02장 – 변수와 자료형, 연산자

Copyright Note

- Book: Do it! 코틀린 프로그래밍 / 이지스퍼블리싱
- Author: 황영덕 (Youngdeok Hwang; sean.ydhwang@gmail.com)
- Update Date: 10-July-2019
- Issue Date: 25-Nov-2017
- Slide Revision #: rev02
- Homepage: acaroom.net
- Distributor: 이지스퍼블리싱 (담당: 박현규)

02 변수와 자료형, 연산자

02-1 코틀린 패키지

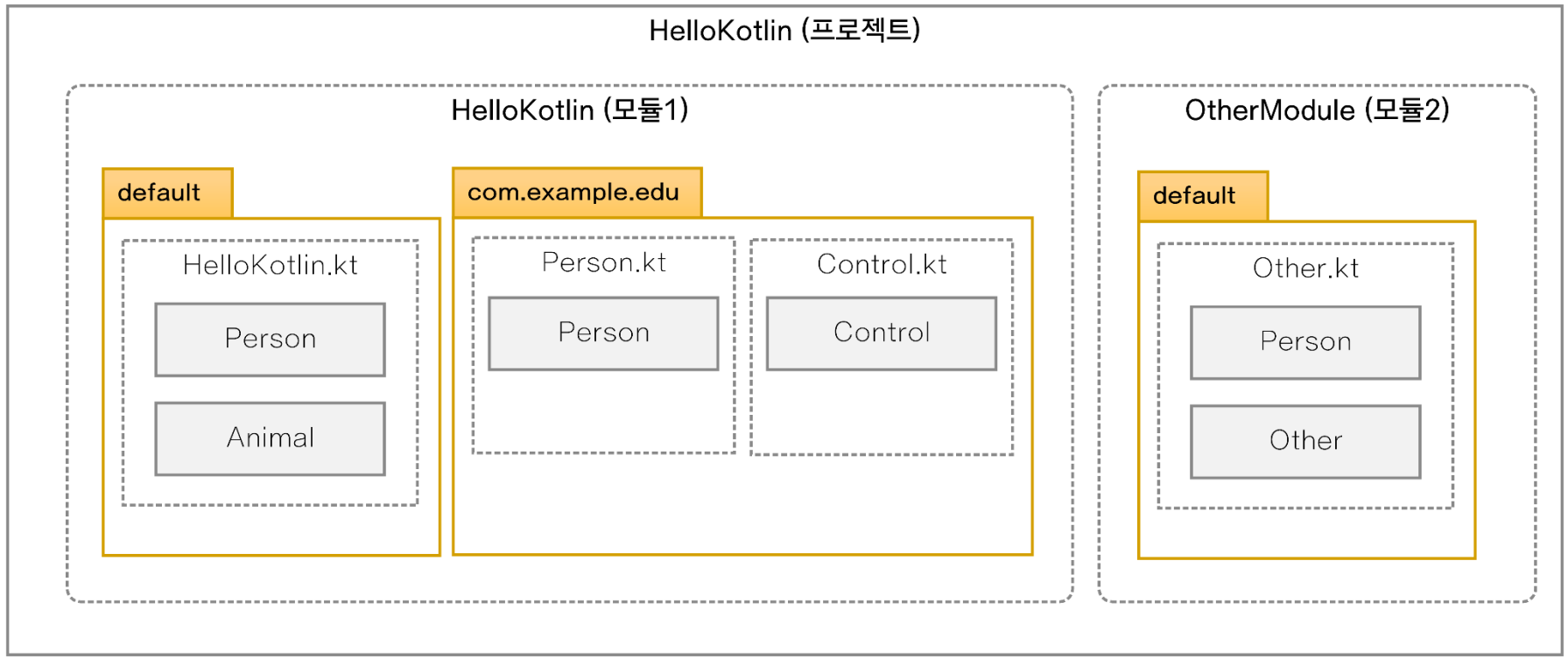
02-2 변수와 자료형

02-3 자료형 검사와 변환

02-4 코틀린 연산자

프로젝트, 모듈, 패키지, 파일의 관계

- 프로젝트와 패키지



패키지 정의

패키지 정의

- 자바 프로젝트처럼 디렉터리와 매치 되어야 하지는 않는다.

```
package com.example.edu

class Person(val name: String, val age: Int)

...
```

- 패키지를 지정하지 않으면 이름이 없는 기본(default) 패키지에 속한다.
- import의 이름이 충돌하면 as 키워드로 로컬에서 사용할 이름을 변경해서 충돌을 피할 수 있다.

```
import com.example.edu.Person
import com.example.edu.Person as User // com.example.edu.Person을 User로 지정
```

- import는 클래스 뿐만 아니라 다른것도 임포트 할 수 있다.
  - 최상위레벨 함수와 프로퍼티, 오브젝트 선언의 함수와 프로퍼티, 열거형 상수

기본 패키지 (코틀린 표준 라이브러리)

- kotlin-stdlib-sources.jar

패키지 이름	설명
kotlin.*	Any, Int, Double 등 코어 함수와 자료형
kotlin.text.*	문자와 관련된 API
kotlin.sequences.*	컬렉션 자료형의 하나로 반복이 허용되는 개체를 열거
kotlin.ranges.*	If문이나 for문에서 사용할 범위 관련 요소
kotlin.io.*	입출력 관련 API
kotlin.collections.*	List, Set, Map 등의 컬렉션
kotlin.annotation.*	애노테이션 관련 API

02 변수와 자료형, 연산자

02-1 코틀린 패키지

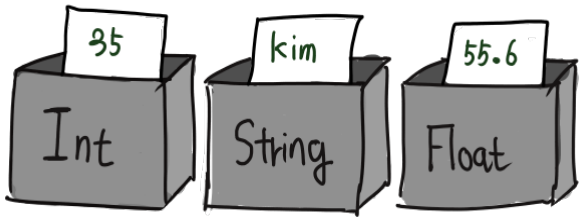
02-2 변수와 자료형

02-3 자료형 검사와 변환

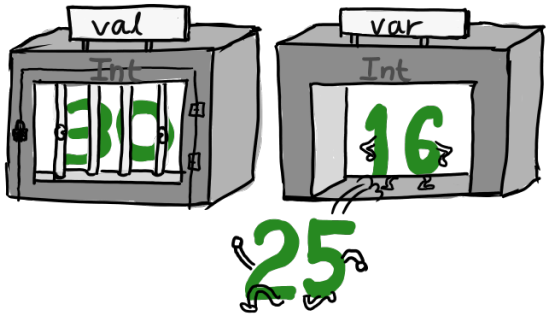
02-4 코틀린 연산자

자료형과 변수

- 자료형
  - Int
  - String
  - Float



- 변수
  - val (value) - 불변형
  - var (variable) - 가변형



변수의 선언

```
val username: String = "Kildong"
```

선언 키워드 | 변수 이름 | 자료형 | 값

변수 선언

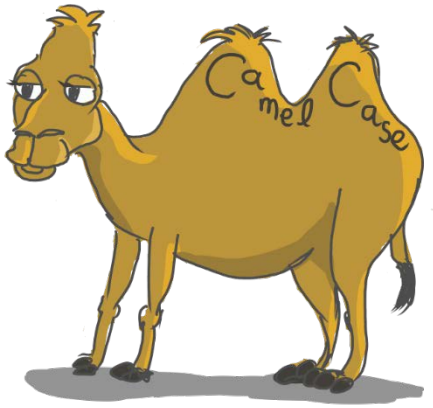
변수 선언 예

- `val username = "Kildong" // 자료형을 추론하여 String으로 결정`
- `var username   // 자료형을 지정하지 않은 변수는 사용할 수 없다`
- `val init: Int   // 사용전 혹은 생성자 시점에서 init변수를 초기화 해야함`
- `val number = 10  // number 변수는 Int형으로 추론`

```
- 변수 이름은 123abc와 같이 숫자로 시작하면 안 됩니다.
- 변수 이름에는 while, if와 같이 코틀린에서 사용되는 키워드는 사용할 수 없습니다.
- 변수 이름은 의미 있는 단어를 사용하여 만드는 것이 좋습니다.
- 여러 단어를 사용하여 변수 이름을 지을 경우 카멜 표기법(Camel Expression)을 사용하세요.
```

변수명

- 일반 변수, 함수명 등 (단봉 낙타와 같은 카멜 표기법)
  - `camelCase`
  - `numberOfBooks`
  - `myFirstNumber`



- 클래스, 인터페이스 등 (쌍봉 낙타 혹은 파스칼 표기법)
  - `AnimalCategory`
  - `CarEngine`

자료형 알아보기

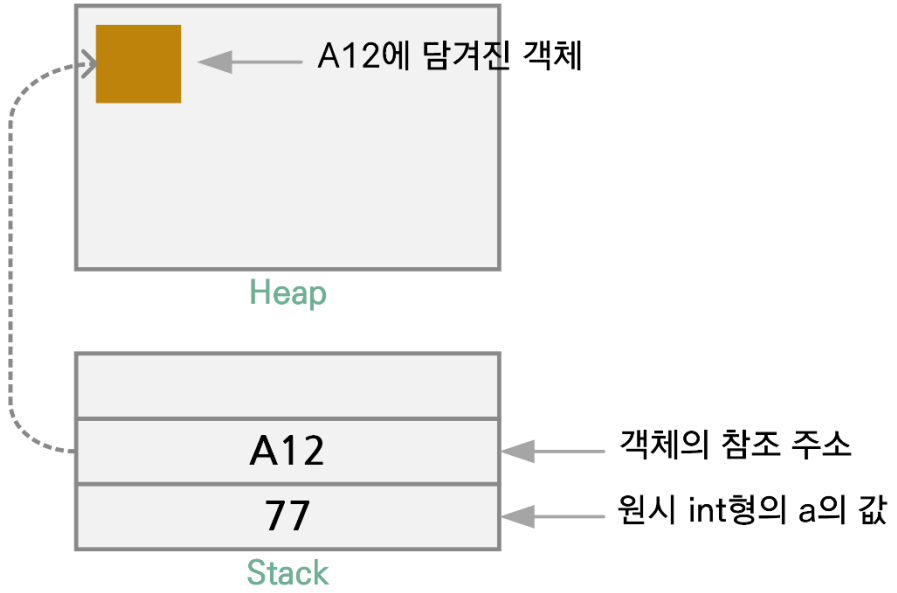
- 기본형 (Primitive data type)
  - 가공되지 않은 순수한 자료형으로 프로그래밍 언어에 내장
  - `int, long, float, double` 등

- 참조형 (Reference type)
  - 동적 공간에 데이터를 둔 다음 이것을 참조하는 자료형
  - `Int, Long, Float,Double` 등

자료형 알아보기

- 기본형과 참조형의 동작 원리 이해하기
  - 자바의 기본형과 참조형의 원리

```
int a = 77; // 기본형
Person person = new Person(); // 객체 참조형으로 person 객체를 위해 참조 주소(A12)를 가진다.
```



자료형 알아보기

- 기본형과 참조형의 동작 원리 이해하기 (Cont.)
  - 코틀린에서는 코딩 시 참조형만 사용되며 코틀린 컴파일러가 자동적으로 최적화함
  - 컴파일 과정을 거친 후 기본형이 사용됨

자료형 알아보기

정수형

음수가 사용되는 부호 있는 자료형

형식	자료형	크기	값의 범위
정수 자료형	Long	8바이트(64비트)	$-2^{63} \sim 2^{63}-1$
	Int	4바이트(32비트)	$-2^{31} \sim 2^{31}-1$
	Short	2바이트(16비트)	$-2^{15} \sim 2^{15}-1(-32{,}768 \sim 32{,}767)$
	Byte	1바이트(8비트)	$-2^{7} \sim 2^{7}-1(-128 \sim 127)$

부호 없는 정수형 (1.3버전 실험적)

형식	자료형	크기	값의 범위
부호 없는 정수형	ULong	8Bytes(64Bits)	$0 \sim 2^{64}-1$
	UInt	4Bytes(32Bits)	$0 \sim 2^{32}-1$
	UShort	2Bytes(16Bits)	$0 \sim 2^{16}(0 \sim 65535)$
	UByte	1Bytes(8Bits)	$0 \sim 2^{8}(0 \sim 255)$

자료형 사용의 예

자료형 생략

```
val num05 = 127 // Int형으로 추론
val num06 = -32768 // Int형으로 추론
val num07 = 2147483647 // Int형으로 추론
val num08 = 92233720368547758O7 // Long형으로 추론
```

접미사 접두사 사용

```
val exp01 = 123      // Int형으로 추론
val exp02 = 123L     // 접미사 L을 사용해 Long형으로 추론
val exp03 = 0x0F     // 접두사 0x를 사용해 16진 표기가 사용된 Int형으로 추론
val exp04 = 0b00001011 // 접두사 0b를 사용해 2진 표기가 사용된 Int형으로 추론
```

작은 값의 사용

```
val exp08: Byte = 127 // 명시적으로 자료형을 지정(Byte)
val exp09 = 32767 // 명시적으로 자료형을 지정하지 않으면 Short형 범위의 값도 Int형으로 추론
val exp10: Short = 32767 // 명시적으로 자료형을 지정(Short)
```

자료형 사용의 예

- 부호 없는 정수 자료형

```
val uint: UInt = 153u
val ushort: UShort = 65535u
val ulong: ULong = 46322342uL
val ubyte: UByte = 255u
```

- 큰 수를 읽기 쉽게 하는 방법
  - 읽기 쉽게 하기 위해 언더스코어(_)를 포함해 표현

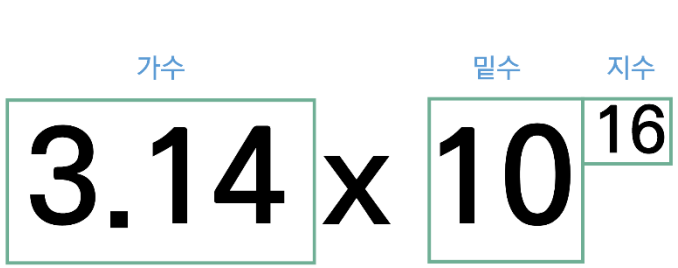
```
val number = 1_000_000
val cardNum = 1234_1234_1234_1234L
val hexVal = 0xAB_CD_EF_12
val bytes = 0b1101_0010
```

자료형 알아보기

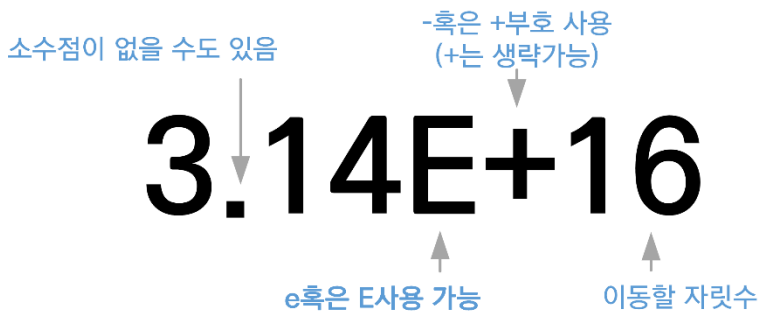
실수 자료형

형식	자료형	크기	값의 범위
실수형	Double	8바이트(64비트)	약 4.9E-324 ~ 1.7E+308(IEEE754)
	Float	4바이트(32비트)	약 1.4E-45 ~ 3.4E+38(IEEE754)

```
val exp01 = 3.14  // Double형으로 추론(기본)
val exp02 = 3.14F // 식별자 F에 의해 Float형으로 추론
```



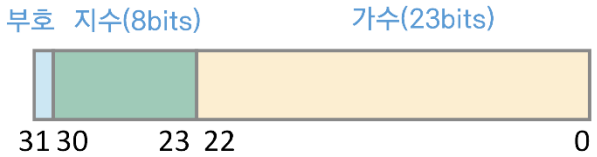
일반 수학의 표현



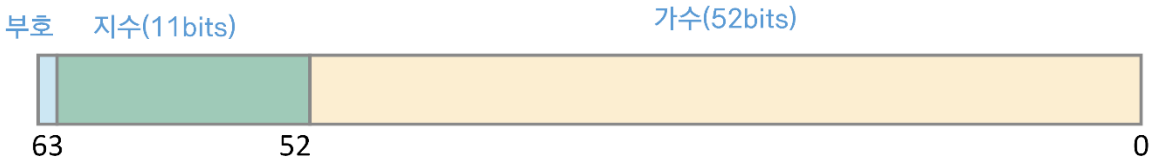
소스 코드상의 표현

부동 소수점 이해

- 32비트와 64비트의 부동 소수점 표현



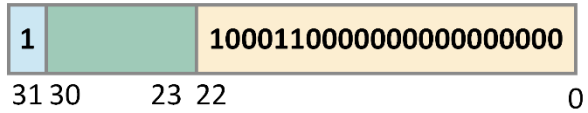
IEEE 754 표준의 float 형식

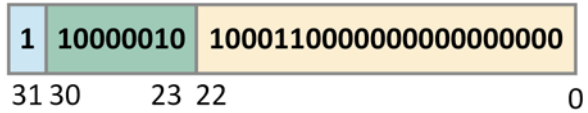


IEEE 754 표준의 double 형식

부동 소수점의 표현

- $-12.375_{(10)}$ 의 표현
  - 음수: -
    - 127(bias) 기준으로 구분
  - 절댓값: 12.375
    - $1100.011_{(2)}$
    - 1.100011×2^3
    - 1생략
    - 100011을 가수부에 표현
    - 2^3 -> 127+3 = $130_{(10)}$ = $10000010_{(2)}$





IEEE 방식의 부동 소수점 제한

- 공간 제약에 따른 부동 소수점 연산의 단점

```
package chap02.section2

fun main() {

    var num: Double = 0.1

    for(x in 0..999) {
        num += 0.1
    }
    println(num) //100.09999999999859
}
```

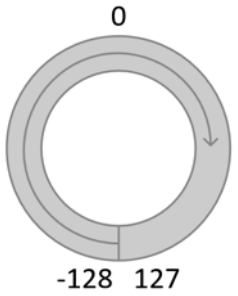
코딩해 보세요! 정수형과 실수 자료형의 최솟값과 최대값 출력하기

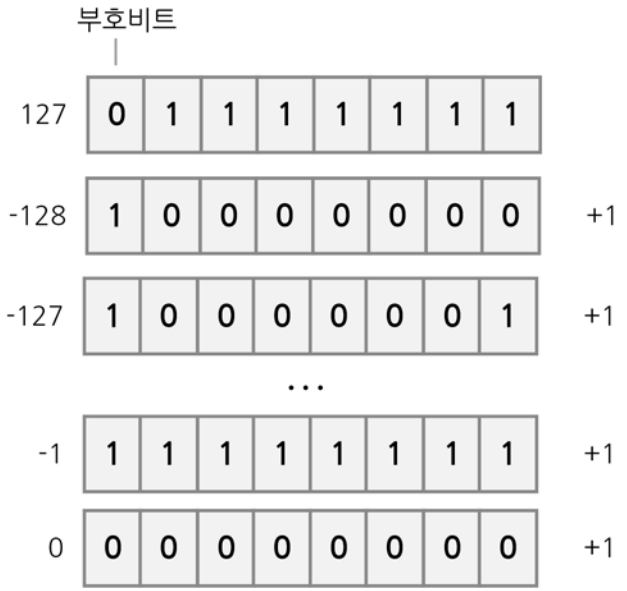
- MinMax.kt

```
package chap02.section2

fun main() {
    println("Byte min: " + Byte.MIN_VALUE + " max: " + Byte.MAX_VALUE)
    println("Short min: " + Short.MIN_VALUE + " max: " + Short.MAX_VALUE)
    println("Int min: " + Int.MIN_VALUE + " max: " + Int.MAX_VALUE)
    println("Long min: " + Long.MIN_VALUE + " max: " + Long.MAX_VALUE)
    println("Float min: " + Float.MIN_VALUE + " max: " + Float.MAX_VALUE)
    println("Double min: " + Double.MIN_VALUE + " max: " + Double.MAX_VALUE)
}
```

2의 보수란?

- 자료형의 최소 최대
  - 음수는 2의 보수 표현을 사용해 연산됨
    - 절댓값의 이진수에 값을 뒤집고 1을 더함
  - 예) $-6_{(10)}$
    - 6의 2진값: $0000\ 0110_{(2)}$
    - 값 뒤집기: $1111\ 1001_{(2)}$
    - 1을 더하기: $1111\ 1010_{(2)}$





  - 왜? 제한된 자료형을 음수/양수로 나누어 최대한 사용하며 2의 보수는 가산 회로만으로 뺄셈을 표현할 수 있기 때문

그 밖의 자료형 알아보기

논리 자료형

형식	자료형	크기	값의 범위
논리형	Boolean	1비트	true, false

```
val isOpen = true // isOpen은 Boolean으로 추론
val isUploaded: Boolean // 선언만 한 경우 자료형(Boolean) 반드시 명시
```

문자 자료형

형식	자료형	크기	값의 범위
문자	Char	2바이트(16비트)	0~2^{15}-1(\u0000 ~ \uffff)

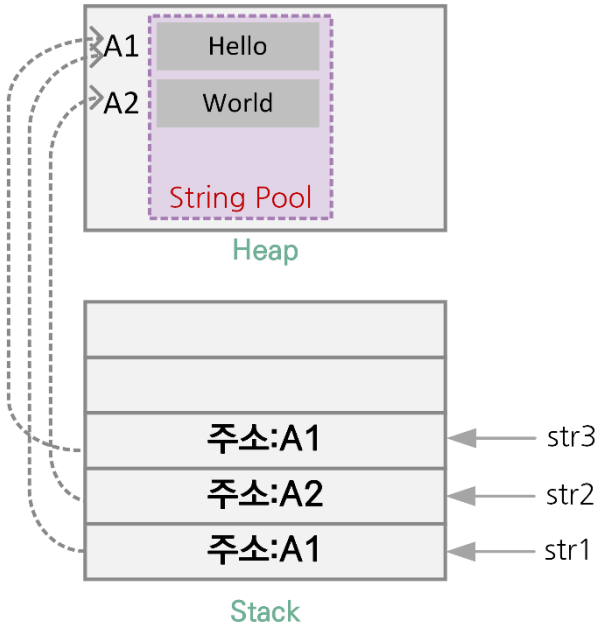
```
val ch = 'c' // ch는 Char로 추론
val ch2: Char // 선언만 한 경우 자료형(Char) 반드시 명시
```

문자열 자료형

문자열

- String으로 선언되며 String Pool이라는 공간에 구성

```
package chap02.section2

fun main() {
    var str1: String = "Hello"
    var str2 = "World"
    var str3 = "Hello"

    println("str1 === str2: ${str1 === str2}")
    println("str1 === str3: ${str1 === str3}")
}
```



표현식에서 문자열

- 표현식과 $ 기호 사용하여 문자열 출력하기

```
var a = 1
val s1 = "a is $a" // String 자료형의 s1을 선언하고 초기화. 변수 a가 사용됨
```

- 코딩해 보세요! StringExpression.kt

```
package chap02.section2

fun main() {
    var a = 1
    val str1 = "a = $a"
    val str2 = "a = ${a + 2}" // 문자열에 표현식 사용

    println("str1: \"$str1\", str2: \"$str2\"")
}
```

02 변수와 자료형, 연산자

02-1 코틀린 패키지

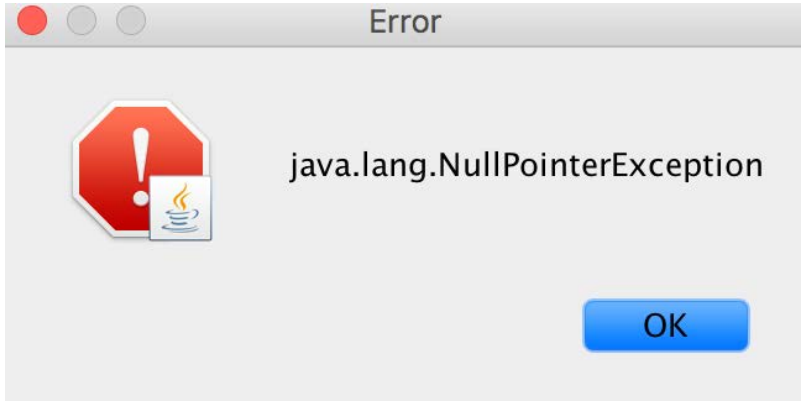
02-2 변수와 자료형

02-3 자료형 검사와 변환

02-4 코틀린 연산자

NPE

- “The billion dollar mistake” - Tony Hoare



null을 허용한 변수 검사

- NPE (NPE, NullPointerException)
  - 사용할 수 없는 null인 변수에 접근하면서 발생하는 예외

- 코틀린의 변수 선언은 기본적으로 null을 허용하지 않는다.
  - `val a: Int = 30`
  - `var b: String = "Hello"`

- null 가능한 선언
  - `val a: Int? = null`
  - `var b: String? = null`

코딩해 보세요!

- NullTest.kt

```
package chap02.section3

fun main() {
    var str1 : String = "Hello Kotlin"
    str1 = null  // null을 허용하지 않음(오류 발생)
    println("str1: $str1")
}
```

  - String을 String?으로 변경해 본다.

null 처리 방법

- Java와 Kotlin의 처리방법 비교
  - Java에서는 @NotNull과 @Nullable 어노테이션을 통해 알리지만
    Kotlin에서는 기본적으로 NotNull이고 Nullable 표현에만 '?'가 사용된다.

```java
public void set(@NotNull String a, @Nullable String b) {
  // Do noting
}
```

JAVA

```kotlin
fun set(a: String, b: String?) {
// Do noting
}
```

Kotlin

```java
String temp = null;
int size = -1;
if (temp != null) {
  size = temp.length();
}
// or

if (!TextUtils.isEmpty(temp)) {
  size = temp.length();
}
```

```kotlin
var temp: String? = null
var size = -1
if (temp != null) {
  size = temp.length
}

// or

var temp: String? = null
val size = if (temp != null) temp.length else -1
```

세이프 콜과 non-null 단정 기호 활용

```
package chap02.section3

fun main() {
    var str1 : String? = "Hello Kotlin"
    str1 = null
    println("str1: $str1 lenghth: ${str1.length}") // null을 허용하면 length가 실행될 수 없음
}
```

- 세이프 콜(Safe-call)
  - `str1?.length`
- non-null 단정 기호
  - `str1!!.length`

조건문을 활용해 null을 허용한 변수 검사하기

- if와 else의 활용

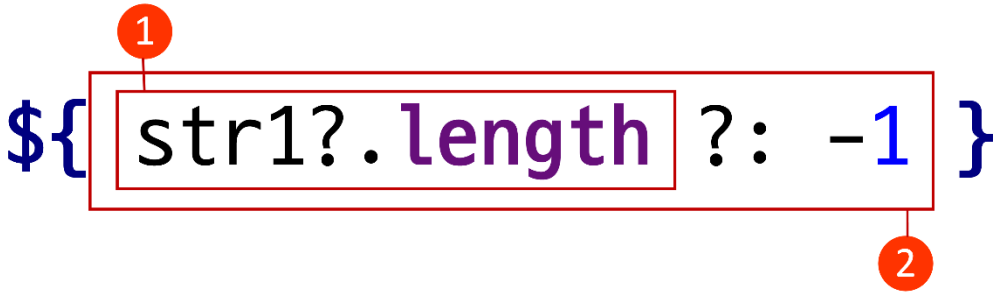
```
fun main() {
    var str1 : String? = "Hello Kotlin"
    str1 = null
    // 조건식을 통해 null 상태 검사
    val len = if(str1 != null) str1.length else -1
    println("str1: $str1 length: ${len}")
}
```

세이프 콜과 엘비스 연산자를 활용해 null을 허용

- 더 안전하게 사용하는 방법
  - `str1?.length ?: -1`

SafeCallandElvis.kt

```
package chap02.section3

fun main() {
    var str1 : String? = "Hello Kotlin"
    str1 = null
    println("str1: $str1 length: ${str1?.length ?: -1}") // 세이프 콜과 엘비스 연산자 활용
}
```



자료형 비교, 검사, 변환

- 코틀린의 자료형 변환
  - 기본형을 사용하지 않고 참조형만 사용
  - 서로 다른 자료형은 변환 과정을 거친 후 비교

```
val a: Int = 1 // 정수형 변수 a 를 선언하고 1을 할당
val b: Double = a // 자료형 불일치 오류 발생
val c: Int = 1.1 // 자료형 불일치 오류 발생
```

  - 변환 메서드의 이용

```
val b: Double = a.toDouble // 변환 메서드 사용
```

  - 표현식에서 자료형의 자동 변환

```
val result = 1L + 3 // Long + Int → result는 Long
```

자료형 비교, 검사, 변환

- 변환 메서드의 종류
  - `toByte: Byte`
  - `toLong: Long`
  - `toShort: Short`
  - `toFloat: Float`
  - `toInt: Int`
  - `toDouble: Double`
  - `toChar: Char`

기본형과 참조형 자료형의 비교 원리

- 이중 등호(==)와 삼중 등호(===)의 사용
  - == 값만 비교하는 경우
  - === 값과 참조 주소를 비교할 때

```
val a: Int = 128
val b: Int = 128
println(a == b) // true
println(a === b) // true
```

- 참조 주소가 달라지는 경우

```
val a: Int = 128
val b: Int? = 128
println(a == b) // true
println(a === b) // false
```

코딩해 보세요!

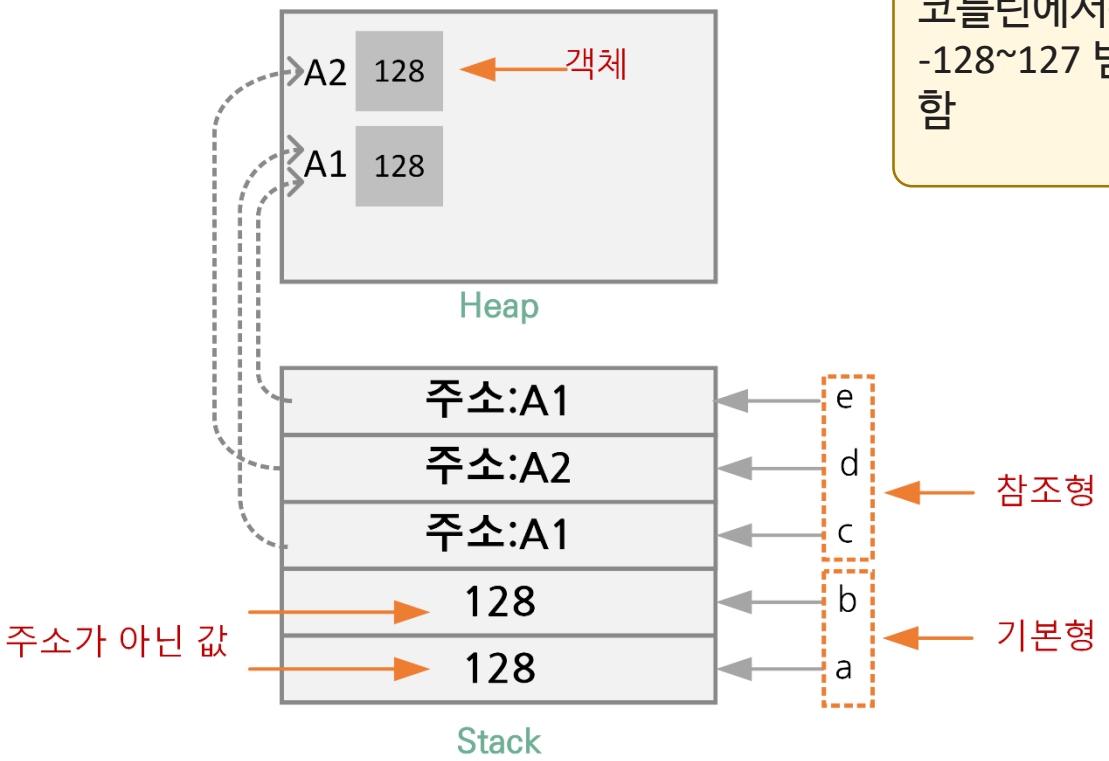
- 이중 등호 비교와 삼중 등호 비교 사용하기 - ValueRefCompare.kt

```
package chap02.section3

fun main() {
    val a: Int = 128
    val b = a

    println(a === b) // 자료형이 기본형인 int가 되어 값이 동일 true
    val c: Int? = a
    val d: Int? = a
    val e: Int? = c
    println(c == d) // 값의 내용만 비교하는 경우 동일하므로 true
    println(c === d) // 값의 내용은 같지만 참조를 비교해 다른 객체(주소 다름)이므로 false
    println(c === e) // 값의 내용도 같고 참조된 객체도 동일(주소 동일)하므로 true
}
```

코딩해 보세요!

- 이중 등호 비교와 삼중 등호 비교 사용하기

코틀린에서는 참조형으로 선언한 변수의 값이 -128~127 범위에 있으면 캐시에 그 값을 저장함



스마트 캐스트

- 구체적으로 명시되지 않은 자료형을 자동 변환
  - 값에 따라 자료형을 결정
  - Number형은 숫자를 저장하기 위한 특수한 자료형으로 스마트 캐스됨

코딩해 보세요!

- 스마트 캐스트 사용해 보기 - NumberTest.kt

```
package chap02.section3

fun main() {
    var test: Number = 12.2 // 12.2에 의해 test는 Float형으로 스마트 캐스트
    println("$test")

    test = 12  // Int형으로 스마트 캐스트
    println("$test")

    test = 120L // Long형으로 스마트 캐스트
    println("$test")

    test += 12.0f  // Float형으로 스마트 캐스트
    println("$test")
}
```

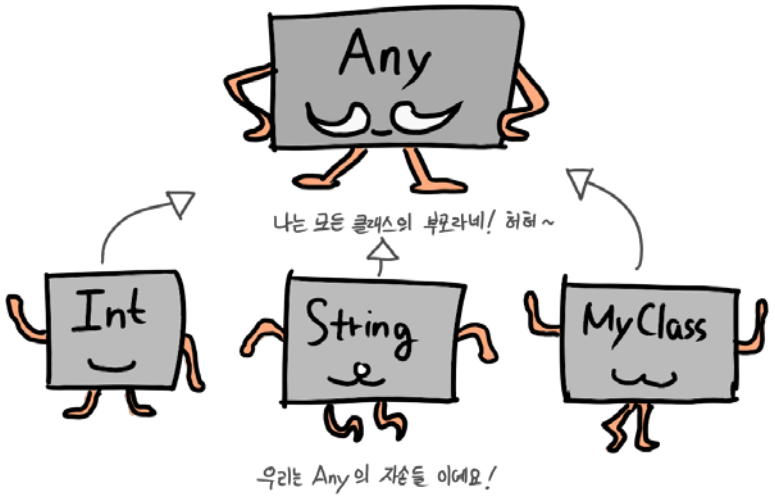
자료형의 검사

- is 키워드를 사용한 검사 - isCheck.kt

```
package chap02.section3

fun main() {

    val num = 256

    if (num is Int) { // num이 Int형일 때
        print(num)
    } else if (num !is Int) { // num이 Int형이 아닐 때, !(num is Int) 와 동일
        print("Not a Int")
    }
}
```

묵시적 변환

- Any
  - 자료형이 정해지지 않은 경우
  - 모든 클래스의 뿌리 - Int나 String은 Any형의 자식 클래스이다.



  - Any는 언제든 필요한 자료형으로 자동 변환 (스마트 캐스트)

코딩해 보세요!

- Any형 변수의 변환

```
package chap02.section3

fun main() {
    var a: Any = 1 // Any형 a는 1로 초기화될 때 Int형이 됨
    a = 20L // Int형이였던 a는 변경된 값 20L에의해 Long이 됨
    println("a: $a type: ${a.javaClass}")  // a의 자바 기본형을 출력하면 long이 나옴
}
```

코딩해 보세요!

- Any형으로 인자를 받는 함수 만들기

```
package chap02.section3

fun main() {
    checkArg("Hello") // 문자열을 인자로 넣음
    checkArg(5) // 숫자를 인자로 넣음
}

fun checkArg(x: Any) {  // 인자를 Any형으로 받음
    if (x is String) {
        println("x is String: $x")
    }
    if (x is Int) {
        println("x is Int: $x")
    }
}
```

02 변수와 자료형, 연산자

02-1 코틀린 패키지

02-2 변수와 자료형

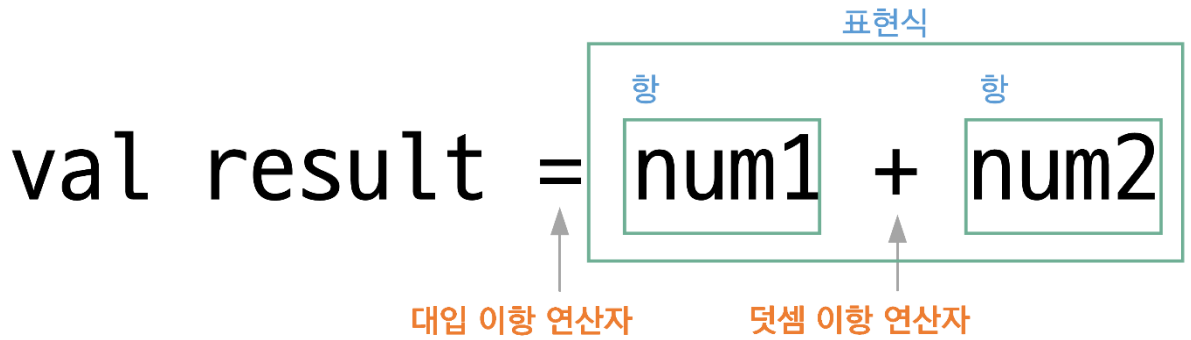
02-3 자료형 검사와 변환

02-4 코틀린 연산자

기본 연산자

- 종류
  - 산술, 대입, 증가, 감소, 비교, 논리 연산자 등

- 수식의 구조



기본 연산자

산술 연산자

연산자	의미	사용 예
+	덧셈	3 + 2
-	뺄셈	3 - 2
*	곱셈	3 * 2
/	나눗셈	3 / 2
%	나머지(Modulus)	3 % 2

```
...
    if ((n % 2) == 1) { // 홀수
        println("n is an Odd number")
    }
    if ((n % 2) == 0) { // 짝수
        println("n is an Even number")
...
```

기본 연산자

대입 연산자

연산자	의미	사용예
=	오른쪽 항의 내용을 왼쪽 항에 대입	num = 2
+=	두 항을 더한 후 왼쪽 항에 대입	num += 2
-=	왼쪽 항을 오른쪽 항으로 뺀 후 왼쪽 항에 대입	num -= 2
*=	두 항을 곱한 후 왼쪽 항에 대입	num *= 2
/=	왼쪽 항을 오른쪽 항으로 나눈 후 왼쪽 항에 대입	num /= 2
%=	왼쪽 항을 오른쪽 항으로 나머지 연산 후 왼쪽 항에 대입	num %= 2

```
num = num + 2 // 산술 연산자와 대입 연산자를 함께 사용하는 경우
num += 2 // 이렇게 간략하게 표현
```

기본 연산자

증가 연산자와 감소 연산자

연산자	의미	사용 예
++	항의 값에 1 증가	++num 또는 num++
--	항의 값에 1 감소	--num 또는 num--

IncDecOperator.kt

```
package chap02.section4

fun main() {
    var num1 = 10
    var num2 = 10
    val result1 = ++num1 // num 값 증가 후 대입
    val result2 = num2++ // 먼저 num 값 대입 후 증가

    println("result1: $result1")
    println("result2: $result2")
    println("num1: $num1")
    println("num2: $num2")
}
```

기본 연산자

비교 연산자

연산자	의미	사용 예
〉	왼쪽이 크면 true, 작으면 false 반환	num 〉 2
〈	왼쪽이 작으면 true, 크면 false 반환	num 〈 2
〉=	왼쪽이 크거나 같으면 true, 아니면 false	num 〉= 2
〈=	왼쪽이 작거나 같으면 true, 아니면 false	num 〈= 2
==	두 개 항의 값이 같으면 true, 아니면 false	num1 == num2
!=	두 개 항의 값이 다르면 true, 아니면 false	num1 != num2
===	두 개 항의 참조가 같으면 true, 아니면 false	num1 === num2
!==	두 개 항의 참조가 다르면 true, 아니면 false	num1 !== num2

기본 연산자

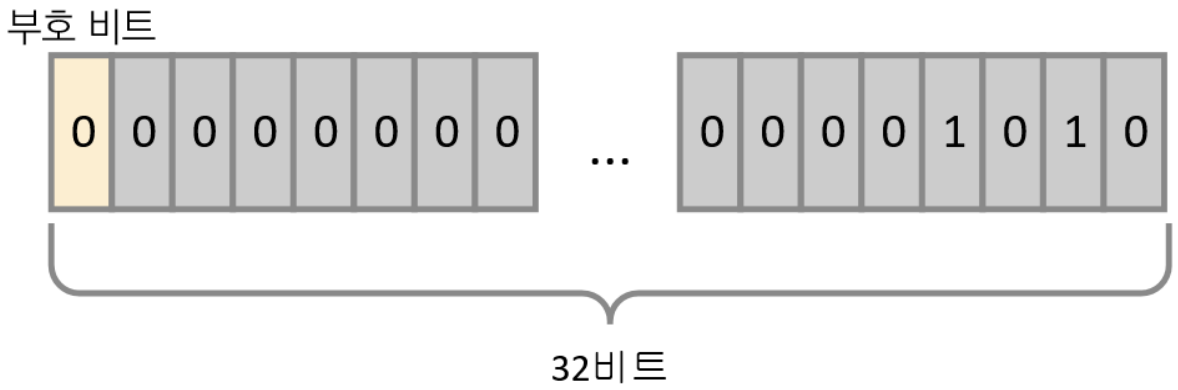
논리 연산자

연산자	의미	사용예
&&	논리곱으로 두 항이 모두 true일 때 true, 아니면 false	exp1 && exp2
||	논리합으로 두 항 중 하나의 항이 true일 때 true, 아니면 false	exp1 || exp2
!	부정 단항 연산자로 true를 false로, false를 true로 바꿈	!exp

```
var check = (5>3) && (5>2)  // 2개의 항((5>3), (5>2))이 모두 참이면 true
check = (5>3) || (2>5)      // 2개 중 1개의 항이 참이면 true
check = !(5>3)              // true는 false로, false는 true로 변경
```

비트 연산자

- 비트와 비트 연산 이해하기
  - $1010_{(2)} = 2^1+2^3 = 10_{(10)}$
  - 가장 왼쪽에 있는 비트는 양(+), 음(-)을 판단하는 데 사용



비트 연산자

- 비트 연산을 위한 비트 메서드

표현식	설명
4.shl(bits)	4를 표현하는 비트를 bits만큼 왼쪽으로 이동(부호 있음)
7.shr(bits)	7을 표현하는 비트를 bits만큼 오른쪽으로 이동(부호 있음)
12.ushr(bits)	12를 표현하는 비트를 bits만큼 오른쪽 이동(부호 없음)
9.and(bits)	9를 표현하는 비트와 bits를 표현하는 비트로 논리곱 연산
4.or(bits)	4를 표현하는 비트와 bits를 표현하는 비트로 논리합 연산
24.xor(bits)	23를 표현하는 비트와 bits를 표현하는 비트의 배타적 연산
78.inv( )	78을 표현하는 비트를 모두 뒤집음

코딩해 보세요! 비트 이동 연산자 사용하기

- BitsShift.kt

```
package chap02.section4

fun main() {
    var x = 4
    var y = 0b0000_1010 // 10진수 10
    var z = 0x0F        // 10진수 15

    println("x shl 2 -> ${x shl 2}")    // 16
    println("x.inv -> ${x.inv()}")    // -5

    println("y shr 2 -> ${y/4}, ${y shr 2}")   // 2, 2
    println("x shl 4 -> ${x*16}, ${x shl 4}")  // 64, 64
    println("z shl 4 -> ${z*16}, ${z shl 4}")  // 240, 240

    x = 64
    println("x shr 4 -> ${x/4}, ${x shr 2}")  // 16, 16

}
```

코딩해 보세요! ushr 이해하기

- UshrEx.kt

```
package chap02.section4

fun main() {
    val number1 = 5
    val number2 = -5

    println(number1 shr 1)
    println(number1 ushr 1) // 변화 없음
    println(number2 shr 1)  // 부호 비트가 1로 유지
    println(number2 ushr 1) // 부호 비트가 0이 되면서 변경
}
```

코딩해 보세요! 비트 논리 연산자 테스트하기

- LogicalBitwise.kt

```
package chap02.section4

fun main() {

    val number1 = 12
    val number2 = 25
    val result: Int

    result = number1 or number2   // result = number1.or(number2)와 동일
    println(result)
}
```

코딩해 보세요! xor 연산자로 두 값을 스왑하기

- XorSwapTest.kt

```
package chap02.section4

fun main() {
    var number1 = 12
    var number2 = 25

    number1 = number1 xor number2
    number2 = number1 xor number2
    number1 = number1 xor number2

    println("number1 = " + number1)
    println("number2 = " + number2)
}
```