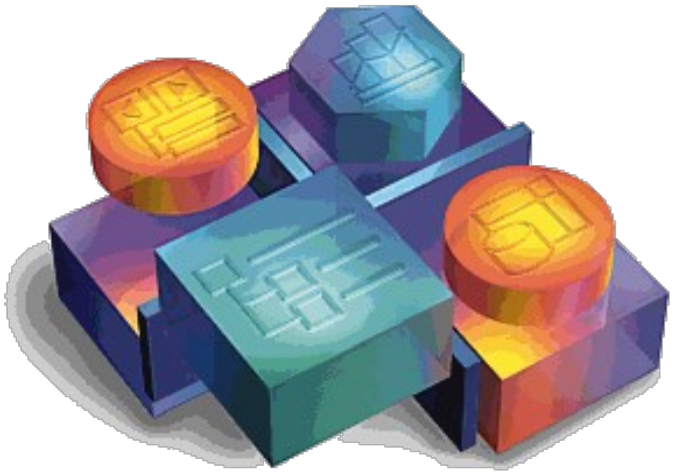
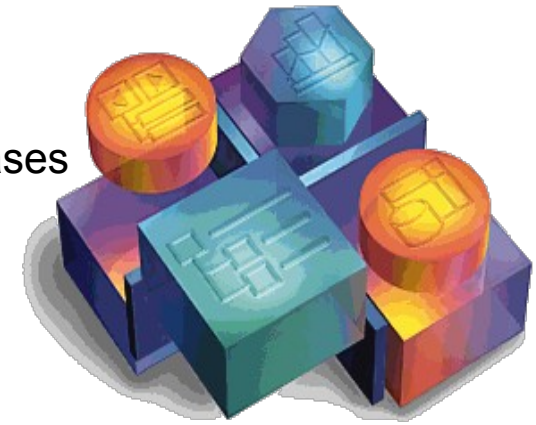
# Object-Oriented Programming Design



1. Introduction to Object-Oriented Concepts
2. How to think in Terms of Objects
3. Advanced Object-Oriented Concepts
4. The anatomy of a Class
5. Class Design guidelines
6. Designing with objects
7. Mastering Inheritance and Composition

**8. Frameworks and Reuse: Designing with interfaces and Abstract classes**

9. Building objects
10. Creating Object Models with UML
11. Objects and Portable Data: XML
12. Persistent Objects: Serialization and Relational Databases
13. Objects and the Internet
14. Objects and Client/Server Applications
15. Design Patterns



# 8. Frameworks and Reuse: Designing with interfaces and Abstract classes

**Framework:** 使开发更具工程性、简便性和稳定性

**reuse:** 重用

**What is a framework?**

Hand-in-hand with the concept of **code reuse** is the concept of standardization., wich is sometimes called **plug-and-play**.

Often, an API is referred to as a framework

API(Application Programming Interface)

# 8. Frameworks and Reuse: Designing with interfaces and Abstract classes

**An example: the dialog box**

►If you need a dialog box, why write code to create a new dialog box when one already exists and has been tested?

# 8. Frameworks and Reuse: Designing with interfaces and Abstract classes

**If you need a dialog box, how do you use the dialog box provided by the framework?**

►You follow the rules that the framework provides you.

**And where might you find these rules?**

►The rules for the framework are found in the documentation. The person or persons who wrote the class or classes should have provided documentation on how tu use the public interfaces of the class or classes.

# 8. Frameworks and Reuse: Designing with interfaces and Abstract classes

**What is a contract?**

►In the context of this chapter, we will consider a **contract** to be any mechanism that **requires** a developer to **comply** with the **specification** of an API.

# 8. Frameworks and Reuse: Designing with interfaces and Abstract classes

**What is a contract?**

- **In Java and .NET languages, the two ways to implement contracts are to use abstract classes and interfaces.**

# 8. Frameworks and Reuse: Designing with interfaces and Abstract classes

## Abstract Classes

- One way a contract is implemented is via an **abstract class**. An **abstract class** is a class that contains one or more methods that do not have any implementation provided.

# 8. Frameworks and Reuse: Designing with interfaces and Abstract classes

**Abstract Classes, an example:**

- Suppose that you have an abstract class called Shape. It is abstract because you cannot instantiate it. If you ask someone to draw a shape, the first thing they will ask you is «What kind of shape? »
- Thus, the concept of a shape is abstract. However, if someone asks you to draw a circle, this does not pose quite the same problem because a circle is a concrete concept. You know what a circle looks like.

# 8. Frameworks and Reuse: Designing with interfaces and Abstract classes

**Abstract Classes, an example:**

- Let's assume that we want to create an application to draw shapes. Or goal is to draw every kind of shape represented in our current design, as well as ones that might be added later.

# 8. Frameworks and Reuse: Designing with interfaces and Abstract classes

**Abstract Classes, an example:**

**There are two conditions we must adhere to.**

**1)** First, we want all shapes to use the same syntax to draw themselves. For example, we want every shape implemented on our system to contain a method called draw().

# 8. Frameworks and Reuse: Designing with interfaces and Abstract classes

**Abstract Classes, an example:**

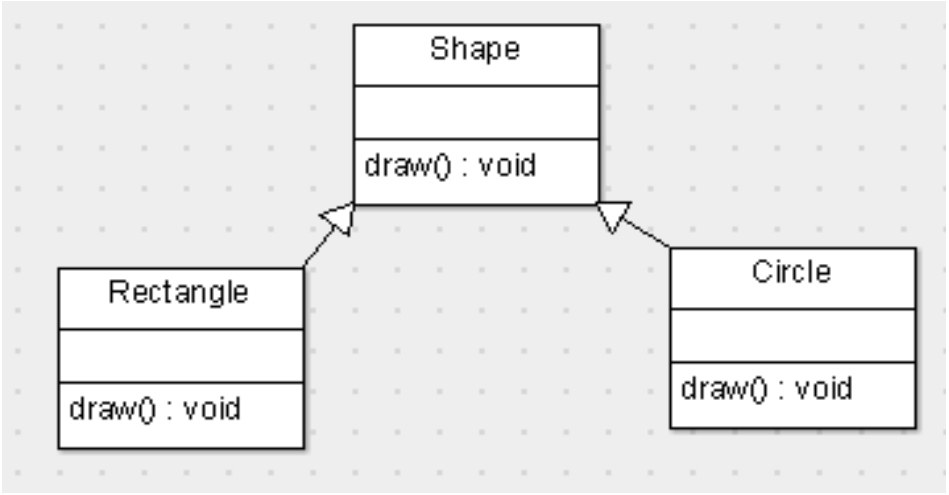
**There are two conditions we must adhere to.**

**2)** Second, remember that is important that every class be responsible for its own actions. Here a class is required to provide a method called «draw » and  must provide its own implementation of the code. For example, the class Circle and the class Rectangle both have a « draw » method.

# 8. Frameworks and Reuse: Designing with interfaces and Abstract classes

**Abstract Classes, an example:**

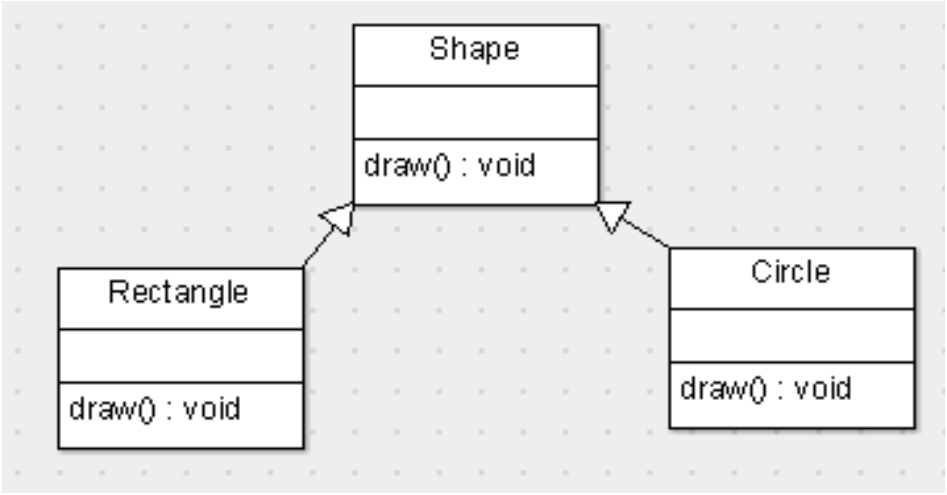
►When we create classes called Circle and Rectangle, wich are subclasss of Shape, these classes must implement their own version of Draw.



# 8. Frameworks and Reuse: Designing with interfaces and Abstract classes

**Abstract Classes, an example:**

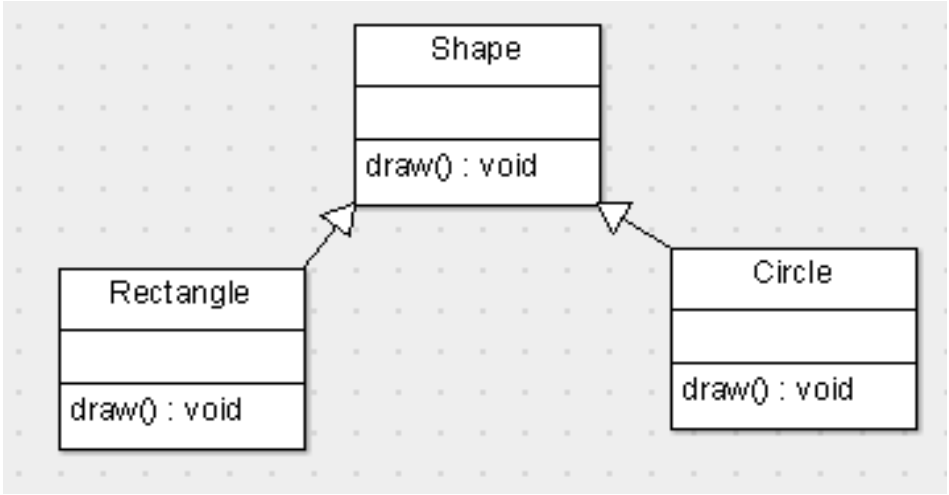
►In this way, we have a **Shape framework** that is truly **polymorphic**




```
circle.draw() // draws a cricle
rectangle.draw() // draws a rectangle
```

# 8. Frameworks and Reuse: Designing with interfaces and Abstract classes

## Abstract Classes, an example:

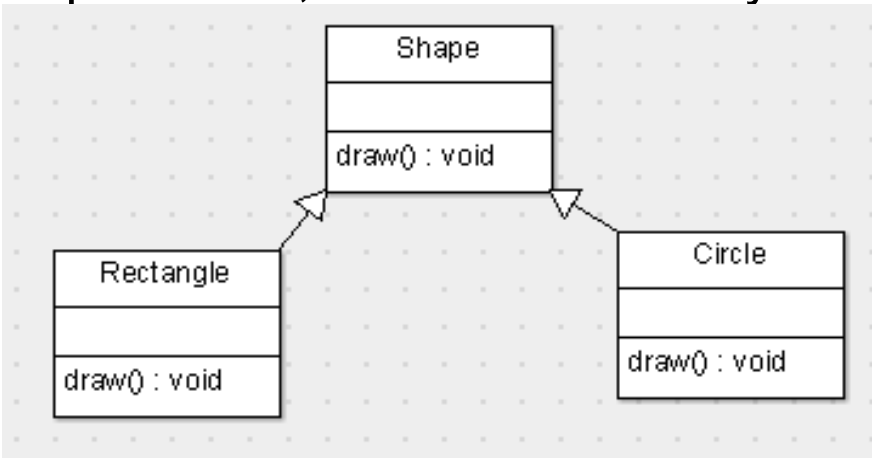



```
public abstract class Shape {
    public abstract void draw();
}

public class Circle extends Shape {

    @Override
    public void draw() {
        System.out.println("Circle");
    }
}

public class Rectangle extends Shape{

    @Override
    public void draw() {
        // TODO Auto-generated method stub
        System.out.println("Rectangle");
    }

}
```

# 8. Frameworks and Reuse: Designing with interfaces and Abstract classes

**Abstract Classes, an example:**

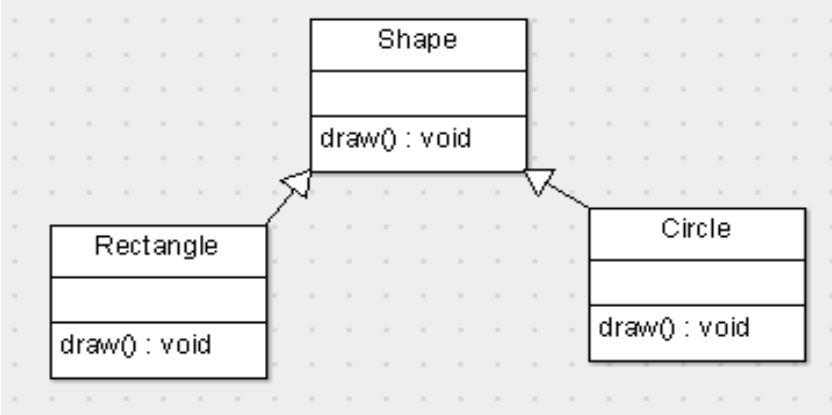
- Note that both Circle and Rectangle extend (that is, inherit from) Shape. They provide the implementation. Here is where the contract comes in.
- If Circle inherit from shape and fails to provide a draw() method, Circle won't even compile. Thus, Circle fail to satisfy the contract with Shape.



# 8. Frameworks and Reuse: Designing with interfaces and Abstract classes

**Abstract Classes, an example:**

►A project manager can require that programmers creating shapes for the application must inherit from Shape. By doing this, all shapes in the application will have a draw() method.



# 8. Frameworks and Reuse: Designing with interfaces and Abstract classes

## Interfaces

Interface Terms

►This is another one of those times when software development terminology gets confusing. The term ***interface*** used in earlier chapters is a term generic to OO development and refers to the public interface to a class. The term interface used in this context refers to a syntactical language construct that is specific to programming language. It is important not to get the two terms confused.

# 8. Frameworks and Reuse: Designing with interfaces and Abstract classes

## Interfaces

- Some languages, such as C++, use only abstract classes to implement contracts; however Java and .Net have another mechanism that implements a contract called an **interface**.

- C++ does not have a construct called an interface. For C++, an abstract class provides the functionality of an interface.

# 8. Frameworks and Reuse: Designing with interfaces and Abstract classes

## Interfaces

- The obvious question is this: If an abstract class can provide the same functionality as an interface, why do Java / .NET provide this construct called an **interface**?

# 8. Frameworks and Reuse: Designing with interfaces and Abstract classes

## Interfaces

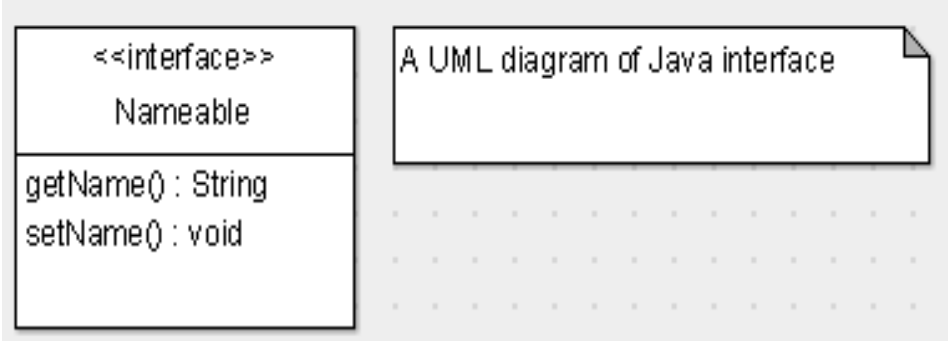
- C++ supports multiple inheritance, whereas Java and .NET do not.

  Because of these considerations, interfaces are often thought to be a workaround for the lack of multiple inheritance. This is not technically true. Interface are a seperate design technique, and although they can be used to design applications that could be done with multiple inheritance, they do not replace multiple inheritance.

# 8. Frameworks and Reuse: Designing with interfaces and Abstract classes

## Interfaces

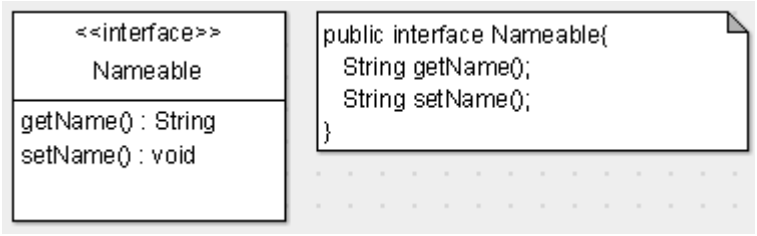
►As with abstract classes, interfaces are a powerful way to enforce contracts for a framework.



# 8. Frameworks and Reuse: Designing with interfaces and Abstract classes

## Interfaces

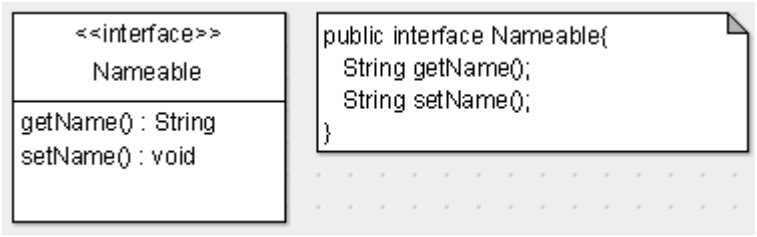
- In the code, notice that « Nameable » is not declared as a class. But as an interface. Because of this, both methods, getName() and setName(), are considered abstract and there is no implementation provided. An interface, unlike an abstract class, can provide **no** implementation at all.



# 8. Frameworks and Reuse: Designing with interfaces and Abstract classes

## Interfaces

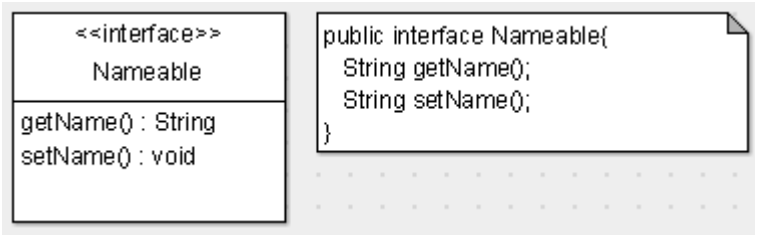
►As a result, any class that implements an interface must provide the implementation for all methods. For example, in Java, a class inherits from an abstract class, whereas a class implements an interface.



# 8. Frameworks and Reuse: Designing with interfaces and Abstract classes

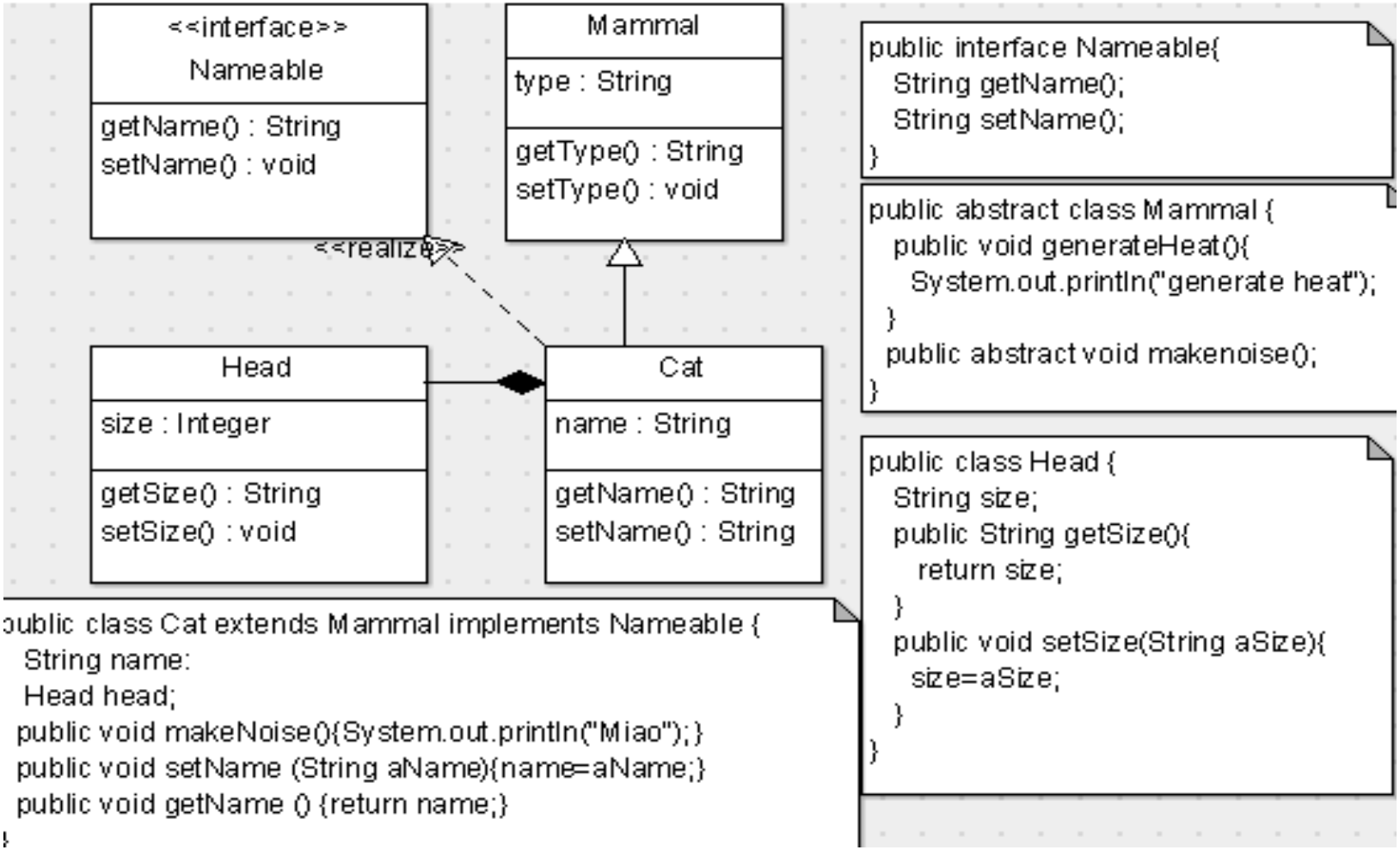
## Interfaces

- ►An abstract class can provide abstract and concrete methods.
- ►An interface provides only abstract methods.



# 8. Frameworks and Reuse: Designing with interfaces and Abstract classes

## Tying it all together (Abstract class and Interfaces)



# 8. Frameworks and Reuse: Designing with interfaces and Abstract classes

**Tying it all together (Abstract class and Interfaces)**

**You should be familiar with the following concepts:**

- ►Cat is mammal, so the relationship is **inheritance**
- ►Cat implements Nameable, so the relationship is an **Interface**.
- ►Cat has a head, so the relationship is **composition**.

1. Introduction to Object-Oriented Concepts
2. How to think in Terms of Objects
3. Advanced Object-Oriented Concepts
4. The anatomy of a Class
5. Class Design guidelines
6. Designing with objects
7. Mastering Inheritance and Composition
8. Frameworks and Reuse: Designing with interfaces and Abstract classes

# 9. Building objects

10. Creating Object Models with UML
11. Objects and Portable Data: XML
12. Persistent Objects: Serialization and Relational Databases
13. Objects and the Internet
14. Objects and Client/Server Applications
15. Design Patterns