Polymorphism

Objective:

After completing this chapter, you will be able to:

Explain overloading and overriding

Describe polymorphism

Explain abstract classes

Explain static and dynamic polymorphism

Polymorphism

Introduction:

Polymorphism allows a software developer to reuse names as well as code,thus making software development, as well as its maintenance and use, more natural.

Java exhibits polymorphism in two forms, static and dynamic, through overloading and overriding.

Static Polymorphism

Static polymorphism is more efficient and reliable.

- Java exhibits static polymorphism through overloading.
- Association of a method with the object is resolved at compile time.
- For an example, There are two overloaded methods:

```
public void overload(int i);
public void overload(string s);
```

– When a user uses `overload(10)`, the binding happen with the overload method that accepts integer argument.

– When a user uses `overload("Java")`, the binding happen with the overload method which accepts `String` argument.

Method overloading:

– Method overloading occurs when two or more methods of a class have the same name but differ by:

- The number of arguments
- The data types of the arguments
- The sequence of the arguments

– Methods return type is not considered for overloading

– Method overloading allows the use of the same name for a group of methods that basically have the same purpose

– The `println()` method is a good example of overloading

Dynamic polymorphism

Dynamic polymorphism is very powerful and provides a high degree of flexibility.

- An important property of dynamic polymorphism is:

– The decision of executing the a particular version of a method depends on the actual type of object, whose reference is stored in the reference variable,

and not on the type of the reference variable on which the method is invoked.

– The decision is dynamic because the compiler has no way of knowing (at the compilation time) the actual type of the object whose reference will be stored in the reference variable

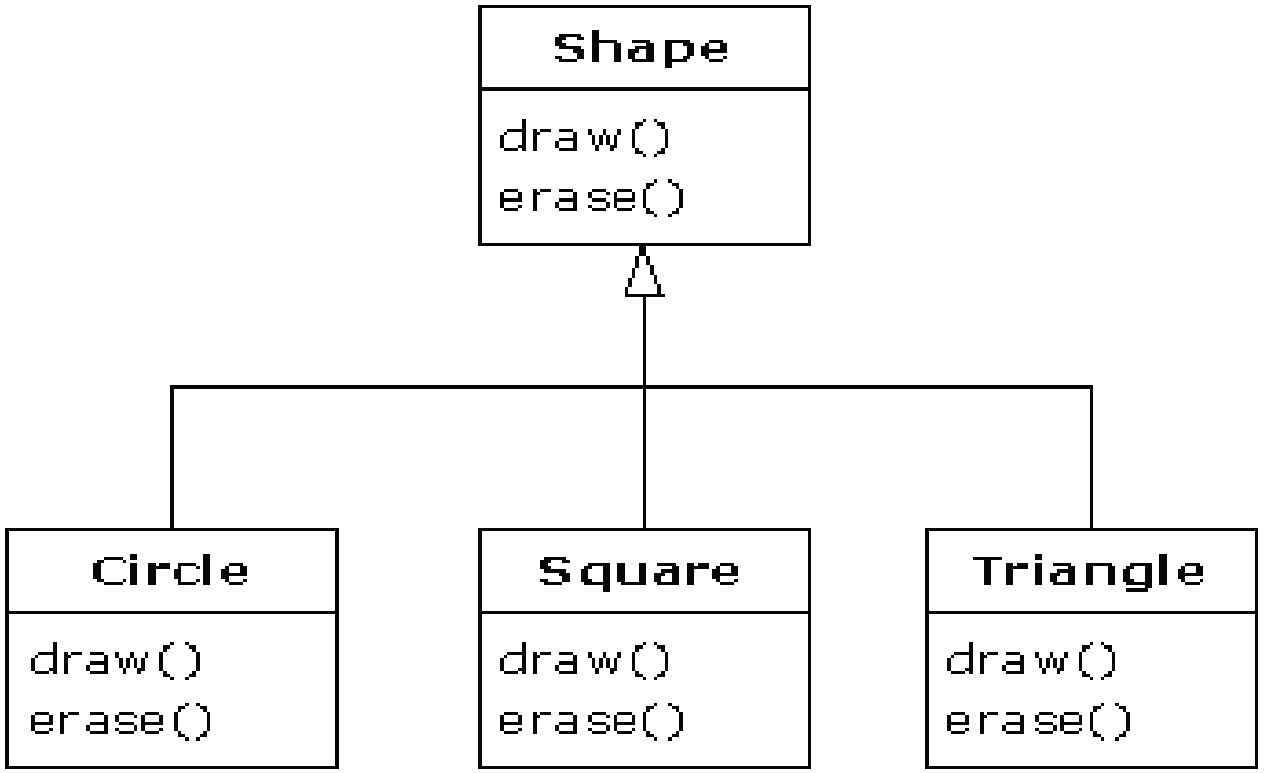
Method overriding:

– A class may redefine some or all of the features that it inherits from its

parents.

– Java exhibits dynamic polymorphism through overriding.

Study


Shape
draw()
erase()
Circle
draw()
erase()
Square
draw()
erase()
Triangle
draw()
erase()

```
Class ShapeDemo{
  public static void printShape(Shape s)
  {
      s.draw();
      s.erase();
  }
}
//generic code to call the different
  implementations of Shape.
```

Abstract Class

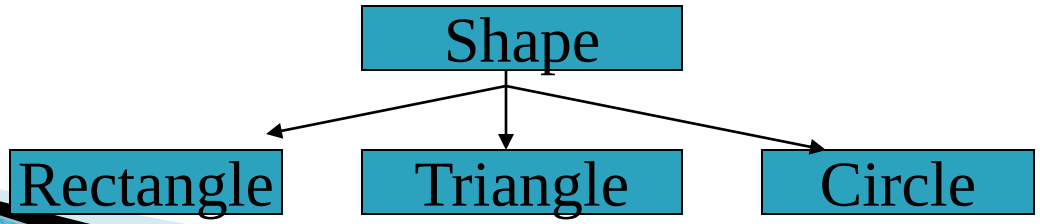
A class that represents an abstract concept as called an abstract class.

- Super classes that act purely as templates for more usable subclasses.
- It serves as nothing more than an abstraction for the common class functionality shared by the subclasses.
- An abstract classes cannot be instantiated because:
  - Parts of it have been specifically left unimplemented
  - More specifically, these parts are made up of methods, called abstract methods, which are yet to be implemented
- An abstract class is a class with one or more abstract methods.
- You can define a class as abstract, even if it contains all concrete methods.

Abstract Class (Contd.)

Abstract methods:

- An abstract method is a method that has no implementation.
- For example, a Shape class is defined to specify the common properties and behavior of its subclasses (Rectangle, Triangle, and Square). The `findArea` method of the shape should be left unimplemented because shape is an abstract concept. But the subclasses must provide the implementation for the method as they are really existing objects



Demo

```
/*
* Shape.java
*/
public abstract class Shape {
public abstract void area();
}
```

Demo contd…

```
/*
* Triangle.java
*/
public class Triangle extends Shape{
public void area{
System.out.println("Area of a
  triangle");
}
}
```

Demo contd…

```
/*
* Rectangle.java
*/
public class Rectangle extends Shape{
public void area{
System.out.println("Area of a
  Rectangle");
}
}
```

Factory pattern

```
public class Factory{
                        static Shape ref;
public static Shape getInstance(int ch)
{
        if(ch==1) ref=new Rectangle();
        else if(ch==2) ref=new Triangle();
   return ref;
}
```

Demo contd…

```
/*
* Tester.java
*/
public class Tester{
public static void main(String[] ar){
Shape base = null;
int choice=Integer.parseInt(args[0]);

     base=Factory.getInstance(choice);
          base.area();
}
}
```

Advantage of factory pattern

Suppose another implementation of findArea() has to be added in a class Circle then a class file is created independently.

And changes are made in the Factory class not in the Client.

Helps in creating loosely coupled components.

Features of an abstract class

- Contains zero or more abstract methods
- A child class of an abstract class can be marked abstract,if abstract methods not implemented.
- Cannot be instantiated
- Can contain concrete methods
- Supports dynamic binding,extensibility.