the list (array), generate a random number from 1 to 4. If the generated number = 1, add a Dog, 2 add a RedFish, 3 add a Chameleon, 4 add a yellowSnake.*

(a). Display each element of the array-list

(b). Display the number of yellowSnake, dogs, RedFish and chameleons in the list.

(c). Create an array of boxes, fill it and display its elements.

Exercise 5.3 *Write a program that creates an ArrayList of pets. An item in the list is either a Dog or a yellowSnake. For each pet, enter its name and type ('c' for cat and 'd' for dog). Stop the input when the string STOP is entered for the name. After the input is complete, output the name and the type of each pet in the list.*

Exercise 5.4 *Repeat Exercise 5.3, but this time group the output by printing out the names of all dogs first and then the names of all chameleons.*

Exercise 5.5 ***Mini-project: Multimedia Library***

Imagine a small-town library named "Maplewood Stacks". For decades, they used paper cards tucked into the back of books to track their inventory. However, as they've expanded to offer local indie films on DVD and jazz albums on CD, the paper system is failing. They need a Java-based digital system. The librarian wants the system to treat "everything as an item" while still remembering that a book has an author, but a movie has a director. Here is the technical breakdown of how to build this using OOP with Java:

① *The Class* ***LibraryItem****: Every object in the library shares basic characteristics:*

- ***Fields****: private String itemId (e.g., "LIB-101"), private String title, protected boolean isAvailable (Defaults to true)*
- ***Methods*** *to implement:*
  - *checkOut(): Should check if isAvailable is true. If yes, set it to false. If no, print an error or " Already Loaned".*
  - *returnItem(): Sets isAvailable back to true.*

- *displayStatus(): Prints the ID, Title, and whether it is "In Library" or "Loaned Out."*

② *Class **Book**:*

- ***Fields**: String author, int pageCount, edition, year*
- ***Methods** to implement: overrides displayStatus() to also mention the author and the other information.*

③ *Class **MediaItem**: CDs and DVDs can be grouped together. You might create a MediaItem class that adds a duration field and create:*

- *Subclass **DVD**: Inherits from MediaItem; adds String resolution (4K, HD). This class overrides displayStatus() to also mention the author and the other information.*
- *Subclass **CD**: Inherits from MediaItem; adds String artist. This class overrides displayStatus() to also mention the author and the other information.*

④ ***EBook** extends Book: Inherits author from Book and itemId from LibraryItem. Note: You might decide EBooks are always available since they are digital.*

⑤ *The true power of this system is that the librarian can handle a list of items without knowing exactly what they are at first:*

- *Create an array or list of LibraryItem.*
- *Inside that array, mix a Book, a CD, and a DVD.*
- *The Polymorphic Loop: Iterate through the array and call displayStatus() on each.*

⑥ *In your Main class, create a CompactDisc object. Call checkOut() twice. The second time should display an "Already Loaned" message.*

⑦ *Store Books and CDs in a LibraryItem array. Loop through it and print only the items where isAvailable is true.*