Scala Programming I

Scala 개요

- Scala는 하이브리드 함수형 프로그래밍 언어
- Martin Odersky에 의해 만들어짐
- 스칼라(Scala)는 Java Virtual Machine에서 실행됨. Java와 동일할 컴파일 모델을 가짐
- 스칼라의 특징
  - 객체 지향적이다
  - 함수형 언어의 특징을 가지고 있다
  - 정적(Static)한 타입이다
  - JVM에서 실행된다
  - Java 코드를 실행할 수 있다
  - 동시 처리 및 동기화 처리를 수행할 수 있다

스칼라 개요

- 자바와 비교되는 특징
  - All types are objects
  - Type inference
  - Nested Functions
  - Functions are objects
  - Domain specific language (DSL) support
  - Traits
  - Closures
  - Concurrency support inspired by Erlang
- 스칼라 웹 프레임워크
  - Lift Framework
  - Play framework
  - Bowler framework

스칼라 설치

- 자바 jdk설치(1.8이상)
  - JAVA_HOME환경 변수 설정
- 환경변수 설정
  - PATH 설정 필요

스칼라 기본 구문

- 대화식 모드
  - scala
- 스크립모드(HelloWorld.scala)

```
object HelloWorld {
  def main(args: Array[String]) {
    println("Hello, world!")
  }
}
```

- scalac HelloWorld.scala
- scala Hello

스칼라 기본 구문

- Object
  - States & behaviors
- Class
  - Template & blueprint
- Methods
  - Behavior
- Fields
  - Unique set of instance variables
- Clouse
  - Function
- Traints
  - Encapsulates method & field definitions

스칼라 기본 구문

- 식별자(Identifiers)
  - Alphanumeric Identifiers
  - Operator Identifiers
  - Mixed Identifiers
  - Literal Identifiers

- 키워드(Keywords)

abstract	case	catch	class
def	do	else	extends
false	final	finally	for
forSome	if	implicit	import
lazy	match	new	Null
object	override	package	private
protected	return	sealed	super
this	throw	trait	Try
true	type	val	Var
while	with	yield	
-	:	=	=>
<-	<:	<%	>:
#	@		

스칼라 기본 구문

- 주석(Comments)
  - /**/, //
- 구분
  - ;
- 스칼라 패키지
  - Package com.test
  - import scala.xml._
  - import scala.collection.mutable.HashMap
  - import scala.collection.immutable.{TreeMap, TreeSet}

스칼라 기본 구문

- Byte, Short, Int, Long, Float, Double, Char
- String, Boolean, Unit, Null, Nothing, Any, AnyRef
- Integral Literals
  - 0, 035, 21, 0xFFFFF, 0777L
- Floating Point Literal
- Boolean Literals
- Symbol Literals
- Character Literals
- String Literals
- Multi-Line Strings
- Null Values
- Escape Sequences
  - \b, \t, \n, \", \\, \'

스칼라 기본 문법

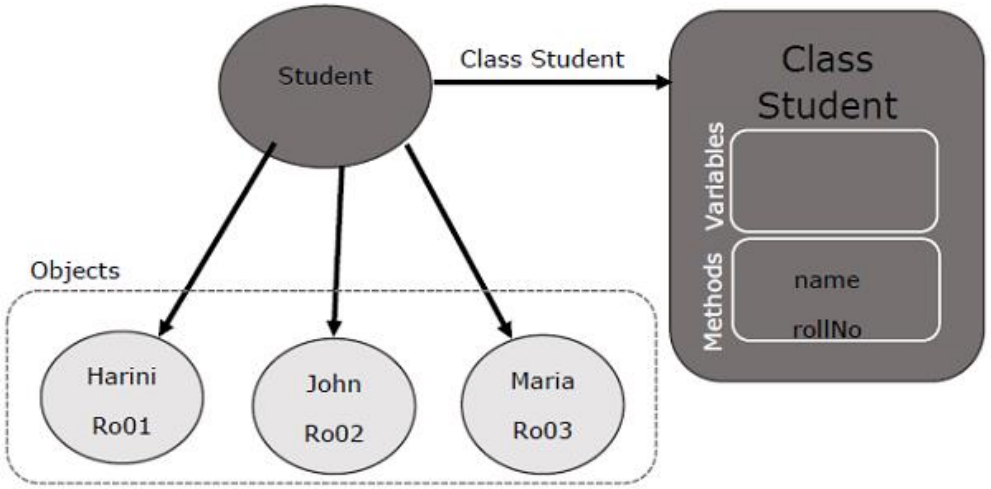
- 변수 선언
  - Var myVar : String = “Hello”
  - var 또는 val 변수 : 데이터 타입 = 초기화값
  - var myVar : Int;
  - val myVal : String;
  - var myVar = 10;
  - var myVal = “Hello”;
  - val (myVal1: Int, myVal2: String) = Pair(30, “test”)
  - val (myVar1, myVar2) = Pair(20, "test")

스칼라 기본 구문

- 변수 범위
  - 필드(Fields)
  - 메소드 파라미터스(Method Parameters)
  - 로컬 변수(Local Variables)

클래스와 객체

```
class Point(xc: Int, yc: Int) {
   var x: Int = xc
   var y: Int = yc

   def move(dx: Int, dy: Int) {
      x = x + dx
      y = y + dy
      println ("Point x location : " + x);
      println ("Point y location : " + y);
   }
}
```



클래스와 객체

- 싱글톤 객체(Singleton Objects)

```
import java.io._
class Point(val xc: Int, val yc: Int) {
  var x: Int = xc
  var y: Int = yc

  def move(dx: Int, dy: Int) {
    x = x + dx
    y = y + dy
  }
}
object Demo {
  def main(args: Array[String]) {
    val point = new Point(10, 20)
    printPoint

    def printPoint{
      println ("Point x location : " + point.x);
      println ("Point y location : " + point.y);
    }
  }
}
```

접근 수정자(Access Modifiers)

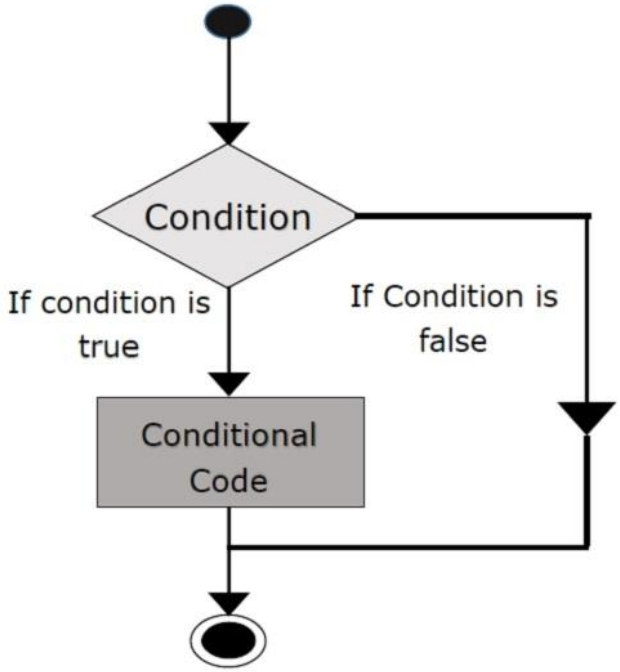
- Private Members
  - 멤버의 정의가 포함된 클래스 또는 객체 내부에서만 접근 가능함
- Protected Members
  - 멤버가 정의된 클래스의 하위 클래스에서만 접근 가능함
- Public Memebers
  - Private, Protected와 달리 모두 접근 가능하고 생략시 Public의 접근자를 가짐

연산자

- 산술 연산자(Arithmetic Operators)
  - +, -, *, /, %
- 관계 연산자(Relational Operators)
  - ==, !=, >, <, >=., <=
- 논리 연산자(Logical Operators)
  - &&, ||, !
- 비트 연산자(Bitwise Operators)
- 배정 연산자(Assignment Operators)
  - =, +=, -=, *=....
- 연산자 우선 순위

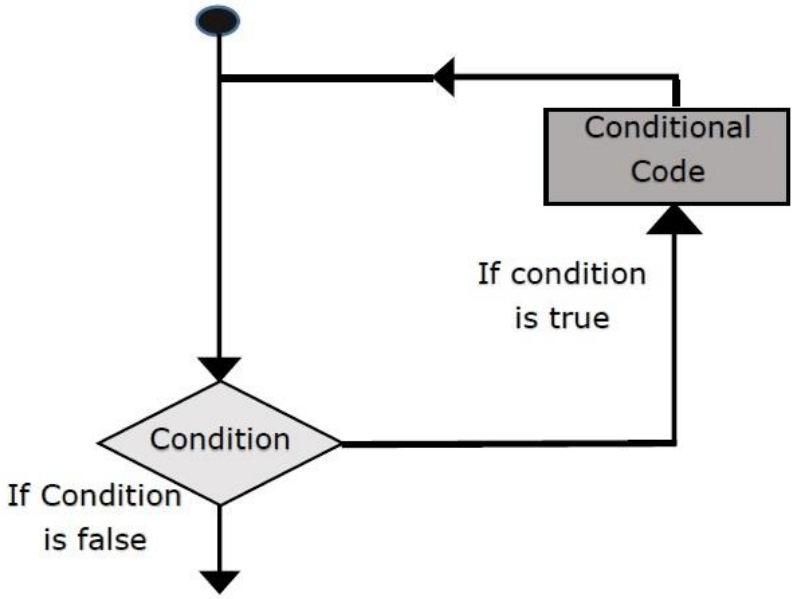
IF ELSE 구문

```
object Demo {
   def main(args: Array[String]) {
      var x = 10;

      if( x < 20 ){
         println("This is if statement");
      }
   }
}
```



Loop 구문

```
object Demo {
   def main(args: Array[String]) {
      var a = 10;

      // An infinite loop.
      while( true ){
         println( "Value of a: " + a );
      }
   }
}
```



함수(Functions)

- 로직을 블랙박스화함
- Swap기능을 가지고 있음

```
object add {
   def addInt( a:Int, b:Int ) : Int = {
      var sum:Int = 0
      sum = a + b
      return sum
   }
}
object Hello{
   def printMe( ) : Unit = {
      println("Hello, Scala!")
   }
}
```