Introduction to Java

Nai-Wei Lin

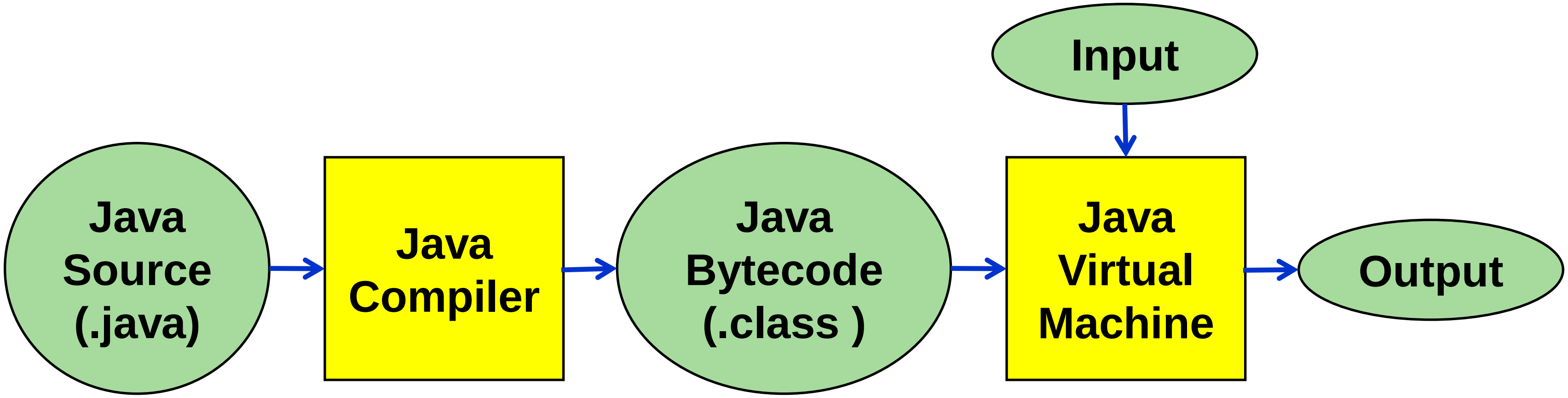
Department of Computer Science and Software Engineering

National Chung Cheng University

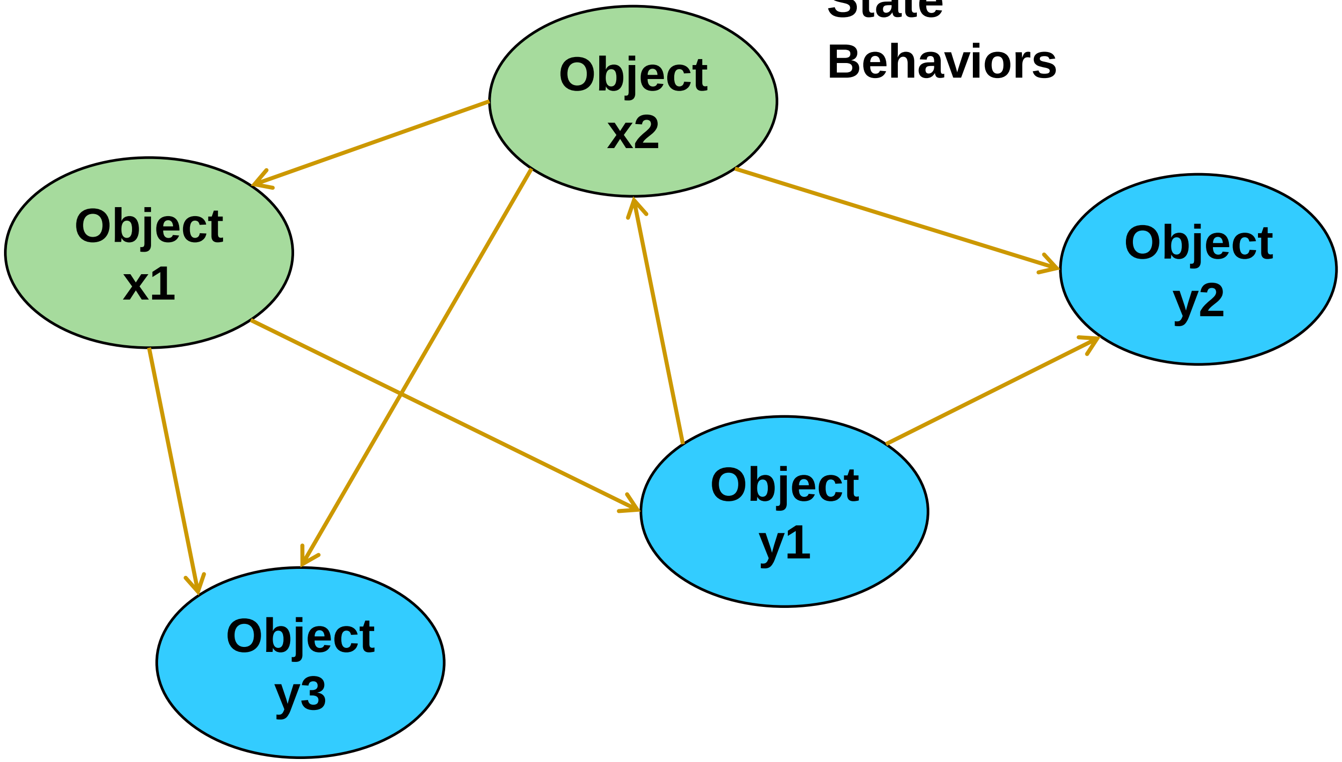
Content

- Platform-independent
- Object-oriented
- Exception-handling

Platform-Independent



Object-Oriented



State
Behaviors

Objects

- Objects have states and behaviors.
- A dog has states: name, breed, color.
- A dog has behaviors: wagging, barking, eating.

Classes

- A class can be viewed as a template or blue print that defines the states and behaviors of objects of the same type.
- An object of a class is an instance of the class.
- The class contains a set of instant variables. The state of an object is represented by the values assigned to its instant variables.
- The class contains a set of methods. Each method defines a behavior of the object.

Classes – An Example

```
class Dog
{
    private String name;
    private String breed;
    private String color;
    public void wagging() { … }
    public void barking() { … }
    public void eating() { … }
        ...
}
```

Constructors

```
class Dog
{
    private String name;
    private String breed;
    private String color;
    public void wagging() { … }
    public void barking() { … }
    public void eating() { … }
    public Dog(String n, String b, String c) { … }
        ...
}
```

Constructors – An Example

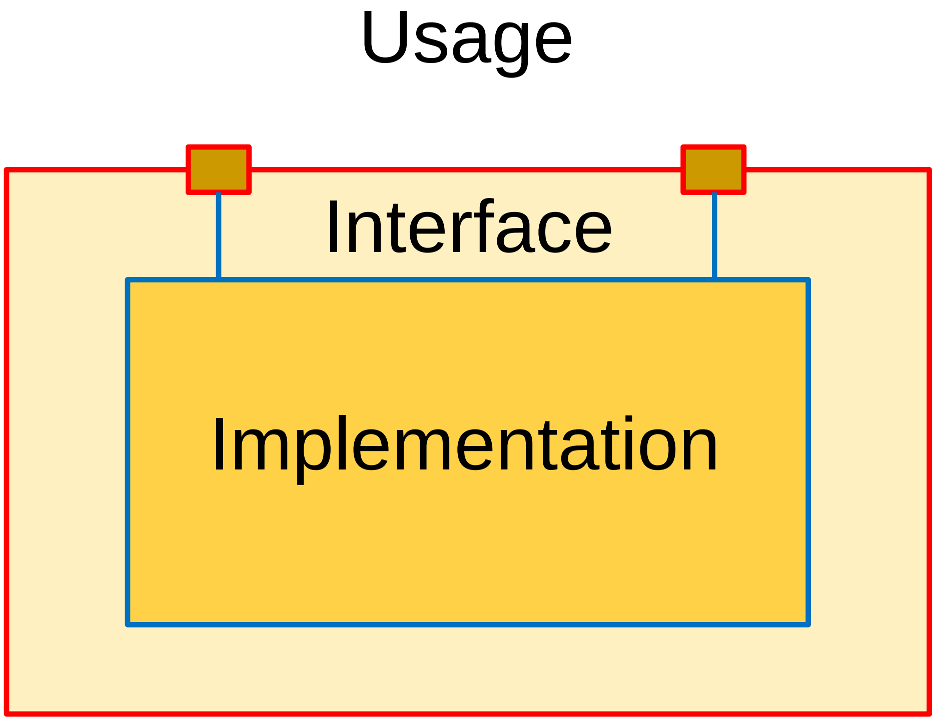
```
public Dog(String n, String b, String c) {
    name = n;
    breed = b;
    color = c;
}

class Main {
    public static void main(String[ ] args) {
        Dog dog =
            new Dog(“SmallBlack”, “Formosan”, “black”);
        dog.wagging();
    }
}
```

Encapsulation

- Encapsulation is the technique of making the instant variables in a class private and providing access to the instant variables via public methods. Access to the data and code is tightly controlled by an interface.
- If a field is declared private, it cannot be accessed by anyone outside the class, thereby hiding the fields within the class. For this reason, encapsulation is also referred to as data hiding or information hiding.

Encapsulation



Usage

Interface

Implementation

Benefits of Encapsulation

- The main benefit of encapsulation is the ability to modify our implementation without breaking the code of others who use our code.
- With this feature encapsulation gives maintainability and extensibility to our code.

Class Complex

```
class Complex
{   // an abstract data type
    private float re;
    private float im;
    public Complex() { re = 0; im = 0; }
    public Complex(float r, float i) { re = r; im = i; }
    public Complex add(Complex c) { … }
    public Complex sub(Complex c) { … }
    public Complex mul(Complex c) { … }
    public Complex div(Complex c) { … }
    public String toString() { … }
        …
}
```

Class Complex – An Example

```
class Main {
    public static void main(String[ ] args) {
        Complex c1 = new Complex(2.0, 3.0);
        Complex c2 = new Complex(3.0, 2.0);
        Complex c3 = c1.add(c2);
        System.out.println(“Value of c3: ” + c3.toString());
    }
}
```

Getters and Setters

```
class Dog
{
    private String name;
    private String breed;
    private String color;
    public String getName() { return name; }
    public String getBreed() { return breed; }
    public String getColor() { return color; }
    public void setName(String n) { name = n; }
    public void setBreed(String b) { breed = b; }
    public void setColor(String c) { color = c; }
        ...
}
```

Getters and Setters – An Example

```
class Main {
    public static void main(String[ ] args) {
        Dog dog =
            new Dog(“SmallBlack”, “Formosan”, “black”);
        dog.setName(“BigBlack”);
        System.out.println(“Name of dog: ” + dog.getName());
    }
}
```

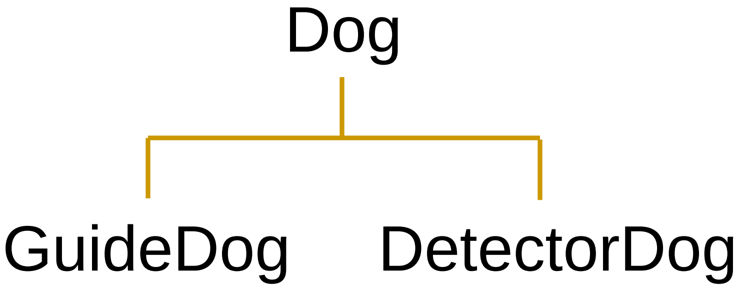
Inheritance

- Objects can be classified as a hierarchy of classes.
- A subclass inherits the instant variables and methods of its superclass.
- A subclass usually also has its own instant variables and methods.
- A subclass is more specific than its superclass.
- An object of a subclass (type) is also an object of its superclass (type).

Inheritance

```
class GuideDog extends Dog
{
    private String hostName;
    public void guide();
        ...
}

class DetectorDog extends Dog
{
    private String airportName;
    public void detect();
        ...
}
```



Polymophism

- An object is polymorphic if it is an object of more than one class.
- An object of a subclass is also an object of its superclass.
- All java objects are polymorphic since any object is an instance of its own class and an instance of the class Object.

Polymophism – An Example

```
class Super { public void fp() { … } }

class Sub { public void fb() { … } }

class Main {
    public static void main(String[ ] args) {
        Super sp = new Super();
        Sub sb = new Sub();
        sp.fp(); ☑     sb.fp(); ☑     sp.fb(); ☒     sb.fb(); ☑
        gp(sp); ☑      gp(sb); ☑      gb(sp); ☒      gb(sb); ☑
    }
    public static void gp(Super s) { … }
    public static void gb(Sub s) { … }
}
```

An object of subclass is also an object of superclass.

Overriding

- A subclass can override a method inherited from its superclass.
- Namely, a subclass can define a behavior that is more specific to the subclass.

Overriding – An Example

```
class Animal {
  public void move(){
    System.out.println("Animals can move");
  }
}

class Dog extends Animal{
  public void move() {
    System.out.println("Dogs can walk and run");
  }
}
```

Overriding – An Example

```
class TestDog {
  public static void main(String args[]){
    Animal a = new Animal();
    // Animal reference and object
    Animal b = new Dog();
    // Animal reference but Dog object
    a.move();  // Runs the method in Animal class
    b.move();  //Runs the method in Dog class
  }
}
```

Exceptions

- An exception is a problem that arises during the execution of a program.
- An exception can be caused by a user error (entering invalid data), a programmer error (accessing a non-existent object), or a physical resource that has failed in some manner (disk malfunction).

Exception Hierarchy

- All exception classes are subclasses of the java.lang.Exception class.

```
class MyException extends Exception
{
    …
}
```

Exception Catching

- A method catches an exception using a try/catch block.

```
try {
        // Protected code
} catch (ExceptionName e) {
        // Catch block
}
```

Exception Throwing

- If a method does not handle a checked exception, the method must declare it using the throws keyword. The throws keyword appears at the end of a method's signature.

```
public void method1() throws Exception1
{
  // Method Implementation
}
```

Exception Throwing

- A method can throw an exception, either a newly instantiated one or an exception that it just caught, by using the throw keyword.

```
public void method1() throws Exception1
{
   // Method Implementation
   // detect a malfunction or an exception
   throw new Exception1();
}
```