# Introduction To Scala for Spark

# Scala ?

- 2003년 마틴 오더스키(Martin Odersky)와 EPEL 연구진이 개발
- 확장 가능한 언어 (SCAlable Language)
- 함수형 프로그래밍과 객체 지향 프로그래밍을 통합한 하이브리드 함수형 언어
- 자바가상머신을 기반으로 실행
- 특징
  - 간결한 코드와 직관성
  - 풍부한 표현식
  - 동시성에 강한 언어 (다중 쓰레드 환경에 최적화)
  - 객체지향 + 함수형 언어
  - 맥락을 읽는 언어 (컴파일러의 문맥 추론 기능)
  - 자바와의 뛰어난 연계성

# 스칼라 환경 구축

- JDK 설치 (1.8 or later)
  - 별도 문서 참고

- Scala 설치
  - download → https://www.scala-lang.org/download/
  - 명령 행 환경
    - » sbt(scala build tool)
    - » scala binaries
  - IDE
    - » IntelliJ
    - » ScalaIDE (eclipse 기반)

# JDK 설치

- 다운로드 → https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
  - 실행 중인 운영체제에 따라 선택적으로 다운로드

**Java SE Development Kit 8u221**

You must accept the Oracle Technology Network License Agreement for Oracle Java SE to download this software.

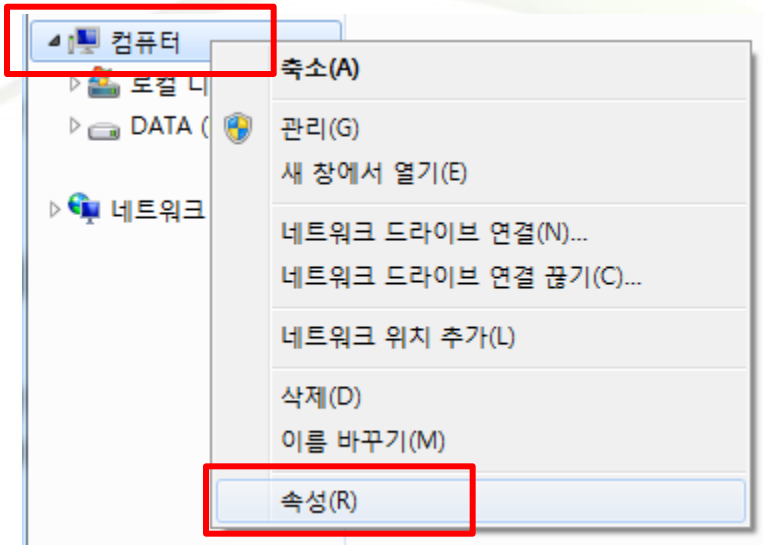
○ Accept License Agreement ◉ Decline License Agreement

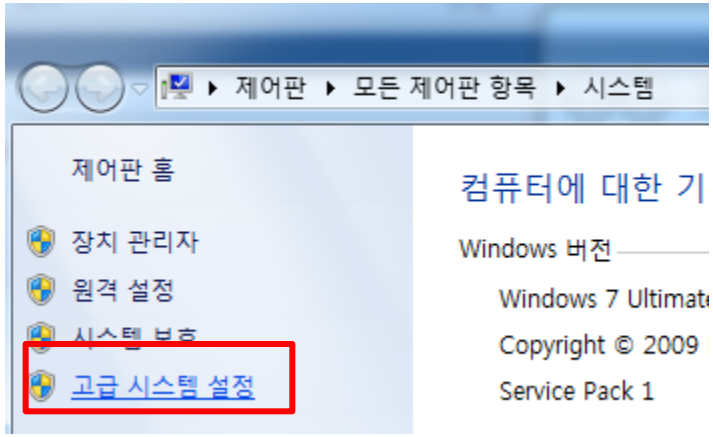
| Product / File Description | File Size | Download |
|---|---|---|
| Linux ARM 32 Hard Float ABI | 72.9 MB | jdk-8u221-linux-arm32-vfp-hflt.tar.gz |
| Linux ARM 64 Hard Float ABI | 69.81 MB | jdk-8u221-linux-arm64-vfp-hflt.tar.gz |
| Linux x86 | 174.18 MB | jdk-8u221-linux-i586.rpm |
| Linux x86 | 189.03 MB | jdk-8u221-linux-i586.tar.gz |
| Linux x64 | 171.19 MB | jdk-8u221-linux-x64.rpm |
| Linux x64 | 186.06 MB | jdk-8u221-linux-x64.tar.gz |
| Mac OS X x64 | 252.52 MB | jdk-8u221-macosx-x64.dmg |
| Solaris SPARC 64-bit (SVR4 package) | 132.99 MB | jdk-8u221-solaris-sparcv9.tar.Z |
| Solaris SPARC 64-bit | 94.23 MB | jdk-8u221-solaris-sparcv9.tar.gz |
| Solaris x64 (SVR4 package) | 133.66 MB | jdk-8u221-solaris-x64.tar.Z |
| Solaris x64 | 91.95 MB | jdk-8u221-solaris-x64.tar.gz |
| Windows x86 | 202.73 MB | jdk-8u221-windows-i586.exe |
| Windows x64 | 215.35 MB | jdk-8u221-windows-x64.exe |

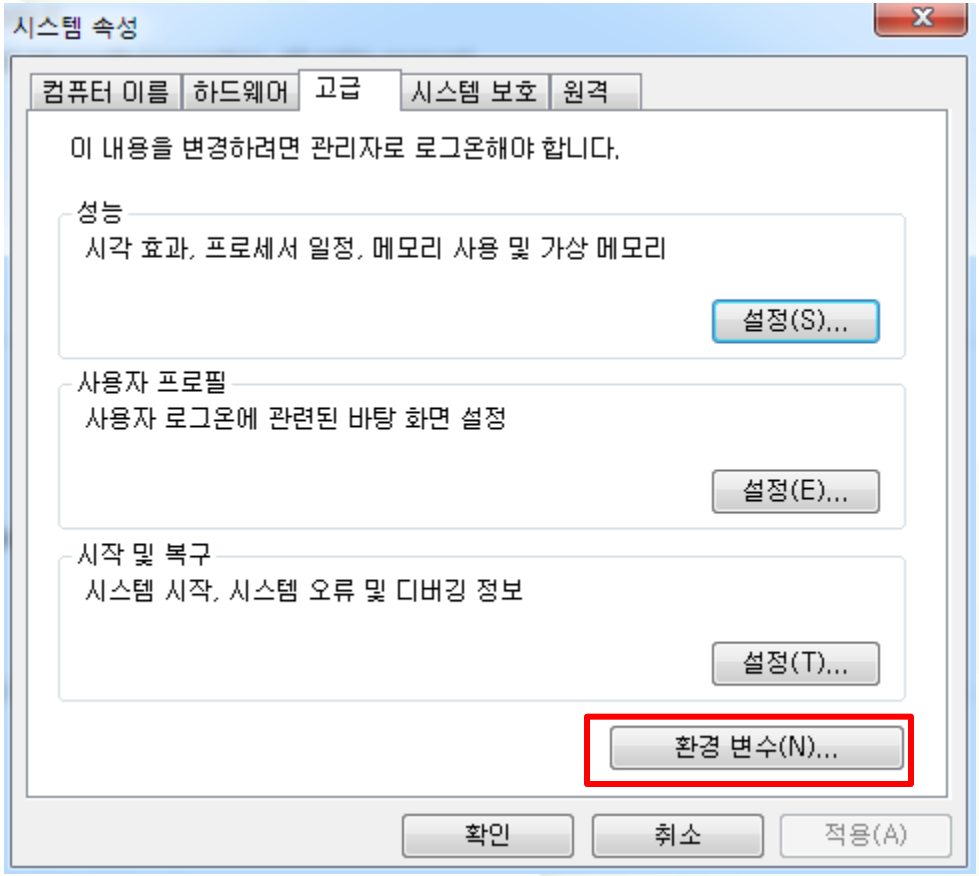
- 다운로드 후 설치 파일을 관리자 권한으로 실행해서 설치

# JDK 설치

- 환경 변수 설정 (윈도우 7 기준)

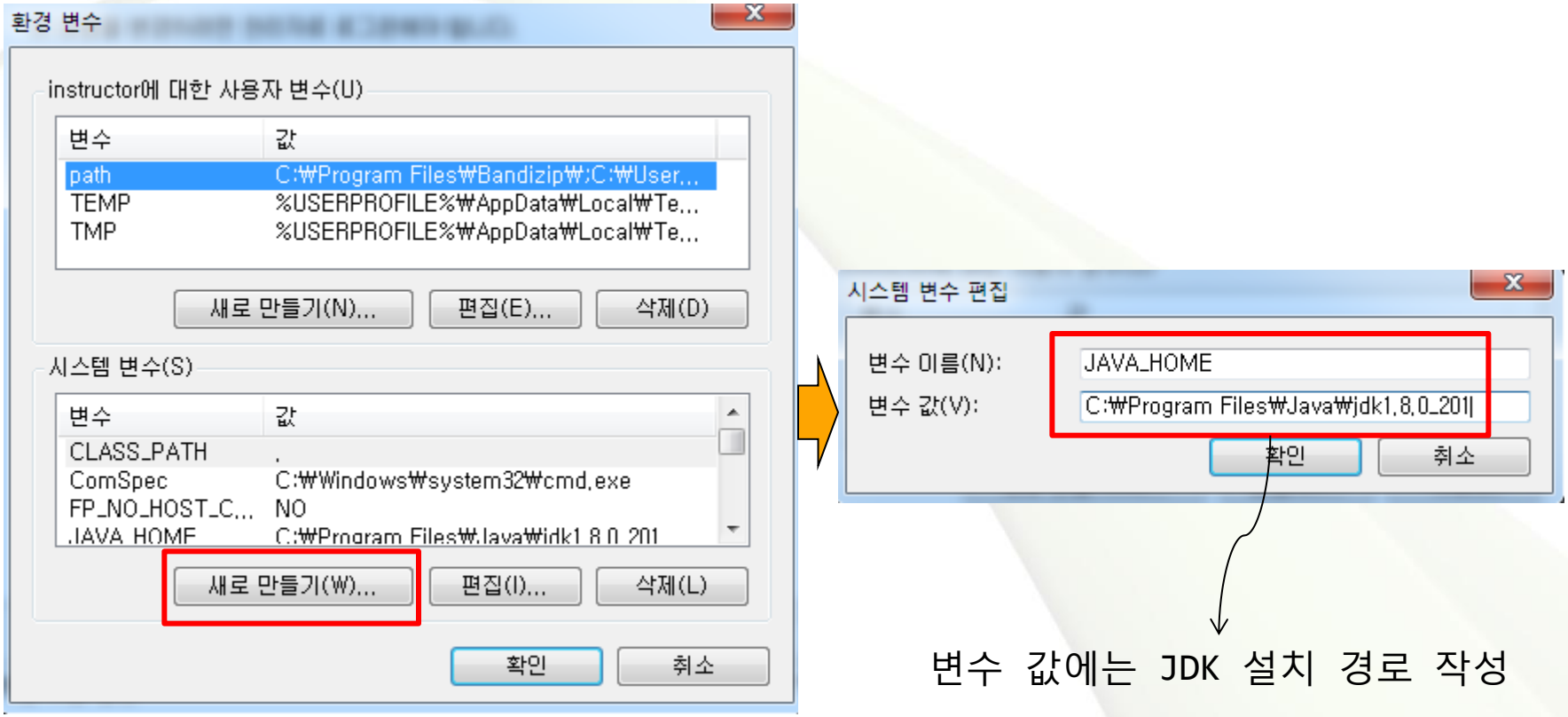


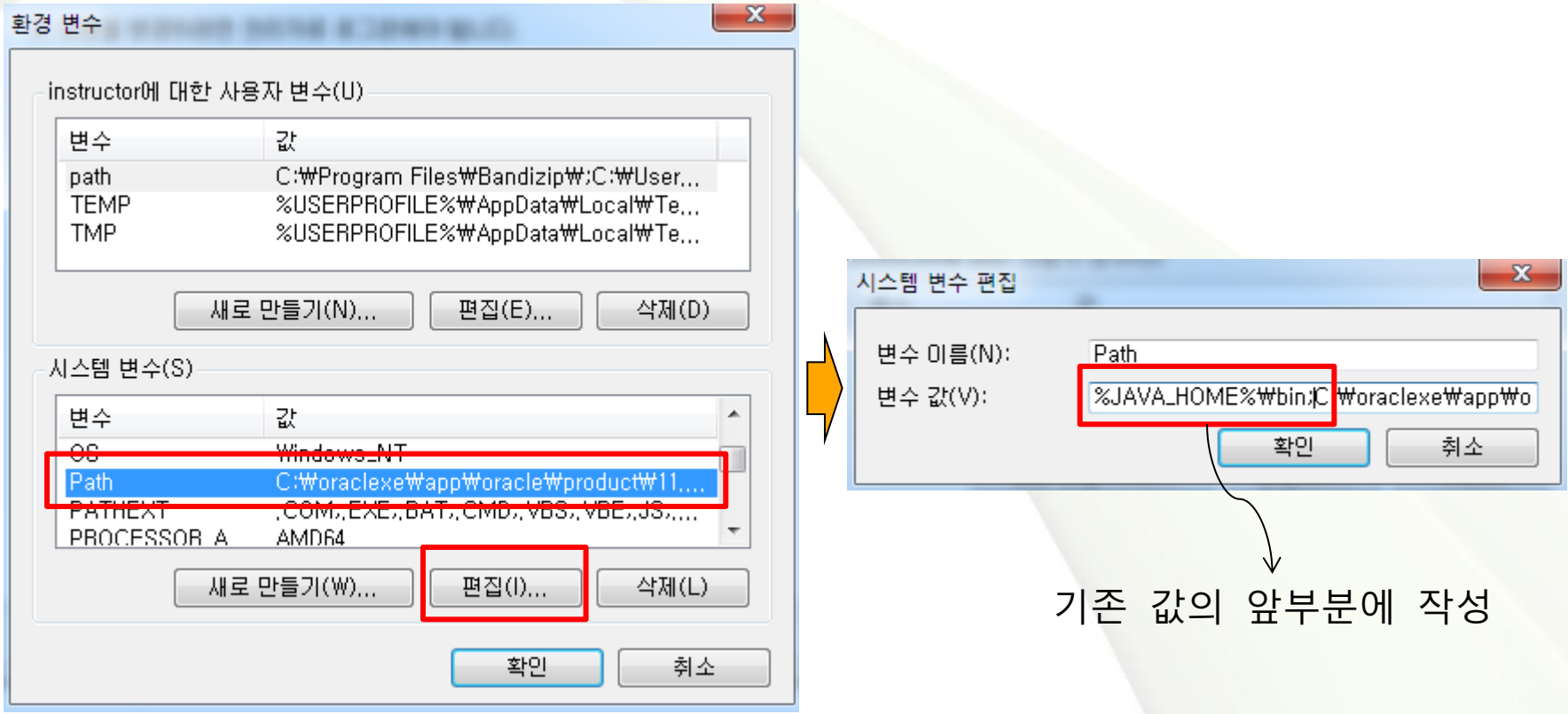
# JDK 설치

- 환경 변수 설정 (윈도우 7 기준)



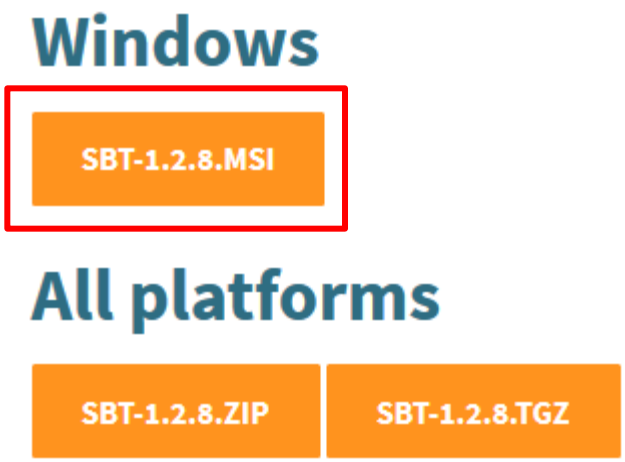
# JDK 설치

- 환경 변수 설정 (윈도우 7 기준)



# sbt 설치

- 다운로드 → https://www.scala-sbt.org/download.html




- 다운로드 후 설치 파일 실행

# Scala binaries 설치

- 다운로드 → https://www.scala-lang.org/download/


- 다운로드 후 설치 파일 실행

# Hello, World in Scala REPL

- Scala REPL 환경 실행
  - 관리자 권한으로 터미널 (명령 프롬프트) 실행
  - sbt 사용

```
> sbt console
```

  - scala binaries 사용

```
> scala
```

- 명령 입력 및 실행

```
scala> println("Hello, Scala Programming World !!!!!")
Hello, Scala Programming World !!!!!
```

- 종료

```
scala> :q

D:₩work-in-progress₩scala-workspace₩HelloWorld>
```

# Hello, World in script mode

- 소스 코드 구현 (HelloWorld.scala)

```
object HelloWorld {
   /* This will print 'Hello World' as the output */
   def main(args: Array[String]) {
      println("Hello, world!") // prints Hello World
   }
}
```

- 컴파일

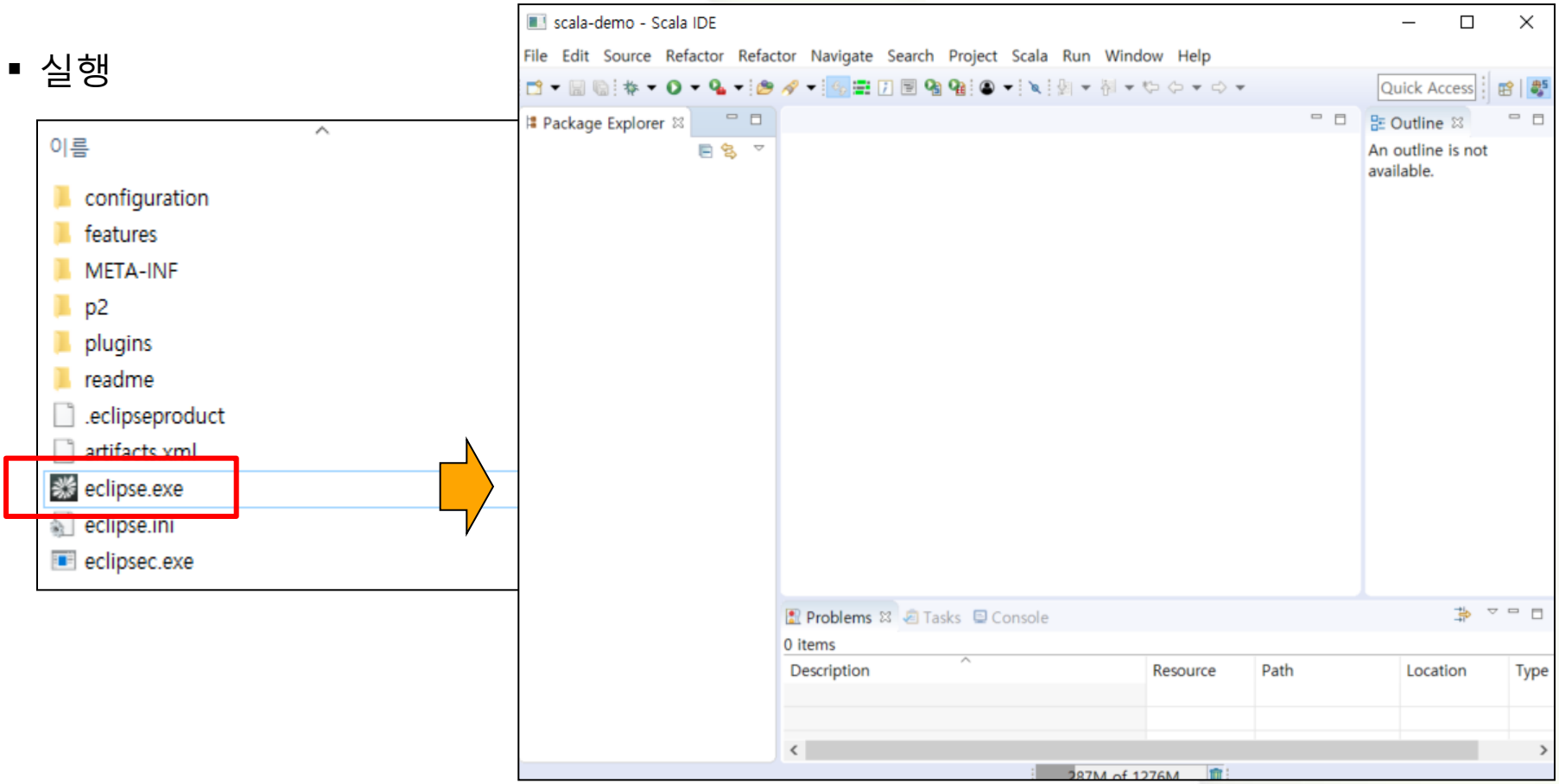
```
scalac HelloWorld.scala
```

- 실행

```
scala HelloWorld
```

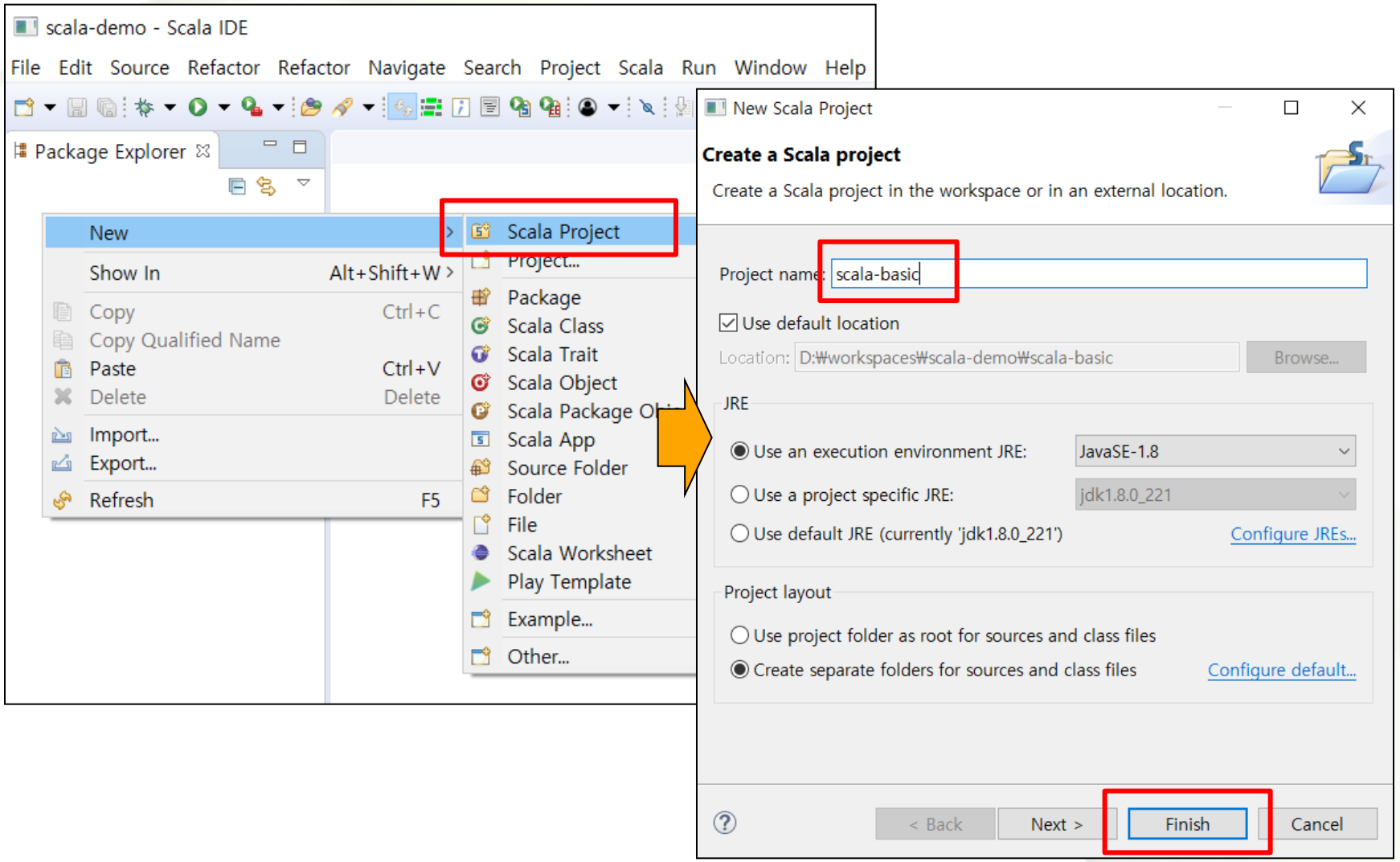
# Scala IDE

- 다운로드 → http://scala-ide.org/download/sdk.html
- 설치 → 다운로드 완료 후 압축 해제
- 실행



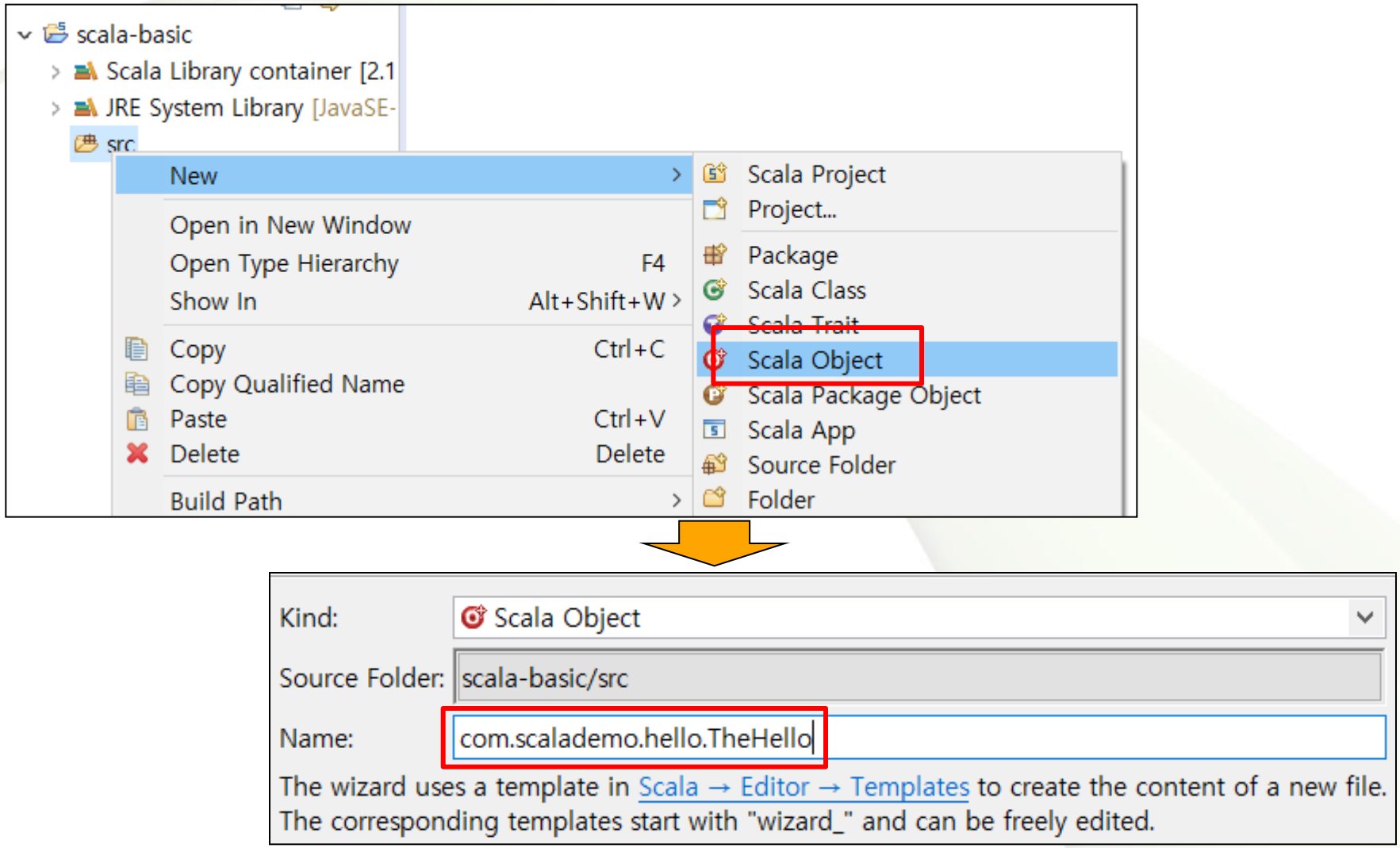
# Scala IDE

- 프로젝트 만들기



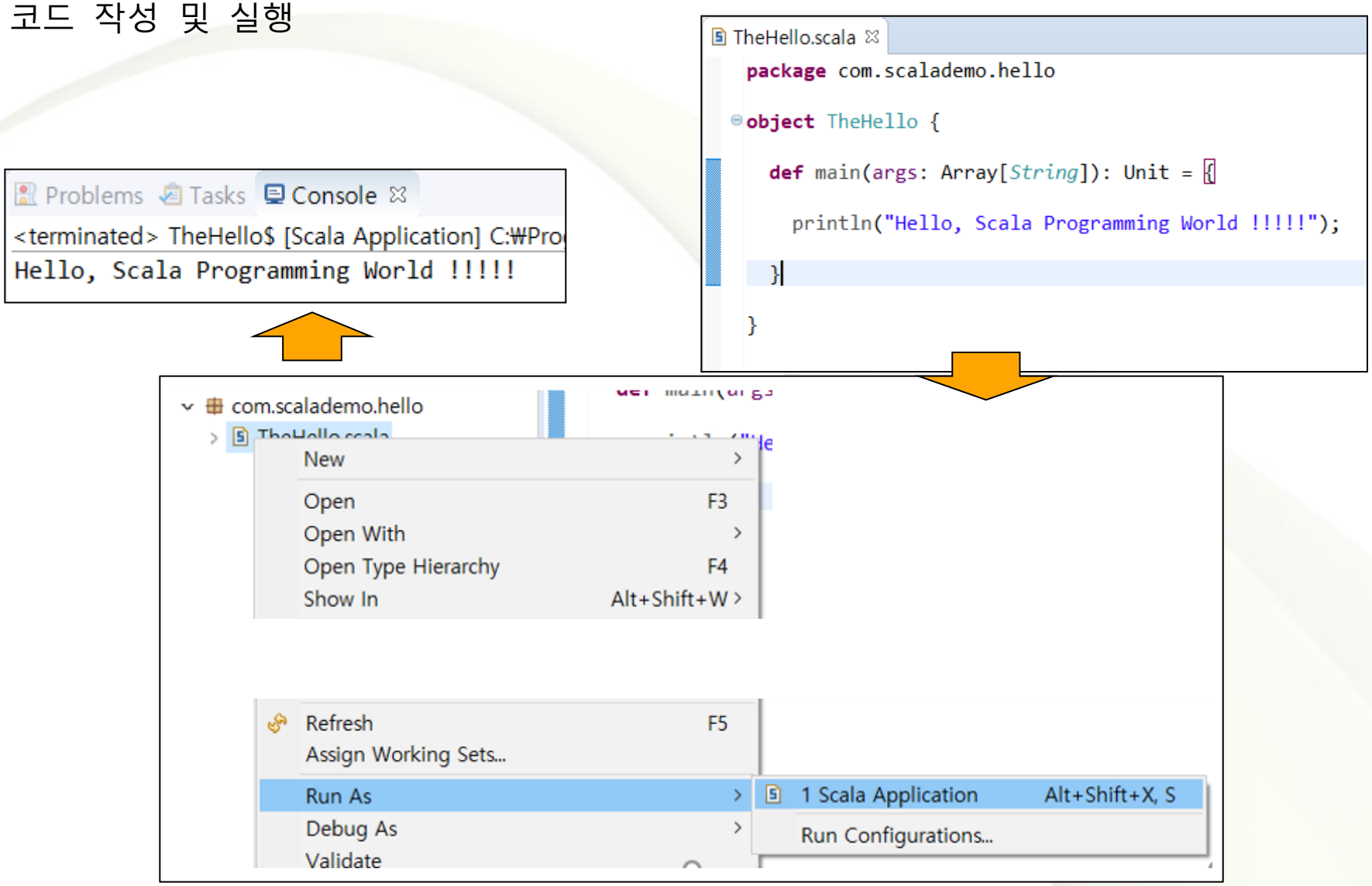
# Scala IDE

- 스칼라 객체 만들기



# Scala IDE

- 코드 작성 및 실행



# 스칼라 주요 구성 요소

| 구성 요소 | 설명 |
| --- | --- |
| object | 클래스의 인스턴스로 상태와 동작으로 구성됨 |
| class | 인스턴스를 만들기 위해 사용하는 템플릿<br>인스턴스의 동작과 상태를 정의하는 설계도(청사진) |
| Method | 객체의 동작 정의<br>데이터 조작, 실행 등의 로직 구현 도구<br>클래스는 여러 개의 메서드를 정의할 수 있음 |
| Field | 각 객체는 필드라는 고유한 인스턴스 변수 소유<br>객체의 상태를 표현하는 도구 |
| Trait | 메서드와 필드 캡슐화 도구<br>클래스는 여러 개의 Traits를 결합해서 사용<br>특정한 메서드를 지원하는 타입을 정의하기 위해 사용<br>(자바의 인터페이스) |

# 기초 구문 규칙

- 대/소문자 구분

- 명명 관행
  - Pascal Case : 클래스, object 이름 ( ex. YourName )
  - Camel Case : 메서드 이름, 변수 이름 등 ( ex. yourName )

- 프로그램 진입점
  - def main(args: Array[String]) 함수

- 주석
  - // 한 줄 주석
  - /* 영역 주석 */
  - /** 문서 주석 */

# 기초 구문 규칙

- 스칼라 키워드 (예약어)

| | | | |
|---|---|---|---|
| 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 | |
| - | : | = | => |
| <- | <: | <% | >: |
| # | @ | | |

# 기초 구문 규칙

- 선택적으로 문장의 끝을 표시하기 위해 ; 사용 → 단, 한 줄에 2개 이상의 문장을 작성하는 경우 각 문장은 ;으로 구분

- 패키지
  - 명명된 코드 모듈
  - 패키지 선언

```
package com.example.mypackage
```

  - 패키지 사용

```
import com.example.mypakage.Name
import com.example.mypackage.{ Name1, Name2 }
import com.example.mypackage._
```

# 변수

- 값을 저장하는 저장소 → 메모리 상의 이름 붙은 공간

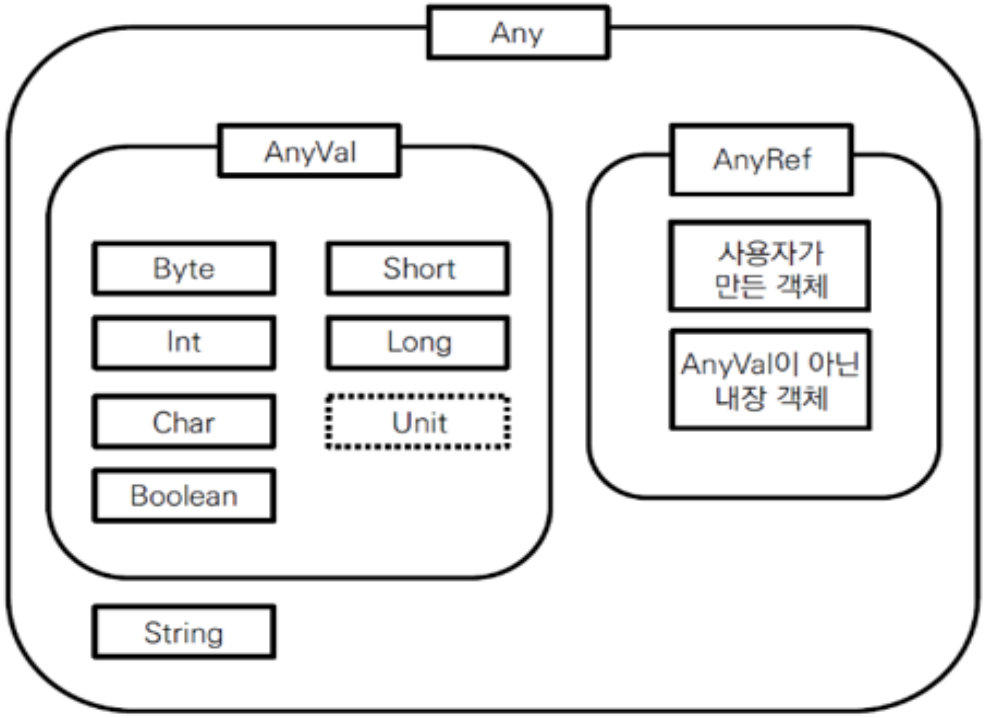
- 변수 선언

```
val or val VariableName : DataType [=  Initial Value]
```

```
var name = "John Doe"              // 이후에 값을 변경할 수 있는 저장소
val gender = "Male"                // 이후에 값을 변경할 수 없는 저장소
val age: Int = 35                  // 자료형을 명시해서 변수 선언

var phone;                         //오류 - 초기화 필수

var company = None                 //빈 값으로 초기화 ( or null )

email = "johndoe@example.com"      //오류 - var 또는 val 지정 필수
```

# 자료형

- 스칼라에서는 모든 것이 객체
  - 자바의 primitive type은 지원하지 않음
  - Int, Long 형식의 변수도 객체이고 따라서 메서드 호출 가능
  - 크게 기본 자료형과 참조 자료형으로 구분



# 자료형

- 저장 형식, 크기 등 변수(데이터)에 대한 정보

- 자바와 동일한 자료형 시스템 지원

| | Data Type & Description |
|---|---|
| Byte | 8 bit signed value. Range from -128 to 127 |
| Short | 16 bit signed value. Range -32768 to 32767 |
| Int | 32 bit signed value. Range -2147483648 to 2147483647 |
| Long | 64 bit signed value. -9223372036854775808 to 9223372036854775807 |
| Float | 32 bit IEEE 754 single-precision float |
| Double | 64 bit IEEE 754 double-precision float |
| Char | 16 bit unsigned Unicode character. Range from U+0000 to U+FFFF |
| String | A sequence of Chars |
| Boolean | Either the literal true or the literal false |
| Unit | Corresponds to no value |
| null | null reference |
| Any | The supertype of any type; any object is of type *Any* |
| AnyRef | The supertype of any reference type |
| AnyVal | The supertype of any value type |

# 리터럴

- 정수 리터럴

```
0
21
0xFFFFFFFF
```

- 부동 소수점 리터럴

```
0.0
3.14159f
1.0e100
.1
```

- Boolean 리터럴

```
true
false
```

- 문자 리터럴

```
'a'
'\u0041'
'\n'
'\t'
```

- 문자열 리터럴

```
"Hello,\nWorld!"
"String contains a \" character."
```

- 다중 행 문자열 리터럴

```
"""the present string
spans three
lines."""
```

- null

```
null
None
```

# 탈출 문자 (Escape Sequence)

| Escape Sequences | Unicode | Description |
|---|---|---|
| \b | \u0008 | backspace BS |
| \t | \u0009 | horizontal tab HT |
| \n | \u000c | formfeed FF |
| \f | \u000c | formfeed FF |
| \r | \u000d | carriage return CR |
| \" | \u0022 | double quote " |
| \' | \u0027 | single quote . |
| \\ | \u005c | backslash \ |

# 연산자

산술 연산자

| Operator | Description | Example |
|---|---|---|
| + | Adds two operands | A + B will give 30 |
| - | Subtracts second operand from the first | A - B will give -10 |
| * | Multiplies both operands | A * B will give 200 |
| / | Divides numerator by de-numerator | B / A will give 2 |
| % | Modulus Operator and remainder of after an integer division | B % A will give 0 |

관계 연산자

| Operator | Description | Example |
|---|---|---|
| == | Checks if the values of two operands are equal or not, if yes then condition becomes true. | (A == B) is not true. |
| != | Checks if the values of two operands are equal or not, if values are not equal then condition becomes true. | (A != B) is true. |
| > | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. | (A > B) is not true. |
| < | Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. | (A < B) is true. |
| >= | Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. | (A >= B) is not true. |
| <= | Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. | (A <= B) is true. |

# 연산자

논리 연산자

| Operator | Description | Example |
|---|---|---|
| && | Called Logical AND operator. If both the operands are non-zero, then condition becomes true. | (A && B) is false. |
| \|\| | Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true. | (A \|\| B) is true. |
| ! | Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false. | !(A && B) is true. |

복합 대입 연산자

| Operator | Description | Example |
|---|---|---|
| = | Simple assignment operator, Assigns values from right side operands to left side operand | C = A + B will assign value of A + B into C |
| += | Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand | C += A is equivalent to C = C + A |
| -= | Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand | C -= A is equivalent to C = C - A |
| *= | Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand | C *= A is equivalent to C = C * A |
| /= | Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand | C /= A is equivalent to C = C / A |
| %= | Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand | C %= A is equivalent to C = C % A |

# 제어문

- if ~ else 선택문

```
object Test {
   def main(args: Array[String]) {
      var x = 30;

      if( x == 10 ){
         println("Value of X is 10");
      }else if( x == 20 ){
         println("Value of X is 20");
      }else if( x == 30 ){
         println("Value of X is 30");
      }else{
         println("This is else statement");
      }
   }
}
```

```
object Test {
   def main(args: Array[String]) {
         var x = 30;
         var y = 10;

         if( x == 30 ){
            if( y == 10 ){
            println("X = 30 and Y = 10");
         }
      }
   }
}
```

# 제어문

- while 반복문

```
object Test {
   def main(args: Array[String]) {
      // Local variable declaration:
      var a = 10;

      // while loop execution
      while( a < 20 ){
         println( "Value of a: " + a );
         a = a + 1;
      }
   }
}
```

# 제어문

- do ~ while 반복문

```
object Test {
   def main(args: Array[String]) {
      // Local variable declaration:
      var a = 10;

      // do loop execution
      do{
         println( "Value of a: " + a );
         a = a + 1;
      }while( a < 20 )
   }
}
```

# 제어문

- for 반복문
  - 단순 반복문

```
object Test {
   def main(args: Array[String]) {
      var a = 0;
      // for loop execution with a range
      for( a <- 1 until 10){
         println( "Value of a: " + a );
      }
   }
}
```

  - 중첩 반복문

```
object Test {
   def main(args: Array[String]) {
      var a = 0;
      var b = 0;
      // for loop execution with a range
      for( a <- 1 to 3; b <- 1 to 3){
         println( "Value of a: " + a );
         println( "Value of b: " + b );
      }
   }
}
```

# 제어문

- for 반복문
  - 컬렉션 반복문

```
object Test {
   def main(args: Array[String]) {
      var a = 0;
      val numList = List(1,2,3,4,5,6);

      // for loop execution with a collection
      for( a <- numList ){
         println( "Value of a: " + a );
      }
   }
}
```

  - 반복문에 필터 사용

```
object Test {
   def main(args: Array[String]) {
      var a = 0;
      val numList = List(1,2,3,4,5,6,7,8,9,10);

      for( a <- numList
           if a != 3; if a < 8 ){
         println( "Value of a: " + a );
      }
   }
}
```

# 제어문

- break 문

```
import scala.util.control._

object Test {
   def main(args: Array[String]) {
      var a = 0;
      val numList = List(1,2,3,4,5,6,7,8,9,10);

      val loop = new Breaks;
      loop.breakable {
         for( a <- numList){
            println( "Value of a: " + a );
            if( a == 4 ){
               loop.break;
            }
         }
      }
      println( "After the loop" );
   }
}
```

# 함수?

- 재사용 가능한 실행문 집합
  - 선언 : 함수 만들기
  - 호출 : 만들어진 함수 사용(실행)하기

- 형식

```
def functionName ([list of parameters]) : [return type] = {
   function body
   return [expr]
}
```

- 선언 ~ 호출 사례

```
object Test {
   def main(args: Array[String]) {
        println( "Returned Value : " + addInt(5,7) );
   }
   def addInt( a:Int, b:Int ) : Int = {
      var sum:Int = 0
      sum = a + b

      return sum
   }
}
```

# 이름으로 함수 호출 (Call by Name)

- 함수를 다른 함수의 매개변수로 전달 → 호출된 함수 내에서 전달된 함수 사용

```
object Test {
   def main(args: Array[String]) {
        delayed(time());
   }

   def time() = {
      println("Getting time in nano seconds")
      System.nanoTime
   }
   def delayed( t: => Long ) = {
      println("In delayed method")
      println("Param: " + t)
      t
   }
}
```

# 전달인자 활용

- 명명된 전달인자 → 함수를 호출할 때 전달인자 이름을 명시해서 값 전달

```
object Test {
   def main(args: Array[String]) {
        printInt(b=5, a=7);
   }
   def printInt( a:Int, b:Int ) = {
      println("Value of a : " + a );
      println("Value of b : " + b );
   }
}
```

- 가변 전달인자 → 전달인자의 개수를 미리 지정하지 않고 함수 정의

```
object Test {
   def main(args: Array[String]) {
        printStrings("Hello", "Scala", "Python");
   }
   def printStrings( args:String* ) {
      var i : Int = 0;
      for( arg <- args ){
         println("Arg value[" + i + "] = " + arg );
         i = i + 1;
      }
   }
}
```

# 전달인자 활용

- 전달인자에 기본 값 지정 → 호출할 때 제공하지 않으면 기본 값 사용

```
object Test {
   def main(args: Array[String]) {
        println( "Returned Value : " + addInt() );
   }
   def addInt( a:Int=5, b:Int=7 ) : Int = {
      var sum:Int = 0
      sum = a + b

      return sum
   }
}
```

# 특별한 형식의 함수

- 중첩 함수 → 함수 내부에 선언된 함수 (선언된 함수 내부에서만 사용)

```
object Test {
   def main(args: Array[String]) {
      println( factorial(0) )
      println( factorial(1) )
      println( factorial(2) )
      println( factorial(3) )
   }

   def factorial(i: Int): Int = {
      def fact(i: Int, accumulator: Int): Int = {
         if (i <= 1)
            accumulator
         else
            fact(i - 1, i * accumulator)
      }
      fact(i, 1)
   }
}
```

# 특별한 형식의 함수

- 익명 함수
  - => 기호를 사용해서 이름 없는 함수 정의
  - 변수에 저장하거나 다른 함수의 전달인자로 사용

```
var inc = (x:Int) => x+1
var x = inc(7)-1
var mul = (x: Int, y: Int) => x*y
println(mul(3, 4))
var userDir = () => { System.getProperty("user.dir") }
println( userDir )
```

# 특별한 형식의 함수

- 부분 적용 함수
  - 다른 함수의 호출 구문에서 일부 전달인자를 비워 두고 정의하는 함수
  - 일부 전달인자를 고정한 상태로 원 함수를 호출하는 효과

```
import java.util.Date

object Test {
    def main(args: Array[String]) {
        val logWithDateBound = log(new Date, _ : String)

            logWithDateBound("message1" )
        logWithDateBound("message2" )
        logWithDateBound("message3" )
    }

    def log(date: Date, message: String)  = {
      println(date + "----" + message)
    }
}
```

# 특별한 형식의 함수

- currying 함수
  - 전달인자를 분할해서 함수 체인처럼 사용하는 함수

```
def strcat(s1: String)(s2: String) = s1 + s2
```

```
strcat("foo")("bar")
```

```
object Test {
   def main(args: Array[String]) {
      val str1:String = "Hello, "
      val str2:String = "Scala!"
      println( "str1 + str2 = " + strcat(str1)(str2) )
   }

   def strcat(s1: String)(s2: String) =
      s1 + s2
   }
}
```

# 클래스와 객체 (Classes & Object)

- 클래스는 객체를 만드는 데 사용하는 설계도
- 객체는 실제 메모리에 만들어지는 인스턴스
- 하나의 클래스로 다수의 객체 생성

```
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);
   }
}
```

```
object Test {
   def main(args: Array[String]) {
      val pt = new Point(10, 20);

      // Move to a new location
      pt.move(10, 10);
   }
}
```

# 싱글톤 객체 (Singleton Object)

- 특정 클래스에 대해 오직 하나의 객체만 만들어서 사용하는 패턴

- Scala에서는 object 키워드로 클래스를 선언하면 클래스 선언과 객체 생성이 통합됨

```
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);
   }
}
```

```
object Test {
   def main(args: Array[String]) {
      val pt = new Point(10, 20);

      // Move to a new location
      pt.move(10, 10);
   }
}
```

# 클래스 상속 (확장)

- 다른 클래스의 내용을 가져와서 새로운 내용을 추가하거나 기존 클래스의 내용을 변경해서 사용하는 기법

```
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);
  }
}

class Location(xc: Int,yc: Int,val zc: Int) extends Point(xc, yc) {
  var z: Int = zc

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

```
object Test {
   def main(args: Array[String]) {
      val loc = new Location(10, 20, 15);

      // Move to a new location
      loc.move(10, 10, 5);
   }
}
```

# 트레이트 (Traits)

- 다른 클래스에 추가되어 사용되는 특성 정의
  - 트레이트의 인스턴스를 만들 수는 없음
  - 트레이트가 추가된 클래스는 반드시 트레이트의 특성을 구현해야 함

```
trait Equal {
  def isEqual(x: Any): Boolean
  def isNotEqual(x: Any): Boolean = !isEqual(x)
}
```

```
class Point(xc: Int, yc: Int) extends Equal {
  var x: Int = xc
  var y: Int = yc
  def isEqual(obj: Any) =
    obj.isInstanceOf[Point] &&
    obj.asInstanceOf[Point].x == x
}
```

```
object Test {
   def main(args: Array[String]) {
      val p1 = new Point(2, 3)
      val p2 = new Point(2, 4)
      val p3 = new Point(3, 3)

      println(p1.isNotEqual(p2))
      println(p1.isNotEqual(p3))
      println(p1.isNotEqual(2))
   }
}
```

# 패턴 매칭

- 특정한 형태의 객체 또는 값을 기반으로 코드를 분기해서 실행하는 기능 제공

- switch ~ case의 발전된 형태

- 기본 자료형 패턴 매칭

```
object Test {
   def main(args: Array[String]) {
      println(matchTest("two"))
      println(matchTest("test"))
      println(matchTest(1))

   }
   def matchTest(x: Any): Any = x match {
      case 1 => "one"
      case "two" => 2
      case y: Int => "scala.Int"
      case _ => "many"
   }
}
```

# 패턴 매칭

- 객체 패턴 매칭

```
object Test {
   def main(args: Array[String]) {
         val alice = new Person("Alice", 25)
            val bob = new Person("Bob", 32)
         val charlie = new Person("Charlie", 32)

      for (person <- List(alice, bob, charlie)) {
         person match {
            case Person("Alice", 25) => println("Hi Alice!")
            case Person("Bob", 32) => println("Hi Bob!")
            case Person(name, age) =>
               println("Age: " + age + " year, name: " + name + "?")
         }
      }
   }
   // case class, empty one.
   case class Person(name: String, age: Int)
}
```

- case class는 패턴 매칭 이외에도 다양하게 활용

# 배열

- 동일 형식의 고정 크기 데이터 집합

- 배열 만들기 및 요소 접근

```
var z:Array[String] = new Array[String](3)

or

var z = new Array[String](3)
```

```
z(0) = "Zara"; z(1) = "Nuha"; z(4/2) = "Ayan"
```

```
var z = Array("Zara", "Nuha", "Ayan")
```

사용사례

```
object Test {
   def main(args: Array[String]) {
      var myList = Array(1.9, 2.9, 3.4, 3.5)

      // Print all the array elements
      for ( x <- myList ) {
         println( x )
      }

      // Summing all elements
      var total = 0.0;
      for ( i <- 0 to (myList.length - 1)) {
         total += myList(i);
      }
      println("Total is " + total);

      // Finding the largest element
      var max = myList(0);
      for ( i <- 1 to (myList.length - 1) ) {
         if (myList(i) > max) max = myList(i);
      }
      println("Max is " + max);

   }
}
```

# 다차원 배열

- 다른 배열을 요소로 하는 배열

- 다차원 배열 만들기 및 사용

```
import Array._

object Test {
   def main(args: Array[String]) {
      var myMatrix = ofDim[Int](3,3)

      // build a matrix
      for (i <- 0 to 2) {
         for ( j <- 0 to 2) {
            myMatrix(i)(j) = j;
         }
      }

      // Print two dimensional array
      for (i <- 0 to 2) {
         for ( j <- 0 to 2) {
            print(" " + myMatrix(i)(j));
         }
         println();
      }

   }
}
```

# 배열 활용

- 배열 결합

```
import Array._

object Test {

   def main(args: Array[String]) {
      var myList1 = Array(1.9, 2.9, 3.4, 3.5)
      var myList2 = Array(8.9, 7.9, 0.4, 1.5)

      var myList3 =  concat( myList1, myList2)

      // Print all the array elements
      for ( x <- myList3 ) {
         println( x )
      }
   }
}
```

범위로 배열 만들기

```
import Array.

object Test {
   def main(args: Array[String]) {
      var myList1 = range(10, 20, 2)
      var myList2 = range(10,20)

      // Print all the array elements
      for ( x <- myList1 ) {
         print( " " + x )
      }
      println()
      for ( x <- myList2 ) {
         print( " " + x )
      }
   }
}
```

# 컬렉션

- 다수의 데이터를 효과적으로 관리하는 자료구조와 알고리즘 구현 클래스 집합

- 종류

| SN | Collections with Description |
|---|---|
| 1 | Scala Lists<br>Scala's List[T] is a linked list of type T. |
| 2 | Scala Sets<br>A set is a collection of pairwise different elements of the same type. |
| 3 | Scala Maps<br>A Map is a collection of key/value pairs. Any value can be retrieved based on its key. |
| 4 | Scala Tuples<br>Unlike an array or list, a tuple can hold objects with different types. |

# 리스트 (Lists)

- 변경 불가능한 연결 리스트

- 리스트 객체 만들기 및 사용

```
// List of Strings
val fruit: List[String] = List("apples", "oranges", "pears")

// List of Integers
val nums: List[Int] = List(1, 2, 3, 4)

// Empty List.
val empty: List[Nothing] = List()

// Two dimensional list
val dim: List[List[Int]] =
   List(
      List(1, 0, 0),
      List(0, 1, 0),
      List(0, 0, 1)
   )
```

```
object Test {
   def main(args: Array[String]) {
      val fruit = "apples" :: ("oranges" :: ("pears" :: Nil))
      val nums = Nil

      println( "Head of fruit : " + fruit.head )
      println( "Tail of fruit : " + fruit.tail )
      println( "Check if fruit is empty : " + fruit.isEmpty )
      println( "Check if nums is empty : " + nums.isEmpty )
   }
}
```

# 셋(Sets)

- 중복 요소를 허용하지 않는 데이터 집합

- 셋 만들기 및 사용

```
// Empty set of integer type
var s : Set[Int] = Set()

// Set of integer type
var s : Set[Int] = Set(1,3,5,7)

or

var s = Set(1,3,5,7)
```

```
object Test {
   def main(args: Array[String]) {
      val fruit = Set("apples", "oranges", "pears")
      val nums: Set[Int] = Set()

      println( "Head of fruit : " + fruit.head )
      println( "Tail of fruit : " + fruit.tail )
      println( "Check if fruit is empty : " + fruit.isEmpty )
      println( "Check if nums is empty : " + nums.isEmpty )
   }
}
```

# 맵(Maps)

- key - value 형식의 데이터 집합
  - key는 고유한 값으로 구성 (Set), value는 중복허용 (List)

- 맵 만들기 및 사용

```
// Empty hash table whose keys are strings and values are integers:
var A:Map[Char,Int] = Map()

// A map with keys and values.
val colors = Map("red" -> "#FF0000", "azure" -> "#F0FFFF")
```

```
object Test {
   def main(args: Array[String]) {
      val colors = Map("red" -> "#FF0000",
                       "azure" -> "#F0FFFF",
                       "peru" -> "#CD853F")

      val nums: Map[Int, Int] = Map()

      println( "Keys in colors : " + colors.keys )
      println( "Values in colors : " + colors.values )
      println( "Check if colors is empty : " + colors.isEmpty )
      println( "Check if nums is empty : " + nums.isEmpty )
   }
}
```

# 튜플 (Tuples)

- 변경 불가능한 서로 다른 종류의 데이터 집합 (형식을 미리 정의하지 않고 사용하는 객체)

- 만들기 및 사용

```
val t = (1, "hello", Console)

val t = (4,3,2,1)

val sum = t._1 + t._2 + t._3 + t._4
```

```
object Test {
   def main(args: Array[String]) {
      val t = (4,3,2,1)

      val sum = t._1 + t._2 + t._3 + t._4

      println( "Sum of elements: "  + sum )
   }
}
```

# Options

- 0(None) 또는 특정 타입 데이터 한 개(Some([T])를 가지고 있는 형식

- Option 사용
  - Map에 포함된 데이터를 key로 검색하면 있는 경우 Some을 없는 경우 None을 반환

```
object Test {
   def main(args: Array[String]) {
      val capitals = Map("France" -> "Paris", "Japan" -> "Tokyo")

      println("show(capitals.get( \"Japan\")) : " +
                                          show(capitals.get( "Japan")) )
      println("show(capitals.get( \"India\")) : " +
                                          show(capitals.get( "India")) )
   }

   def show(x: Option[String]) = x match {
      case Some(s) => s
      case None => "?"
   }
}
```

# 함수 컴비네이터 (Combinator)

- 구현된 코드 논리에 따라 컬렉션을 변형한 후 동일한 자료형의 컬렉션을 반환하는 함수

```
def main(args: Array[String]) {

  val o = List(1, 2, 3, 4)
  val oo = List(5, 6, 7, 8)
  val ooo = List(o, oo)

  println(o)

  val n = o.map(i => i * 10)
  println(n)

  o.foreach(i => i * 20)

  val n2 = o.filter(i => i >= 3).map(i => i * 2)
  println(n2)

  val n3 = o.foldLeft(0)( (i, j) => i + j )
  println(n3)

  val n4 = o.partition(i => i % 2 == 0)
  println(n4)

  val n5 = o zip oo
  println(n5)
  println(n5.unzip)
```

```
    val n6 = o.find( i => i >= 2 ) // find first element
    println(n6)

    val n7 = o.drop(2) // remove 0 ~ 2 elements
    println(n7)

    val n8 = o.dropWhile( i => i < 3 ) // remove element if condition true
    println(n8)

    val n9 = ooo.flatten
    println(n9)

  }
```

# 예외 처리

- 예외 발생

```
throw new IllegalArgumentException
```

- 예외 처리

```
import java.io.FileReader
import java.io.FileNotFoundException
import java.io.IOException

object Test {
   def main(args: Array[String]) {
      try {
         val f = new FileReader("input.txt")
      } catch {
         case ex: FileNotFoundException => {
            println("Missing file exception")
         }
         case ex: IOException => {
            println("IO Exception")
         }
      } finally {
         println("Exiting finally...")
      }
   }
}
```