Object Oriented Programming

LESSON 07

Inheritance and Polymorphism (Part 2)

Outline

1. What is Interface?
2. Declaring Interface
3. Realizing Interface
4. Interface and Polymorphism
5. Inheriting Interface Overriding method and attributes

Overview

In this chapter, you are going to learn about

- Know interface
- Know how to create interface
- Know how to use interface
- Know how to use interface with polymorphism
- Know how to implement and override method and attributes of interface

Learning content

1. What is Interface?
   - Introductory example
   - Interface definition
   - Interface in UML
2. Declaring Interface
   - Syntax in Java
   - Compilation in Java
   - Declaration
3. Realizing Interface
   - Realizing Interface in UML
   - Realizing Interface in Java
   - Realizing multiple interfaces
4. Interface and Polymorphism
   - Example using interface as a type
   - Using Interface to group classes
   - Using Interface as future features
5. Inheriting Interface
   - Inheritance of interface
   - Inheritance multiple interfaces
   - Advantages of Interface

Pre-Test

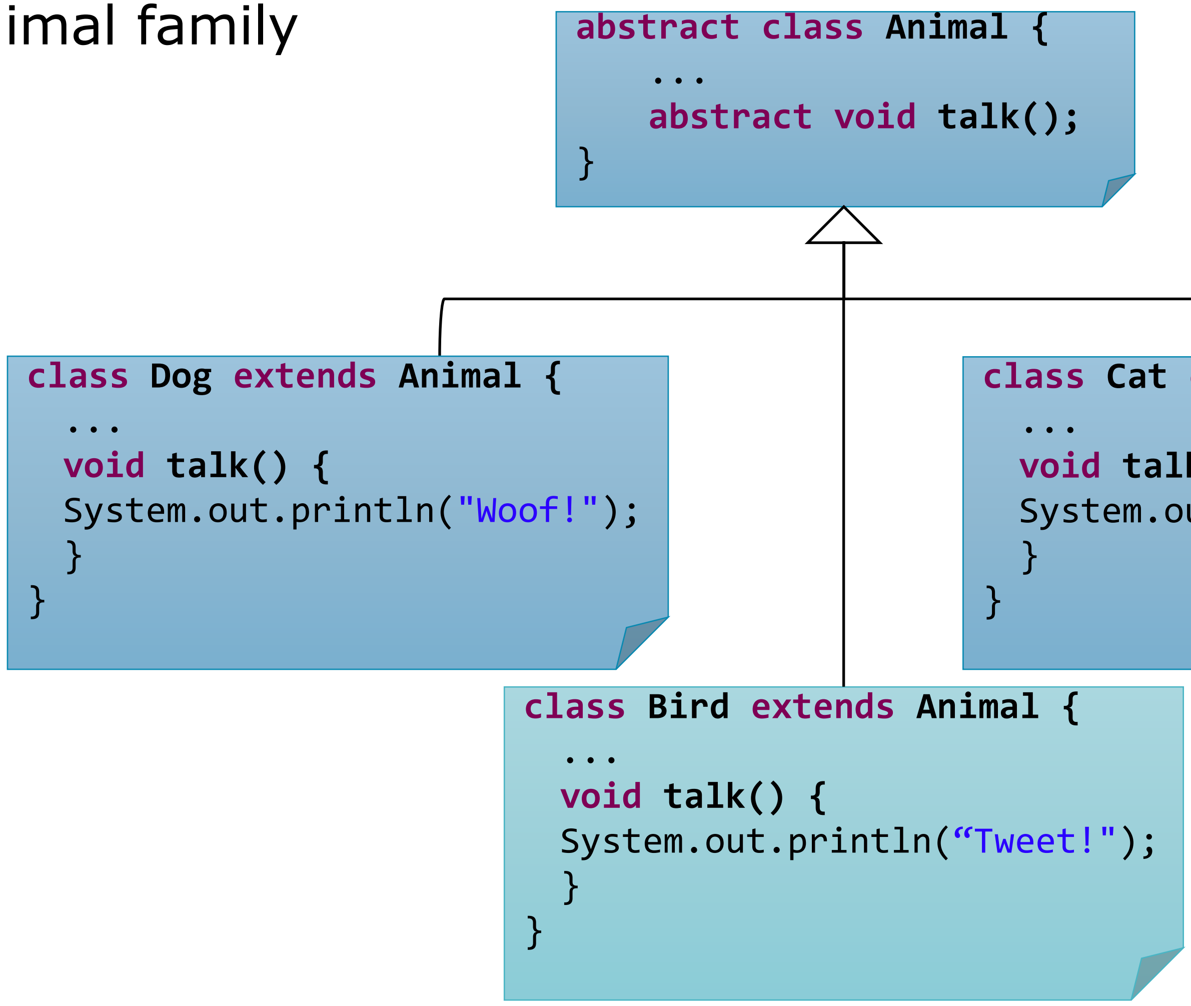
Question	Possible answers	Correct Answer	Question Feedback
What are things that role of Doctor do at hospital?	a) Reading books b) Register patients c) Examine patients d) Cue patients	c) Examine patients d) Cue patients	Doctor can read books but not his role now. Doctor can Register patients, but not his role now.
Which role can do hair cut?	a) Actress b) Book keeper c) Hairdresser d) Any personal	c) Hairdresser	Others may be able to do hair cut, but not their roles.
Which service can consider as waiter/waitress?	a) Clean the table b) Clean the floor c) Bring the menu d) Total money	c) Bring the menu a) Clean the table	Total money is role of Cashier; Clean the floor is role of cleaner.

1. What is Interface?

- A Java interface is a bit like a class, except a Java interface can only contain:
  - Method signatures (name, parameters and exceptions)
  - Fields.
- And Java interface cannot contain
  - Implementation of the methods
  - Instance variables.

1.1. Introductory example

- Animal family



```
abstract class Animal {
    ...
    abstract void talk();
}
```

```
class Dog extends Animal {
  ...
  void talk() {
  System.out.println("Woof!");
  }
}
```

```
class Cat extends Animal {
  ...
  void talk() {
  System.out.println(“Meow!");
  }
}
```

```
class Bird extends Animal {
  ...
  void talk() {
  System.out.println(“Tweet!");
  }
}
```

1.1. Introductory example

Polymorphism means that a reference type (class) given may designate an object of any class and in accordance with nature of this item produce different behavior

```
Animal animal =  new Dog();
...
animal = new Cat();
```

animal can be a dog, a cat or any subclass of **Animal**

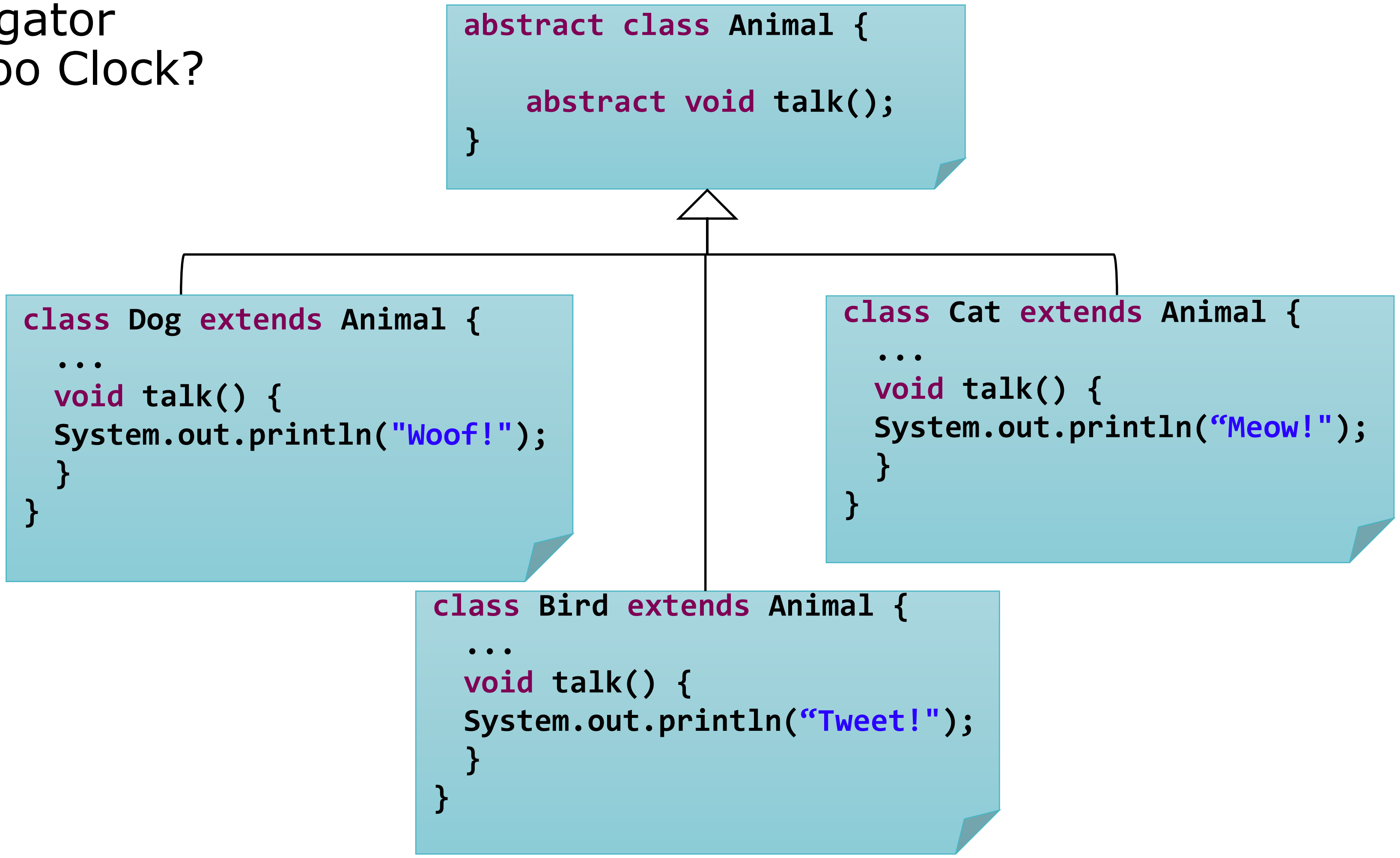
JAVA polymorphism is made possible by dynamic binding

```
class Interrogator {

   static void makeItTalk(Animal subject) {
      subject.talk();
   }
}
```

JVM decides at runtime which method to invoke based on the class of the object

1.1. Introductory example

- How to use Interrogator also talk to a Cuckoo Clock?



```
abstract class Animal {

    abstract void talk();
}
```

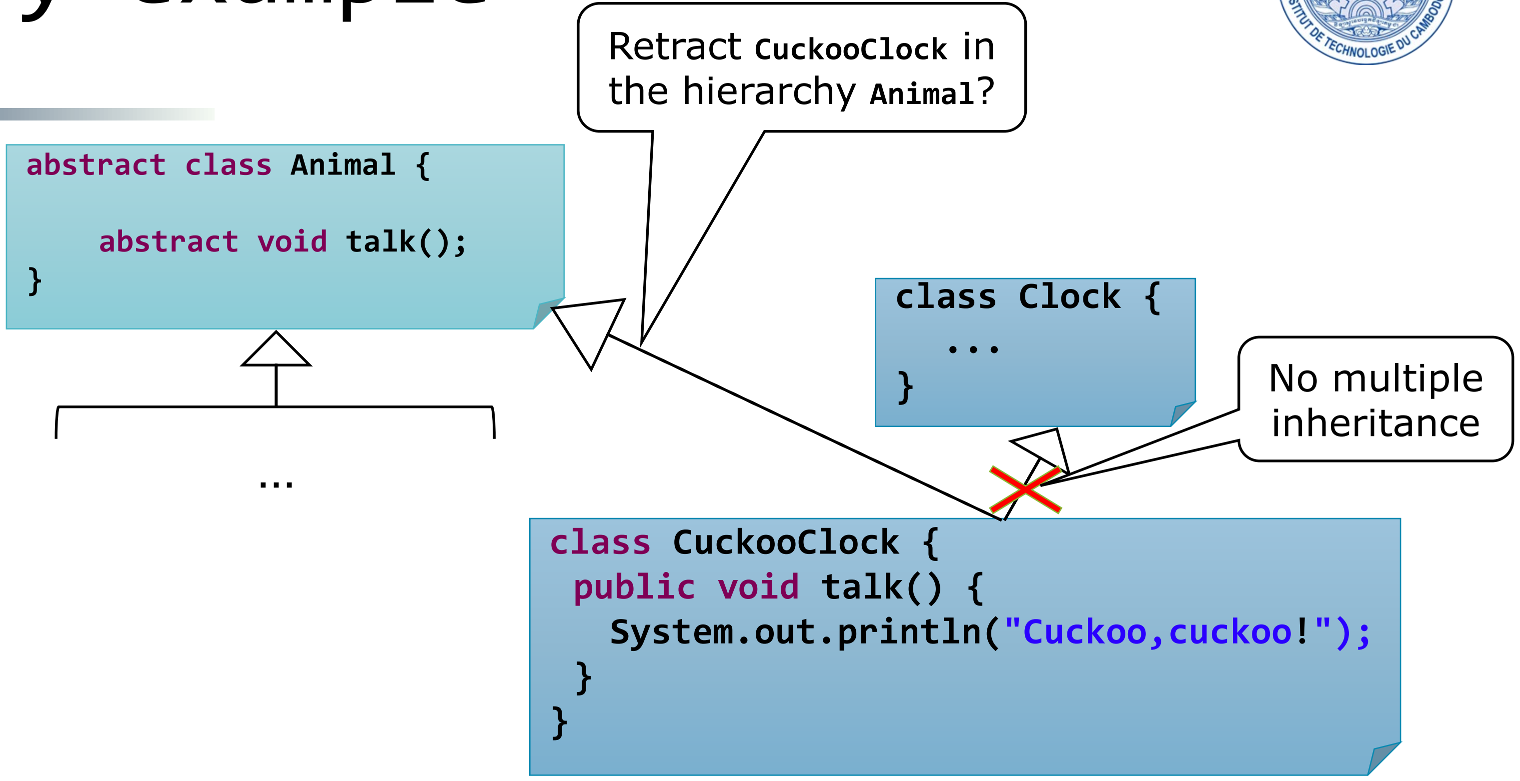
```
class Dog extends Animal {
  ...
  void talk() {
  System.out.println("Woof!");
  }
}
```

```
class Cat extends Animal {
  ...
  void talk() {
  System.out.println(“Meow!");
  }
}
```

```
class Bird extends Animal {
  ...
  void talk() {
  System.out.println(“Tweet!");
  }
}
```

1.1. Introductory example

- How to use Interrogator also talk to a Cuckoo Clock?



```java
abstract class Animal {

    abstract void talk();
}
```

```
...
```

```java
class Clock {
  ...
}
```

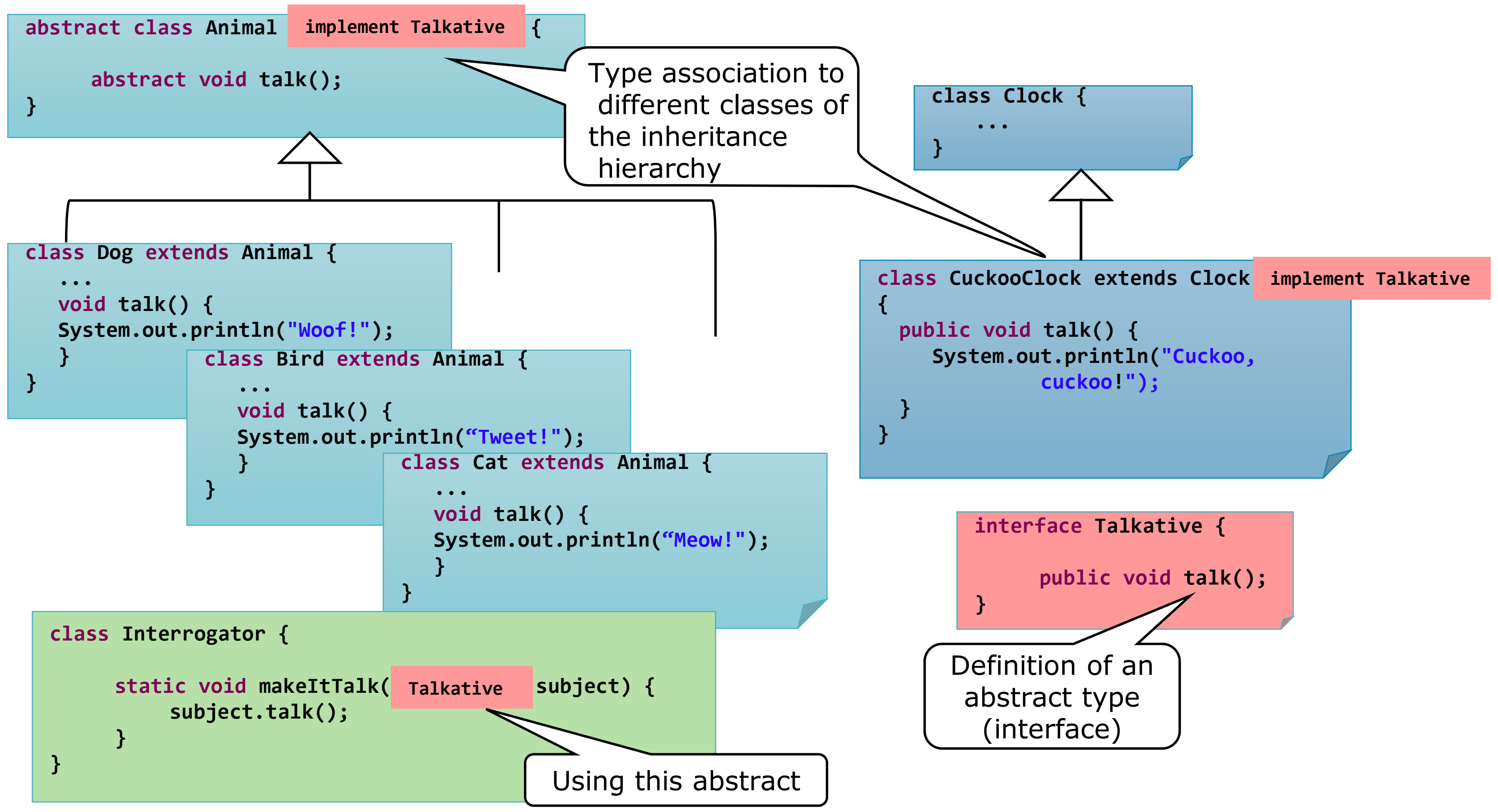
```java
class CuckooClock {
 public void talk() {
   System.out.println("Cuckoo,cuckoo!");
 }
}
```

```java
class Interrogator {

    static void makeItTalk(Animal subject)
    {
        subject.talk();
    }
}
```

```java
class CuckooClockInterrogator {

    static void makeItTalk(CuckooClock subject)
    {
        subject.talk();
    }
}
```

Polymorphism happen?

1.1. Introductory example



```
abstract class Animal implement Talkative {

    abstract void talk();
}
```

```
class Dog extends Animal {
   ...
   void talk() {
   System.out.println("Woof!");
   }
}
```

```
class Bird extends Animal {
   ...
   void talk() {
   System.out.println("Tweet!");
   }
}
```

```
class Cat extends Animal {
   ...
   void talk() {
   System.out.println("Meow!");
   }
}
```

```
class Clock {
    ...
}
```

```
class CuckooClock extends Clock implement Talkative
{
  public void talk() {
     System.out.println("Cuckoo,
               cuckoo!");
  }
}
```

```
interface Talkative {

      public void talk();
}
```

```
class Interrogator {

      static void makeItTalk( Talkative subject) {
           subject.talk();
      }
}
```

Type association to different classes of the inheritance hierarchy
Definition of an abstract type (interface)
Using this abstract

- The interfaces allowing more polymorphism with interfaces because it is not necessary to fit everything into a single family (hierarchy) class

1.2. Interface definition

- Interfaces

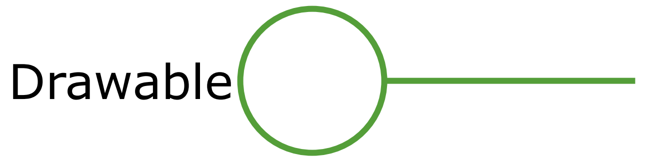
  - Java's interface gives you more polymorphism than you can get with singly inherited families of classes, without the "burden" of multiple inheritance of implementation.

  Bill Venners Designing with Interfaces - One Programmer's Struggle to

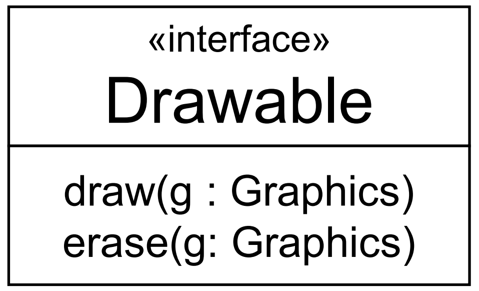
  Understand the Interface http://www.atrima.com/designtechniques/index.html

1.3. Interface in UML

- In UML we can represent Interface in 2 notations:
  - Ball notation
  - Class notation



OR



2. Declaring Interface

- Java Interface is declared in file with extension .java, just like class.
    - The name of interface follows the rule as class name
        - Start with letter a-zA-Z_
        - Then can be numbers or letters (a-zA-Z0-9_)
- The fields in interface are implicitly *static final*
- The methods in interface are implicitly *public*
- The methods in interface are *abstract*

2.1. Syntax in Java

- Syntax of declaration of interface

```
public interface NameOfInterface {
  //Any number of final, static fields
  //Any number of abstract method declarations\
}
```

- Example

```
/* File name : Animal.java */
interface Animal {
  public void eat();
  public void travel();
}
```

2.2. Compilation in Java

- Just like class compilation:

$$\begin{tikzcd}\boxed{\text{javac Animal.java}} \arrow[r] & \boxed{\text{Animal.class}}\end{tikzcd}$$

- Content of Animal.java

```java
/* File name : Animal.java */
interface Animal {
  public void eat();
  public void travel();
}
```

2.3. Declaring Interface

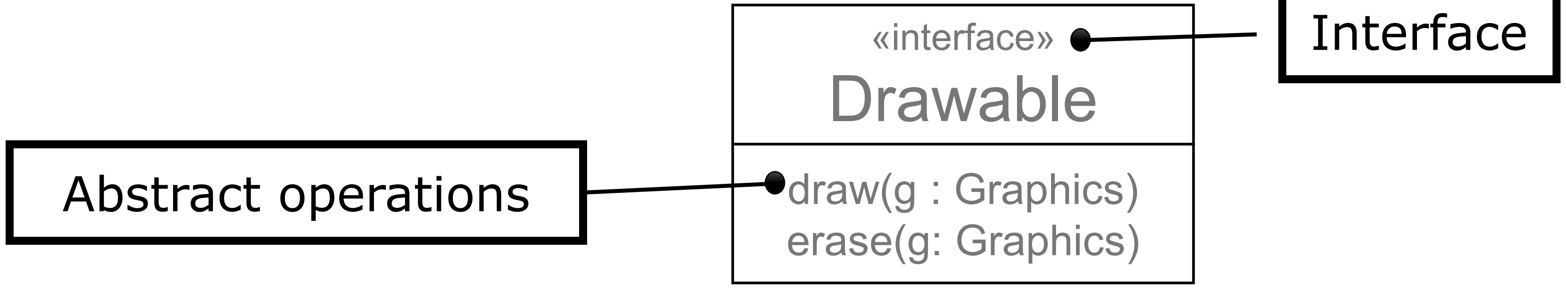
- An interface is a collection of operations used to specify a service of a class.
- An interface can be seen as an abstract class without attributes which all operations are abstract.

Drawable.java

```java
import java.awt.*;
public interface Drawable {

  public void draw(Graphics g);
  void erase(Graphics g);
}
```

- All methods are abstract
- They are implicitly public

- Ability to define attributes provided that they are of primitive type attributes
- These attributes are implicitly declared as **`static final`**

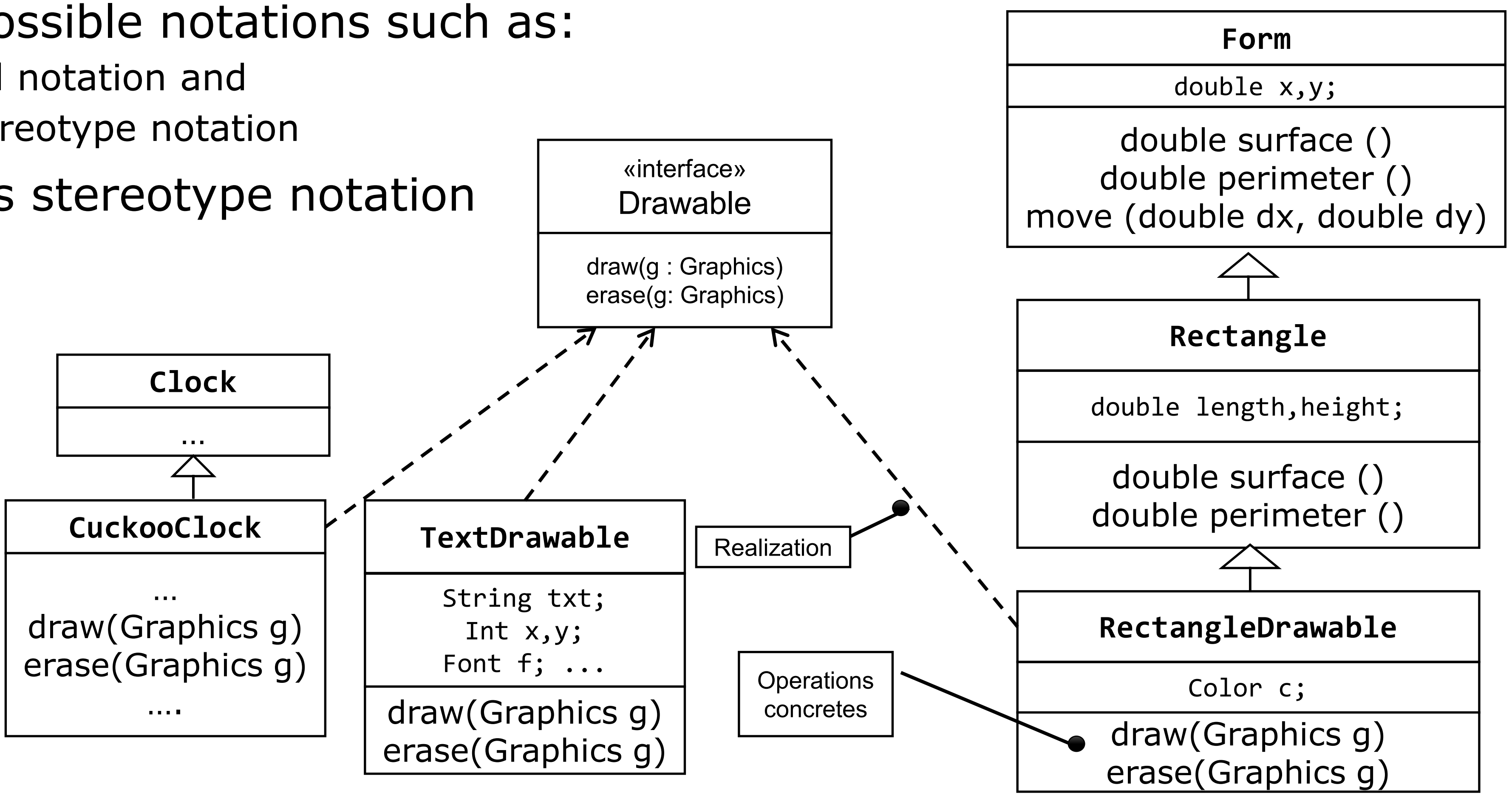


3. Realizing Interface

- An interface is intended to be "realized" (implemented) by other classes (they inherits all descriptions and make concrete the abstract operations).
  - Realized classes commits to provide the service specified by the interface
- The implementation of an interface is free.

3.1. Realizing Interface in UML

- Two possible notations such as:
  - Ball notation and
  - Stereotype notation
- Here is stereotype notation



3.2. Realizing Interface in Java

- In the same way that a class extends its superclass can optionally implement one or more interfaces
  - in the definition of the class, after the clause **extends nameSuperClass**, make explicit the **implements** keyword followed by the name of implemented interface

```
class RectangleDrawable extends Rectangle implements Drawable {

    public void draw(Graphics g){
      g.drawRect((int) x, (int) y, (int) length, (int) height);
    }

    public void erase(Graphics g){
      g.clearRect((int) x, (int) y, (int) length, (int) height);
    }

}
```

  - if the class is a concrete class must provide an implementation (a body) to each of the abstract methods defined in the interface (which must be declared public)

3.3. Realizing multiple interfaces

- A Java class can implement multiple interfaces simultaneously
  - To do this, list names of the interfaces to implement separated by commas should follow the implements keyword

Methods of Drawable interface

Comparable interface methods

```java
class RectangleDrawable extends Rectangle implements Drawable, Comparable {

    public void draw(Graphics g){
      g.drawRect((int) x, (int) y, (int) length, (int) height);
    }
    public void erase(Graphics g){
      g.clearRect((int) x, (int) y, (int) length, (int) height);
    }

    public int compareTo(Object o) {
        if (o instanceof Rectangle)
           ...
    }
}
```

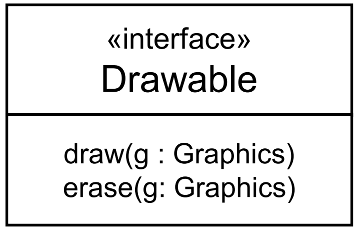
4. Interface and Polymorphism

- An interface can be used as a type
  - A variable (references) whose type is an interface it is possible to allocate instances of any class that implements the interface or subclass of such a class.
- Polymorphism rules apply in the same way as for classes:
  - Static verification code
  - Dynamic Link

4.1. Example using interface as a type

- Drawable Interface has only 2 methods:
  - so when using an object as Drawable, we can use only 2 methods: draw() and erase()

```
public class Window {
  private nbrFigures;
  private Drawable[] figures;
  ...
  public void add(Drawable d){
    ...
  }
  public void delete(Drawable o){
    ...
  }

  public void draw() {
     for (int i = 0; i < nbrFigures;
     i++)
        figures[i].draw(g);
  }
}
```

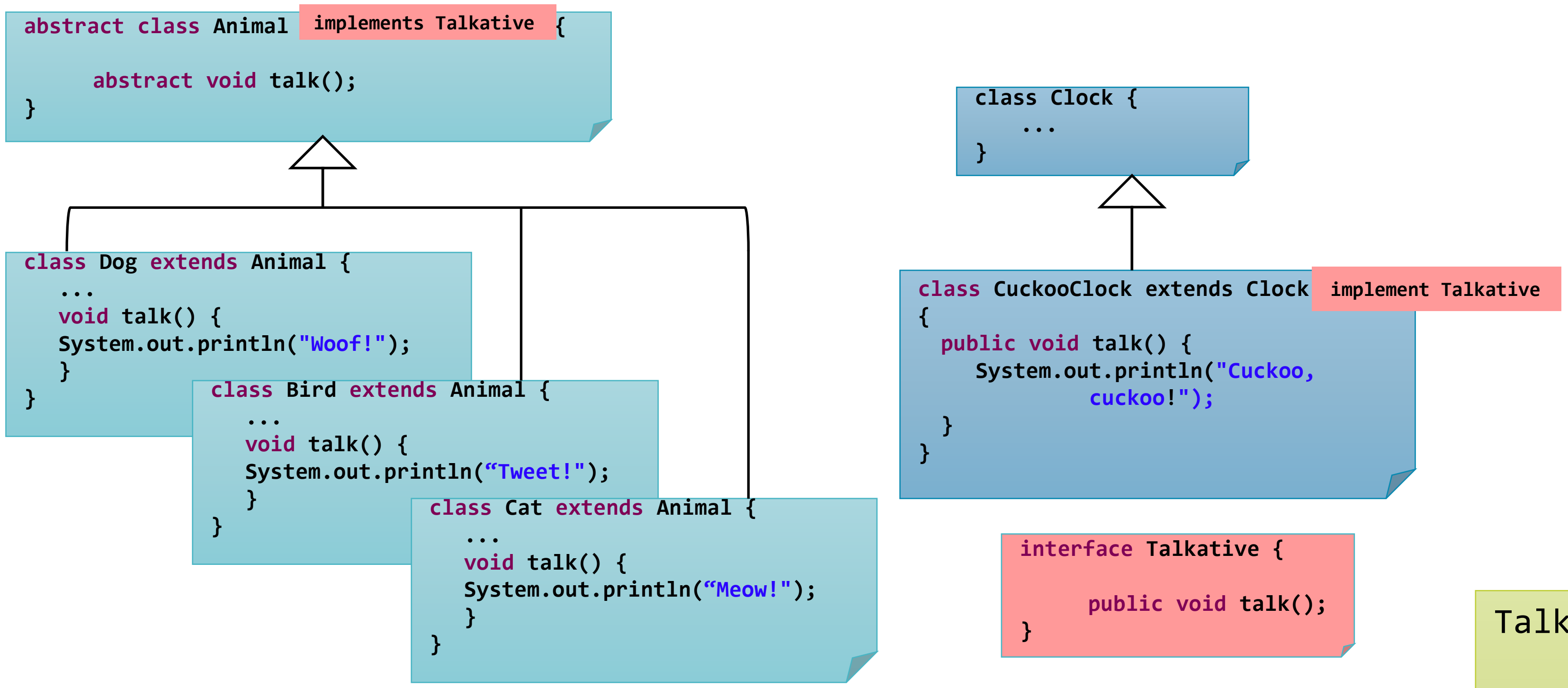


```
Drawable d;
..
d = new
RectangleDrawable(…);
...
d.draw(g);
~~d.surface();~~
```

allows to focus only certain features of an object

4.2. Using Interface to group classes

- Sometimes, we want to list down all things or animals that can produce sound (Talkative)



```java
abstract class Animal implements Talkative {

      abstract void talk();
}
```

```java
class Dog extends Animal {
   ...
   void talk() {
   System.out.println("Woof!");
   }
}
```

```java
class Bird extends Animal {
   ...
   void talk() {
   System.out.println("Tweet!");
   }
}
```

```java
class Cat extends Animal {
   ...
   void talk() {
   System.out.println("Meow!");
   }
}
```

```java
class Clock {
   ...
}
```

```java
class CuckooClock extends Clock implement Talkative
{
  public void talk() {
     System.out.println("Cuckoo,
           cuckoo!");
  }
}
```

```java
interface Talkative {

      public void talk();
}
```

```java
Talkative soundMakers[] =
                  new Talkative[10];
soundMakers[0] = new Dog();
soundMakers[1] = new Bird();
…
```

4.3. Using Interface as future features

- There are some services or features are not yet implemented, just have plan to have next version of application
  - At this version, these planed features are abstracts
- Some features could not be implemented at this class level, so keep it abstract to solve in next level (sub-classes)
  - This class will become abstract class, to prevent it to become abstract, we implement method, but in it we through **`UnsupportedOperationException`**

```
public void doSomething(){
  throw new UnsupportedOperationException("Next Version");
}
```

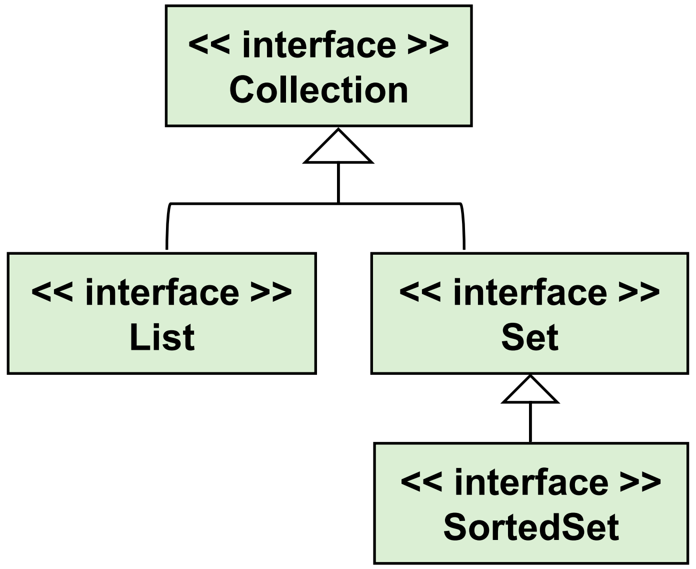
5. Inheritance of interface

- In the same way that a class can have subclasses, an interface can have "sub-interfaces"
- A sub-interface
  - inherits all abstract methods and constants of "super-interface"
  - can define new constants and abstract methods

```
interface Set extends Collection
{
    ...
}
```

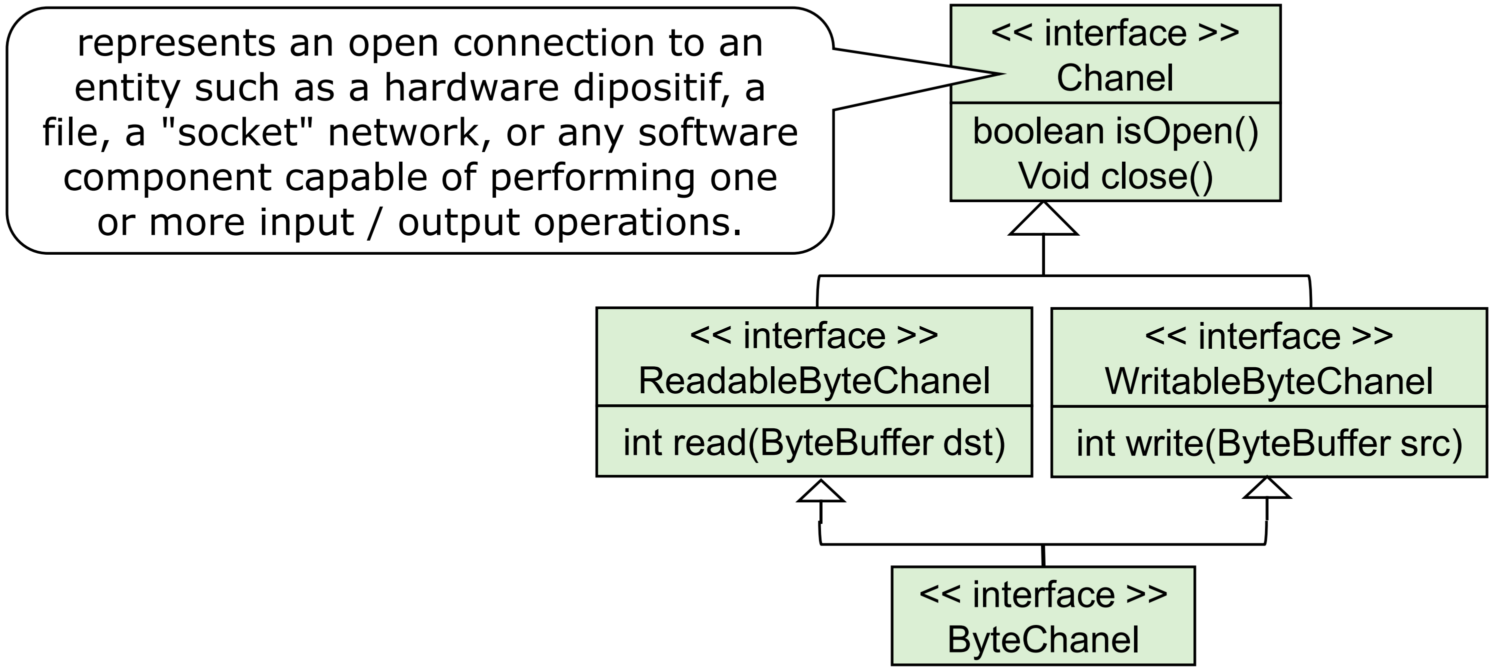
5.1. Inheritance of interface

- A class that implements an interface must implement all abstract methods defined in the interface and in the parent interfaces it inherited.



5.2. Inheritance multiple interfaces

- Unlike class, an interface can extend more than one interface at a time


represents an open connection to an entity such as a hardware dipositif, a file, a "socket" network, or any software component capable of performing one or more input / output operations.
<< interface >> Chanel
boolean isOpen()
Void close()
<< interface >> ReadableByteChanel
int read(ByteBuffer dst)
<< interface >> WritableByteChanel
int write(ByteBuffer src)
<< interface >> ByteChanel

```java
package java.nio;
interface ByteChannel extends ReadableByteChanel, WriteableByteChanel
{
}
```

5.3. Advantages of interface

- The interfaces allows to overcome constraints of inheritance.
  - When considering a class that implements one or more interfaces, one is sure that the implementation code is in the body of the class. Excellent location code (default multiple inheritance unless you inherit purely abstract classes).
  - Allows high scalability object model
- « Smarter Java development » *Michael Cymerman* , javaworld August 1999.

  http://www.javaworld.com

5.3. Advantages of interface

- By incorporating interfaces into your next project, you will notice benefits throughout the lifecycle of your development effort. The technique of coding to interfaces rather than objects will improve the efficiency of the development team by:
  - *Allowing the development team to quickly establish the interactions among the necessary objects, without forcing the early definition of the supporting objects*
  - *Enabling developers to concentrate on their development tasks with the knowledge that integration has already been taken into account*
  - *Providing flexibility so that new implementations of the interfaces can be added into the existing system without major code modification*
  - *Enforcing the contracts agreed upon by members of the development team to ensure that all objects are interacting as designed*

Test

Question	Possible Answers	Correct Answer
Fill in the gaps:	Java's interface gives you more ……………… than you can get with singly inherited families of classes, without the "burden" of multiple ……… of implementation.	
Fill in the gap:	An interface can extends multiple ……………. The class that implements this interface must implement all methods of this interface and its …………………………….	
Which are benefits of interfaces:	a) No need to define early of the supporting objects b) Add new implementation without big modification c) Without “burden” of inheritance many interfaces d) Define future methods	

Test

Question	Possible Answers	Correct Answer
The fields in interface are:	a) Implicitly private b) Implicitly static c) Implicitly final d) Implicitly static final	
The methods in interface are:	a) Public b) Abstract c) Final d) Static	

Practice

Exercises
1) Implement class Animals and sub-classes in this course
2) Create a class that implements interface SortedSet in this course
3) A Person can be a - student (at learning time) - waiter/waitress (at part time job during evening time) - sport person (at weekend) Create interfaces to define each services of a Person. Then, create class Person that implements these interfaces.

Summarize

- Interface is similar to class but it does only methods without body (abstract methods) and final static fields
- To declare interface, we use syntax “public interface InterfaceName {…}”.
- To realize interface, we use syntax “public class ClassName implements InterfaceName {…}”
- Using interface, we can group all different types of classes.
- An interface can inherit many interface at a time

Reference

- “Object Oriented Programming” by Ph. Genoud – Université Joseph Fourier, 2006
- “Building Skills in Object-Oriented Design” by Steven F. Lott, 2009
- “Exercices en Java” 2nd edition by Groupe Eyrolles, 2006
- “Java Programming – Introductory” by Joyce Farrell, 1999
- “Java Examples in a Nutshell” 3rd Edition by David Flanagan, 2004
- “Programmer en Java” 5th edition by Groupe Eyrolles, 2008