ADVANCED OOP WITH JAVA

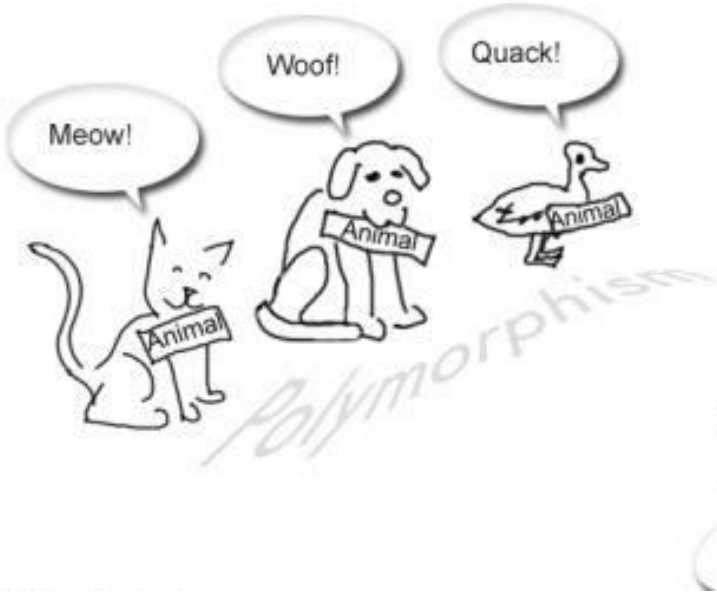
Instructor:

Section 1

POLYMORPHISM

Polymorphism Overview (1/2)

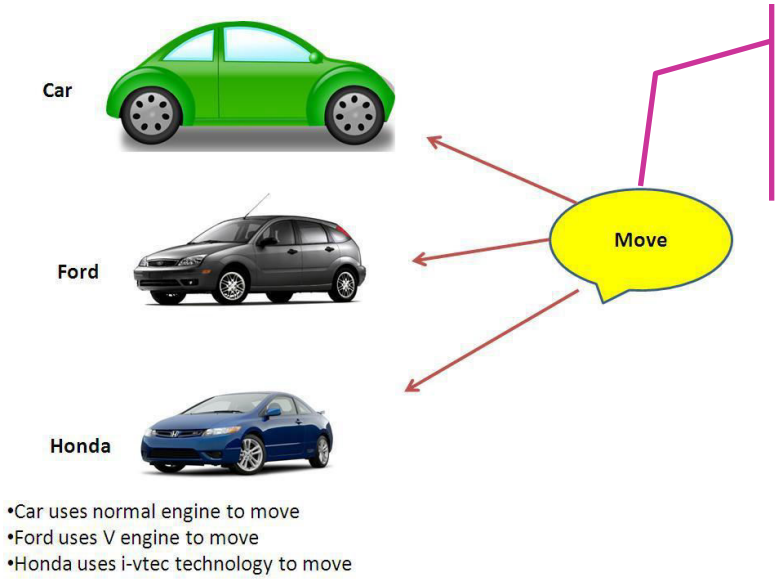
- In object-oriented programming, **polymorphism** (from the Greek (Hy Lạp) meaning "**having multiple forms**").
  - ✓ Polymorphism is derived from 2 greek words: poly and morphs.
  - ✓ The word "*poly*" means *many* and "*morphs*" means *forms*.



- There are two types of polymorphism in java:
  - ✓ compile time polymorphism: method overloading.
  - ✓ runtime polymorphism: method overriding.

one name, many forms.

Polymorphism Overview (2/2)



Polymorphism is a generic term for having many forms.

- You can use the same name for several different things
  - ✓ the compiler automatically figures out which version you wanted.
- There are several forms of polymorphism supported in Java, shadowing, overriding, and overloading.

Types of Polymorphism

- There are two types of polymorphism in java:
  - ✓ **Static Polymorphism** also known as **compile time** polymorphism
  - ✓ **Dynamic Polymorphism** also known as **runtime** polymorphism
- **Compile time Polymorphism** (or Static polymorphism)
  - ✓ Polymorphism that is resolved during compiler time is known as static polymorphism. Method overloading is an example of compile time polymorphism.
  - ✓ **Method Overloading**: This allows us to have **more than one method** having the **same name**, if the parameters of methods are different in number, sequence and data types of parameters.

Method Overloading

- Overloaded methods let you reuse the same method name in a class, but with different arguments (and optionally, a different return type).
- There are two ways to overload the method in java:
  - ✓ By changing number of arguments
  - ✓ By changing the data type
- In a subclass,
  - ✓ you can overload the methods inherited from the superclass.
  - ✓ such overloaded methods neither hide nor override the superclass methods
  - ✓ they are new methods, unique to the subclass.

Method Overloading

- Some main rules for overloading a method:
  - ✓ Overloaded methods **must change the argument list**.
  - ✓ Overloaded methods **can change the return type**.
  - ✓ Overloaded methods **can change the access modifier**.
    - *With override: Cannot reduce the visibility*
  - ✓ A method can be overloaded in the same class or in a subclass.

Note: When I say method signature I am not talking about return type of the method, for example if two methods have same name, same parameters and have different return type, then this is not a valid method overloading example. This will throw compilation error.

Types of Polymorphism

```
public class SimpleCalculator {

  int add(int number1, int number2) {
    return number1 + number2;
  }

  int add(int number1, int number2, int number3) {
    return number1 + number2 + number3;
  }

  double add(double number1, double number2) {
    return number1 + number2;
  }
}


public class TestSimpleCalculator {

  public static void main(String[] args) {
    SimpleCalculator simpleCalculator = new SimpleCalculator();

    System.out.println(simpleCalculator.add(10, 20));
    System.out.println(simpleCalculator.add(10, 20, 30));
    System.out.println(simpleCalculator.add(12.5, 20.5));
  }
}
```

Method Overriding

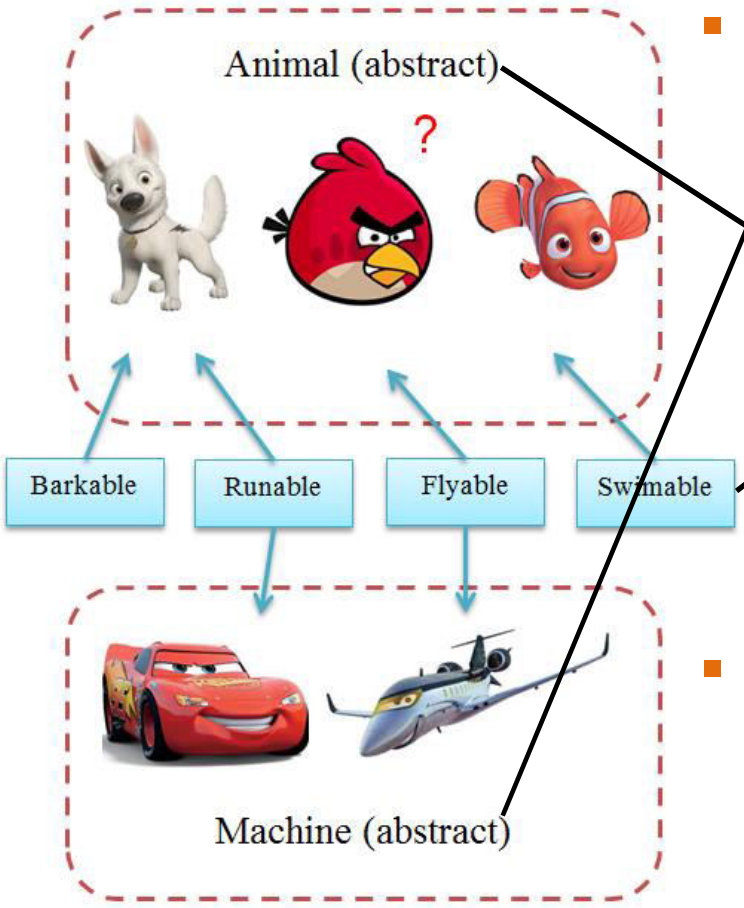
- Declaring a method in **subclass** which is already present in **parent class** is known as method overriding.
- The main advantage of method overriding is:
  - ✓ the class can give its own specific implementation to a inherited method without even modifying the parent class(base class).
- **Rules of method overriding in Java:**
  - ✓ Method name: method must have the same name as in the parent class.
  - ✓ Argument list: must be same as that of the method in parent class,
  - ✓ Access Modifier: cannot be more restrictive than the overridden method of parent class.

Section 2

ABSTRACTION

Abstraction Overview

- **Abstraction** is a process of hiding the implementation details and showing only functionality to the user.



- **Example:**
  - ✓ Sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery

It shows only important things to the user and hides the internal details

Focus on what the object does instead of how it does it.

- **There are two ways to achieve abstraction in java:**
  - ✓ Abstract class (0 to 100%)
  - ✓ Interface (100%)

Interface

- An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.
- The interface in Java is *a mechanism to achieve abstraction*.
  - ✓ There can be only abstract methods in the Java interface, not method body.
  - ✓ It is used to achieve abstraction and multiple inheritance in Java.
- **Why use Java interface?**

1. It is used to achieve abstraction.
2. By interface, we can support the functionality of multiple inheritance.
3. It can be used to achieve loose coupling.

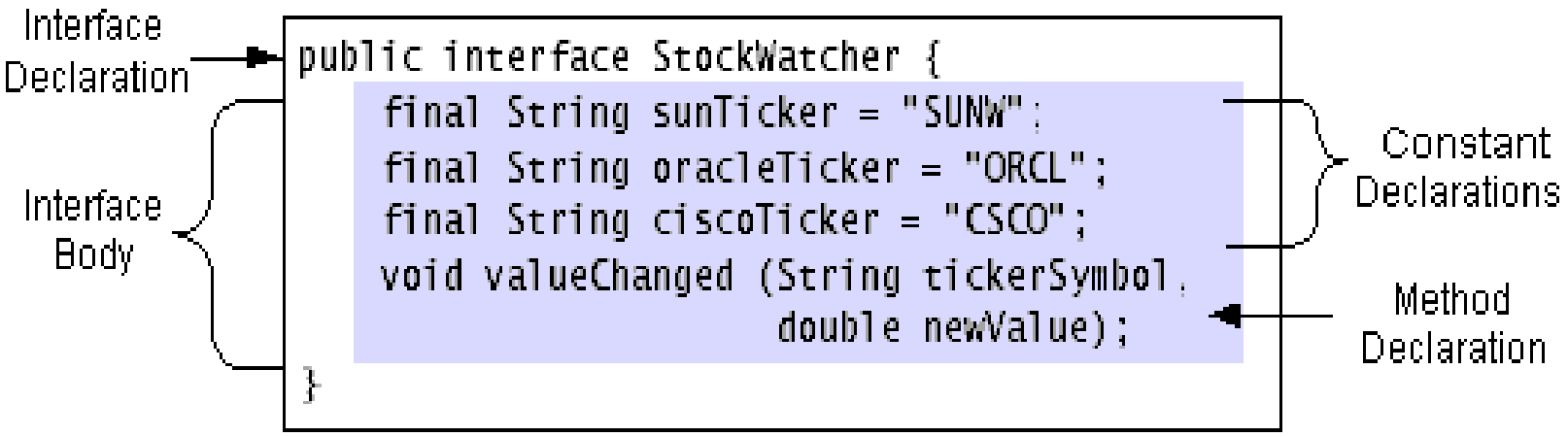
Tính trừu tượng giúp giảm coupling vì nó cho phép bạn tách biệt hành vi khỏi chi tiết triển khai, từ đó giúp các lớp phụ thuộc vào một hợp đồng chung (contract) thay vì phụ thuộc vào cách thực hiện chi tiết của lớp khác..

Interface

- **Syntax:**

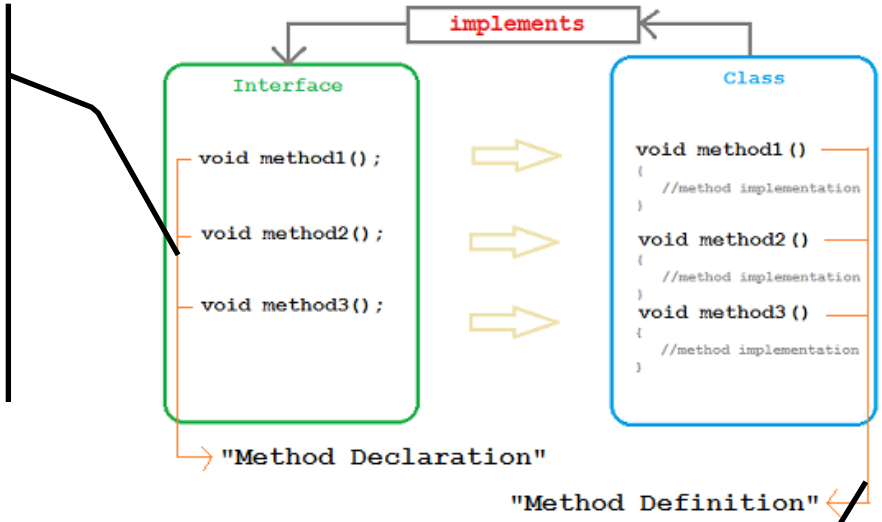
```
[public] interface <InterfaceName>[extends SuperInterface] {
 // Interface Body
}
```

- **Example**:



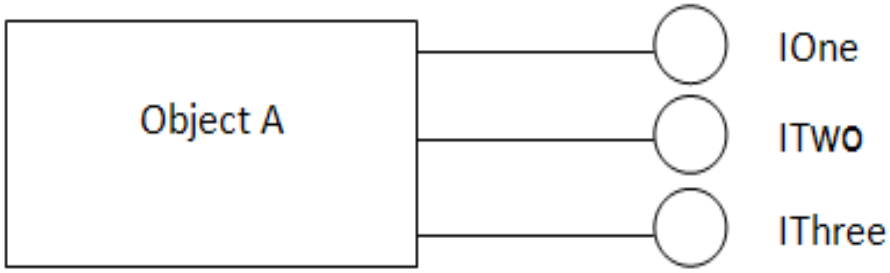
Interfaces

- ✓ An interface is a definition of method prototypes and possibly some constants (static final fields).
- ✓ An interface does not include the implementation of any methods.



- ✓ A class can **`implement`** an interface, this means that it provides implementations for all the methods in the interface.

- ❖ Java classes can implement any number of interfaces (multiple interface inheritance).



Multiple inheritance in Java by interface

- Multiple inheritance is not supported in the case of class because of ambiguity.
- However, it is supported in case of an interface because there is no ambiguity.
- It is because its implementation is provided by the implementation class.
- **Example**:

```
interface Printable {
  void print();
}

interface Showable {
  void print();
}

class A4 implements Printable, Showable {
  public void print() {
    System.out.println("Printing and showing document");
  }
}

public class TestInterface2 {
public static void main(String[] args) {
    A4 a4 = new A4();
    a4.print();
  }
}
```

Abstract class and Interface

- **Abstract class** and **interface** both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.
- But there are many differences between abstract class and interface that are given below.

No.	Abstract class	Interface
1	Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods.
2	Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3	Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4	Abstract class **can have static methods, main method and constructor**.	Interface **can't have static methods, main method or constructor**.
5	Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
6	The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.

7	**Example:**
	public abstract class Shape{
	public abstract void draw();
	}
		Example:
		public interface Drawable{
		void draw();
		}

Thank you