

**De La Salle University • College of Computer Studies**

**OOP**

Name (last name first) : Fernandez, Ryan Austin

Poblete, Clarisse Felicia M.

San Pedro, Marc Dominic

Tan, Johansson E.

Section : G01

Date of Submission : April 7, 2016

1. Differentiate the following parameter passing schemes and identify whether your language makes use of them or not:
   * 1. Pass-by-Value
     2. Pass-By-Reference
     3. Pass-by-Value Result

Passing by value only passes the actual value to the function. Any changes made to the value are not reflected in the original calling function.

Passing by Reference passes a pointer to the value. Any changes are reflected in the original calling function.

Passing by Value Result creates a copy of the parameter’s value, and passes that copy (by reference) to the function. The final value of the copy variable is copied back to original parameter.

**C++**

C++ supports passing by value, passing by reference and passing by value result.

**C#**

C# supports passing by value and passing by reference.

**Python**

Python supports passing by value and passing by reference, depending on the object type.

**Javascript**

Javascript supports pass-by-value and pass-by-reference.

**Scala**

Scala only supports objects. As a result, everything is passed by reference.

1. Discuss how the different parameter schemes are being implemented by languages and their side-effects when it comes to their usage.

**C++**

Without using any & or \* for the parameter, it follows pass by value. Using & for the parameter is pass by reference. Using \* for the parameter is pass by value result.

**C#**

C# supports pass by value for primitive data types, and pass by reference for Objects.

**Python**

Python treats data or collection types as either mutable or immutable.

Immutable types, such as *int* or *String*, are passed to functions by value, and there is no direct way of passing them by reference.

Mutable types, such as lists and custom classes, are passed to functions by reference, so any changes made to these objects will reflect in the original object.

**Javascript**

For primitive types, it follows pass by value. For Objects, it follows pass-by-reference.

**Scala**

In addition, all actual parameters are treated as constants, so modifying them is not allowed. You can, however, change the attributes of any passed objects, but changing values of passed Ints or Floats will not work.

1. In C++ what is the purpose of const in a function and parameter declaration.

When const is used in parameter declaration, it ensures that the parameter will not change value in the function. If const is used before a function’s name, then it makes the return type as constant. If const is after the () of a function, then the function itself is a const. These case is only used for member functions of a class. Using const for this makes the function not able to change any class members. Using const for variables makes these variables not changeable. They would also need to be initialized.

1. In C++ what is the difference of passing a parameter with “&” versus “\*”? Cite important difference and when they should be used in C++. Comment about the language’s choice for the design.

Say we have this:

void test(int &y){...}

void test2(int \*y){...}

int main()

{

int x;

test(x);

int \*z;

test2(z);

}

In test, parameter y, being used with the & symbol, will be a reference to the argument when the function was called. This means that the parameter is like the variable, used as argument, itself.

In test2, parameter y, being used with the \* symbol, will be a pointer of the argument when the function was called.

& parameters cannot be null while \* parameters can. This constraint on each shows when to use one over the other. For structs and variables, using & is, in a sense, better than using \*.

1. Discuss which languages have named parameters and how they are used. How are they written in your respective languages? Is there a difference between how they implement it?

**C#**

In C#, one can use the names of the parameters if he/she does not remember the order of these parameters. For example, if one has a function CalculateBMI(double weight, double height), then one might get the parameters mixed up occasionally, especially since they are both of datatype *double*. To avoid this, either of the following can be done:

CalculateBMI(weight: 123, height: 64);

CalculateBMI(height: 64, weight: 123);

Both these function calls will produce the same result.

**Python**

Python supports named parameters. For example, given the function:

def func(one, two, three):

# Function code here

This function can be called like this:

func(three = 3, two = “Two”, one = 1.0)

**Scala**

Scala supports named parameters. For example:

def test(x : Test,increment:Int) {

x.value += increment

}

Which uses the class

class Test(var value : Int) {

override def toString:String = value.toString

}

So the sample code

var temp = new Test(5)

println(temp)

test(temp,5)

println(temp)

Will work as well as the code

var temp = new Test(5)

println(temp)

test(increment = 5,x = temp)

println(temp)

And both calls perform the same operation.

1. Discuss which languages have optional parameters and how they are used. How are they written in your respective languages? Is there a difference between how they implement it?

**C++**

C++ supports optional parameters. An example is this:

void func(int x, int y = 24601)

{

//some code

}

int main()

{

func(1);

func(1,2);

}

Optional parameters must be the rightmost parameters. IF you have more than one optional parameters, you cannot supply your own value for 2nd, and so forth, optional parameters if you do not put a value for the preceding optional parameters.

**C#**

In C#, a function parameter p1 of a certain function f can be specified as optional to mean that a call to f may or may not necessarily supply a value to p1. Each optional parameter has a default value as part of its definition. If no argument is sent for that parameter, the default value is used.

Optional parameters must be defined at the end of the parameter list, after any required parameters. If the caller provides an argument for any one of a succession of optional parameters, it must provide arguments for all preceding optional parameters.

Sample:

public void ExampleMethod(int required, string optionalstr = "default string",  
 int optionalint = 10)

Given this method signature, the following call

ExampleMethod(3, ,4);

will cause an error, since an argument was provided for the third parameter but not the second. Instead, if you want to override the default value of optionalInt, what should be done is:

ExampleMethod(3, optionalint: 4);

**Python**

Python supports optional parameters. Here is an example of how they are written:

def func(required, optional1 = 10, optional2 = “String”):

# Function code here

In case you want to assign a value to the second optional parameter but not the first, Python still allows you to since it allows you to specify arguments (even required ones) in any order by specifying the name. For example:

func(optional2 = “Hello”, required = 10)

**Scala**

Scala supports optional parameters. A sample function is

def test(x : Test,increment:Int = 5) {

x.value += increment

}

Which uses the class

class Test(var value : Int) {

override def toString:String = value.toString

}

So the sample code

var temp = new Test(5)

println(temp)

test(temp)

println(temp)

Results in the default value 5 being added while

var temp = new Test(5)

println(temp)

test(temp,1)

println(temp)

Results in the provided value 1 being added.

1. Identify features that are supported intrinsically by your languages for object-oriented system/program design.

**C++**

C++ supports abstraction, inheritance, polymorphism, and encapsulation.

**C#**

C# supports abstraction, inheritance, polymorphism, and encapsulation.

**Python**

Python supports polymorphism, inheritance and encapsulation

**Javascript**

Javascript supports instantiation,aggregation,composition

**Scala**

Scala supports attributes, access modifiers,aggregation,composition, inheritance, polymorphism, abstract classes, traits,

1. Discuss how polymorphism and different means of inheritance is written in your respective languages. Is there any particular difference in the manner it is written and/or their behaviors.

**C++**

Inheritance in C++ is like the usual inheritance in Java. However, in C++, it is possible to have multiple inheritance unlike the one class for extends in Java. Access types for variables and functions (public, private and protected) function identically to those in Java. An example for inheritance:

class Shape

{

protected:

int width;

int height;

public:

int getWidth()

{

return width;

}

int getHeight(int h)

{

return height;

}

};

class Rectangle: public Shape

{  
 public:  
 int getArea()  
 {   
 return (width \* height);   
 }  
};

Abstract classes and interfaces are essentially combined in C++. A class will be the usual interface if all its functions are declared with virtual & equated to 0 and variables are declared with static const. A class is made abstract if at least one function is declared with virtual & equated to 0. The difference in C++ is that you cannot instantiate objects using either of them. As such, one needs to make a derived class.

Polymorphism in C++ is concerned more about a call to a member function in a class depending on the type of object the call is used. These functions, if in base classes, will need to be declared with virtual. Concerns regarding overriding and hiding is discussed in the next question.

**C#**

C# supports the implementation of multiple interfaces, but not the extension of multiple base classes. To support polymorphism, C# makes use of the *virtual* and *override* keywords. Specifying a method or property as *virtual* in a base concrete class means that extending classes can (but do not necessarily have to) override these properties/methods. In order for an extending class to override a base class’ virtual property/method (eg. public virtual void drawShape()), the *override* keyword is added (eg. public override void drawShape()). The *virtual* modifier cannot be used with the *static, abstract, private*, or *override* modifiers. By default, properties/methods are non-virtual and cannot be overriden.

Abstract classes in C# work in almost the same way they do in Java. To create an abstract class, the *abstract* keyword is added to the class signature. Any methods that the abstract class wants to force extending classes to implement will also be marked with this keyword. Abstract methods are implicitly virtual methods. Extending classes then need to mark their implemented methods with the *override* keyword.

C# interfaces are identical to Java interfaces in functionality. All interface methods are intrinsically abstract. The methods of implementing classes no longer need to be supplied with the *abstract* keyword.

**Python**

Given the following parent class:

class Parent:

# Class methods here

The child class is written as follows:

class Child(Parent)

# Class methods here

Python does not support any type of abstract classes or methods, and simply makes use of naming conventions to inform others of the programmers’ intent.

**Scala**

Scala supports Traits, which are like Java 8’s interfaces, wherein you may define a default implementation of the interface methods. Scala also supports multiple inheritance. To resolve the diamond problem, it picks the last syntactically defined superclass. For example.

trait Orig {

def doPrint

}

trait X extends Orig {

override def doPrint {

println("X")

}

}

trait Y extends Orig {

override def doPrint {

println("Y")

}

}

class Test extends X with Y

class Test2 extends Y with X

In the sample code

var test : Orig = new Test

test.doPrint

Since the rightmost superclass in Test’s definition is Y, the doPrint method will print “Y”. On the other hand:

var test : Orig = new Test2

test.doPrint

Since the rightmost superclass in Test2’s definition is X, the doPrint method will print “X”.

1. What is the difference between a method override versus hiding? Which languages support these two concepts and how are they written?

Method overriding and method hiding have almost identical uses, both of which allow a deriving class to change the method implementation of a base class without changing the method signature. However, the main difference between the two is that if method hiding is used, the actual method called is based on the class type defined at compile-time, while in method overriding, it is based on the class type defined at run-time.

For example, take the following C# code below:

public class BaseClass   
public void WriteNum() { Console.WriteLine(12);   
public virtual void WriteStr() { Console.WriteLine("abc"); }   
}   
  
public class DerivedClass : BaseClass {   
public new void WriteNum() { Console.WriteLine(42); } // BaseClass’s WriteNum is hidden  
public override void WriteStr() { Console.WriteLine("xyz"); } // BaseClass’s WriteStr is overriden  
}

/\*...\*/

BaseClass foo = new DerivedClass();  
foo.WriteNum();   
foo.WriteStr();

Then WriteNum(), being a hidden method, will call the BaseClass’s WriteNum, and produce and output of 12, as *foo* was of type *BaseClass* at compile-time. However, WriteStr(), being overriden, will call the DerivedClass’s WriteStr, and produce an output of xyz, as *foo* was of type *DerivedClass* at run-time.

(taken from http://stackoverflow.com/questions/3838553/overriding-vs-method-hiding)

**C++**

C++ supports method overriding. This can be done by simply rewriting the method of the base class(superclass) in the derived class (subclass). Hiding exists in C++. It is illustrated here:

class TestA {

public:

void test() {}

void test(int i) {}

};

class TestB : public TestA

{

public:

void test(int i) {}

};

int main() {

TestB\* test = new TestB();

test->test();

}

In here, calling the test function with no arguments will fail as that function is hidden in class TestB. Hiding occurs when at least one method from overloaded methods (2 or more) in a class is overwritten in a derived class. The override causes those other overload methods to be hidden.

**C#**

C# supports both method overriding and method hiding. In order to a method m2 to override a method m1, m1 must be declared as *virtual* while m2 must be declared as *override* (as in #8). However, C# also supports method hiding by declaring m1 as a regular function without any other keywords, and declaring m2 as a *new* function.

**Python**

Python supports overriding, which is done by simply rewriting the method of the child class.

**Scala**

Scala supports method overriding. By declaring a subclass’ implementation with the override keyword

e.g.

override def sampleMethod : Int = {

}

, the superclass’ implementation is overridden.

1. Identify and list the difference in access modifiers supported by the languages. What is the main purpose of the access modifiers and why have they limited access to these sets.

**C++**

C++ has 3 access modifiers: public, private and protected. They function like in the use of these access modifiers in standard OOP.

**C#**

C# supports the standard OOP *private* and *public* access modifiers. C# also supports *protected*, which allows the method to be called by the same class/struct, or any classes that derive that class. There is also an *internal* modifier, which allows the method to be called by any code from within the same assembly (project), but not from other assemblies.

**Python**

Python does not support access modifiers; all methods and attributes in Python are public. Often, an underscore or a double underscore are used to indicate that something is protected or private, respectively, though that is done only as a convention.

**Scala**

Scala supports private, public, and protected. Public is the default. One can also implement scope protection as follows.

package Pack1 {

package Pack2 {

class Sample {

private[Pack1] var1 = null

private[Pack2] var2 = null

}

class Sample2 {

}

}

class Sample3 {

}

}

The variable var1 is accessible in any class inside the package Pack1, which includes any class inside Pack2, namely Sample2 and Sample3. Regarding var2, however; it is only accessible by classes inside Pack2, which is just Sample2.