

# Object-Oriented Programming in Scala

Dr. Nguyen Hua Phung

HCMC University of Technology, Viet Nam

08, 2016

**1 OOP Introduction**

**2 Scala**

- Programs as collections of collaborating objects
- Object has public interface, hidden implementation
- Objects are classified according to their behavior
- Objects may **represent real-world entities** or entities that produce services for a program
- Objects may be **reusable** across programs

There are many object-oriented programming (OOP) languages

- Some are pure OOP language (e.g., Smalltalk).
- Newer languages do not support other paradigms but use their imperative structures (e.g., Java and C#).
- Some support procedural and data-oriented programming (e.g., Ada and C++).
- Some support functional program (e.g., Scala)

- Abstract data types
  - Encapsulation
  - Information Hiding
- Class Hierarchy
- Inheritance
- Polymorphism - Dynamic Binding

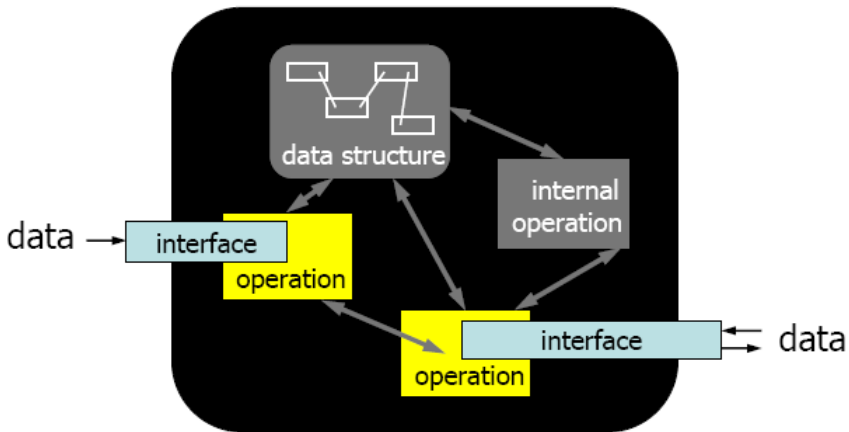
An **abstract data** type is data type that satisfies the following two conditions:

- **Encapsulation:**

- Attributes and operations are combined in a single **syntactic unit**
- compiled **separately**

- **Information Hiding:**

- The **access** to members are **controlled**
- **Reliability** increased



Operations on a stack:

- `create(stack)`
- `destroy(stack)`
- `empty(stack)`
- `push(stack,element)`
- `pop(stack)`
- `top(stack)`

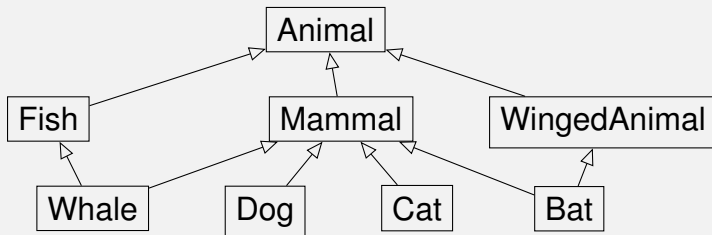
Client code:

```
...  
create ( stk1 );  
push( stk1 , color1 );  
push( stk1 , color2 );  
if ( ! empty( stk1 ) )  
    temp = top( stk1 );  
...
```

Implementation can be adjacent or linked list. Client: don't care!



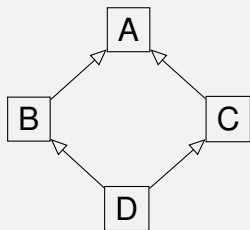
- A class may have some Subclasses
- A class may have one/many Superclass(es)



- A subclass is able to inherit **non-private** members of its superclasses
- A subclass can add its own members and **override** its inherited members
- Single vs. multiple inheritance
- Inheritance increases the **reusability** in OOP



# Diamond Problem in Multiple Inheritance



- if D inherits from B and C different versions of the same behaviour, **which version** will be effective in D?
- this problem, called **diamond problem**, is solved differently in different OO language
- is there the diamond problem in Java?

ko, java ko có đa kế thừa, mà có đa hiện thực

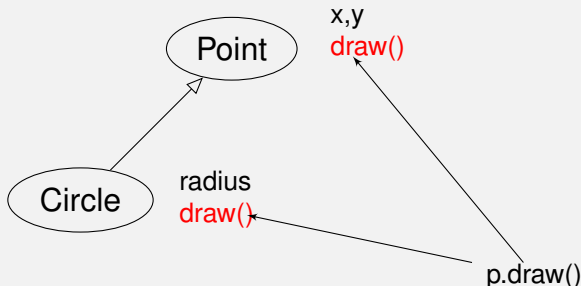
## Class vs. Object

- A class defines the abstract characteristics of a thing
  - its attributes or properties and
  - its behaviors or methods or features.  
A Dog has *fur* and is able *to bark*
- An object is a particular instance of a class.  
Lassie is a dog

## Method vs. Message

- A method **describes a behavior** of an object.  
A dog **can** bark
- A **message** is a process at which a method of an object is invoked.  
Lassie barks

- Polymorphism: different objects can respond to the **same message** in different ways.



- Dynamic binding**

- There are two kinds of variables in a class:
  - Class variables
  - Instance variables
- There are two kinds of methods in a class:
  - Class methods – accept messages to the class
  - Instance methods – accept messages to objects

- Invented by Martin Odersky at EPFL, Lausanne, Switzerland.
- Similar to Java
- Work smoothly with Java
- Run on Java Virtual Machine
- OOP + FP
- Include lexer and parser generator



## Class

- class [1]
- abstract class [5]
- trait [2]
- case class [3]

## Object

- new <class name>
- <case class name>
- object



## Example [7]

```
class Rational(n: Int, d: Int){  
  require(d != 0)  
  
  private val g = gcd(n.abs, d.abs)  
  private def gcd(a: Int, b: Int): Int =  
    if (b == 0) a else gcd(b, a % b)  
  val numer = n / g  
  val denom = d / g  
  
  def this(n: Int) = this(n, 1)  
  
  def + (that: Rational): Rational =  
    new Rational(  
      numer * that.denom + that.numer * denom,  
      denom * that.denom  
    )  
  
  override def toString = numer + "/" + denom  
}
```

## Example on Abstract class [4]

```
abstract class Element {  
    def contents: Array[String] //no body: abstract  
    val height = contents.length  
    val width =  
        if (height == 0) 0 else contents(0).length  
}  
class ArrayElement(cons: Array[String])  
    extends Element {  
    def contents: Array[String] = cons  
}  
class LineElement(s: String)  
    extends ArrayElement(Array(s)) {  
    override def width = s.length  
    override def height = 1  
}
```

```
object Element {  
  
  def elem(contents: Array[String]): Element =  
    new ArrayElement(contents)  
  
  def elem(line: String): Element =  
    new LineElement(line)  
}  
  
val space = Element.elem(" ")  
val hello = Element.elem(Array("hello ", "world"))
```

Which kind of Element will be assigned to *space* and *hello*?

```
abstract class Expr
```

```
case class Var(name: String) extends Expr
```

```
case class Number(num: Double) extends Expr
```

```
case class UnOp(operator: String, arg: Expr)  
                                extends Expr
```

```
case class BinOp(operator: String,  
                  left: Expr, right: Expr) extends Expr
```

```
val v = Var("x")
```

```
val op = BinOp("+", Number(1), v)
```

```
v.name
```

```
op.left
```

## Example 1 on Traits [9]

Reusability:

```
abstract class Bird
```

```
trait Flying {  
    def flyMess: String  
    def fly() = println(flyMess)  
}
```

```
trait Swimming {  
    def swim() = println("I'm swimming")  
}
```

```
class Penguin extends Bird with Swimming
```

```
class Hawk extends Bird with Swimming with Flying {  
    val flyMess = "I'm a good flyer"  
}
```

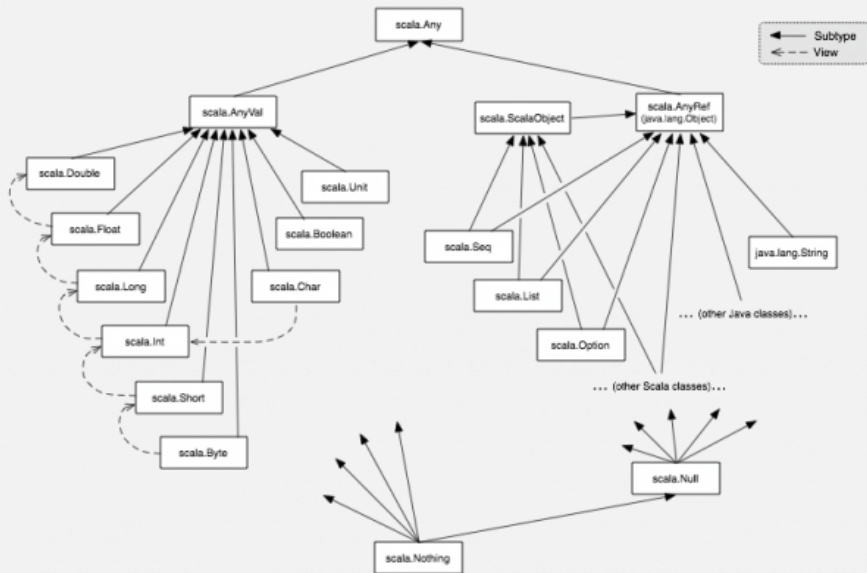
```
class Frigatebird extends Bird with Flying {  
    val flyMess = "I'm an excellent flyer"  
}
```



Stackable Modifications:

```
abstract class IntQueue {  
    def get(): Int  
    def put(x: Int)}  
class BasicIntQueue extends IntQueue {  
    private val buf = new ArrayBuffer[Int]  
    def get() = buf.remove(0)  
    def put(x: Int) { buf += x }  
trait Doubling extends IntQueue {  
    abstract override def put(x: Int) { super.put(2*x) }  
trait Incrementing extends IntQueue {  
    abstract override def put(x: Int) { super.put(x+1) }  
  
val queue =  
    new BasicIntQueue with Incrementing with Doubling  
queue.put(10)  
queue.get() //???
```

# Scala Hierarchy [8]



- public
- protected
- private
- protected[<name>]
- private[<name>]

Example,

**package** Assignment

...

**protected**[Assignment] **val** field1 = 100

**private**[this] **val** field2 = 2;



What are still in your mind?

- [1] **Classes and Objects**, <http://www.artima.com/pinsled/classes-and-objects.html>, 19 06 2014.
- [2] **Traits**, <http://www.artima.com/pinsled/traits.html>, 19 06 2014.
- [3] **Case Class and Pattern Matching**, <http://www.artima.com/pinsled/case-classes-and-pattern-matching.html>, 19 06 2014.
- [4] **Abstract Members**, <http://www.artima.com/pinsled/abstract-members.html>, 19 06 2014.
- [5] **Compositions and Inheritance**, <http://www.artima.com/pinsled/composition-and-inheritance.html>, 19 06 2014.
- [6] **Packages and Imports**, <http://www.artima.com/pinsled/packages-and-imports.html>, 19 06 2014.

- [7] **Functional Objects**, <http://www.artima.com/pins1ed/functional-objects.html>, 19 06 2014.
- [8] **Scala Hierarchy**, <http://www.artima.com/pins1ed/scalas-hierarchy.html>, 19 06 2014.
- [9] **Learning Scala part seven -Traits**, <http://joelabrahamsson.com/learning-scala-part-seven-traits/>, 19 06 2014.