02_1 Java Basics

Object-Oriented Programming

이 강의에서는 Java 언어에서 객체지향 부분 이전에 기본이 되는 프로그래밍 부분에 대해 강의하겠습니다.

Basic structure of a Java program (1/5)

```java
/**
 * This is a JavaDoc comment to generate documentation
 */
public class HelloWorld1 {
    public static void main(String[] args) {
        // Single statement
        System.out.println("Hello, World!");

        // Block of statements
        {
            System.out.println("This is a block.");
            System.out.println("It contains multiple statements.");
        }
    }
}
```

- public: access modifier
  - The class can be used anywhere
- class: keyword for class definition
- HelloWorld1: the name of the class
- begins with { and ends with }

페이지 2

먼저 Java program의 일반적인 구조에 대해 알아보겠습니다.
Java program은 모든 내용이 class의 일부가 되어야 하기 때문에
어쨋든 class를 정의해야 합니다.
이 예제 에서는 HelloWorld1이라는 class가 정의되어 있습니다.

public은 access modifier라고 불리는 keyword인데
이 class가 어떤 다른 class에서도 사용될 수 있다는 뜻입니다.

class는 class type을 정의하는 keyword입니다.

그 뒤에 class 이름이 나오는데 이 예에서는 HelloWorld1 입니다.
이전에도 언급했지만 class의 이름은
java source file의 이름과 일치해야 합니다.

Class body의 시작은 왼쪽 brace { 로 열고
class의 끝은 오른쪽 brace } 로 닫게 됩니다.

Basic structure of a Java program (2/5)

```java
/**
 * This is a JavaDoc comment to generate documentation
 */
public class HelloWorld1 {
    public static void main(String[] args) {
        // Single statement
        System.out.println("Hello, World!");

        // Block of statements
        {
            System.out.println("This is a block.");
            System.out.println("It contains multiple statements.");
        }
    }
}
```

- Method "main"
- The first method executed when the program starts
- Should be included in any executable program

페이지 3

Class body안에는 class에 속한 data를 나타내는 각종 field와
operation을 나타내는 method들이 존재합니다.

이 예제의 class는 data, 즉, field를 가지지 않고 있습니다.
다만 하나의 method인 main을 가지고 있습니다.

Main method는 프로그램이 시작될 때 처음 실행되는 method입니다.

따라서 실행이 가능한 프로그램 (class) 일 경우
Main method를 반드시 가져야 합니다.

Basic structure of a Java program (3/5)

```java
/**
 * This is a JavaDoc comment to generate documentation
 */
public class HelloWorld1 {
    public static void main(String[] args) {
        // Single statement
        System.out.println("Hello, World!");

        // Block of statements
        {
            System.out.println("This is a block.");
            System.out.println("It contains multiple statements.");
        }
    }
}
```

- public: access modifier
  - main method can be called from anywhere
- static: method belongs class not instance
- void: no return type
- main: method name
- Strings[] args: array of Strings, command line arguments

페이지 4

main method의 header에 가장 먼저 나오는 것은
역시 access modifier인 public 입니다.
이 public은 OS에서 프로그램을 실행하기 위해
main을 call 할 수 있게 한다라고 이해하면 되겠습니다.

다음에 나오는 static은 main method가 이 class 내에서
Static method 라는 것을 나타내는데
간단히 말하면 이 class를 틀로 만들어지는 모든 object가
단 하나의 main method를 공유한다는 뜻입니다.
아직 static 의 의미를 모두 이해하기 힘들 수 있으나
나중에 자세히 학습하도록 하겠습니다.

void는 main method의 return value가 없다는 뜻입니다.
Method, 즉, function은 흔히 어떤 값을
Return 할 수도 있고 그렇지 않을 수도 있습니다.
f(a, b) = a + b 라는 function f가 있다고 하면
Return value는 a + b 값이 될 수 있을 것입니다.
그러나 어떤 function (method) 는 return value 없이
일련의 작업만을 연속적으로 실행할 수도 있습니다.

main은 method의 이름을 말합니다.

그 뒤에 괄호안에 들어있는 String[] args 는
parameter의 type과 이름입니다.
이전의 예에서 f(a, b) 라는 function, 즉, method에서는
a와 b라는 두 개의 parameter를 받아
어떤 계산을 하게 되어있습니다.
Main method의 경우 이 parameter를
Command line arguments라고 부르는데
프로그램을 실행하는 명령에 붙이는 parameter를 말합니다.
예를 들면 "javac -version" 이라는 프로그램을 실행했다고 하면
javac는 프로그램이 이름이고
-version은 command line arguments로 프로그램에 전달됩니다.
그러나 우리는 아직 이 command line arguments를 이용하지는 않고 있습니다.
String[] 에서 bracket 기호는
args가 String의 array type이라는 것을 말하고 있습니다.
예를 들어 "foo 3 Seoul 7" 이라는 프로그램 실행을 고려하면

args[0] = “3”, args[1] = "Seoul", args[2] = “7” 이 되겠습니다.

Basic structure of a Java program (4/5)

```java
/**
 * This is a JavaDoc comment to generate documentation
 */
public class HelloWorld1 {
    public static void main(String[] args) {
        // Single statement
        System.out.println("Hello, World!");

        // Block of statements
        {
            System.out.println("This is a block.");
            System.out.println("It contains multiple statements.");
        }
    }
}
```

multi-line comments: /* .... */
/** ... */: can be captured by JavaDoc tool (automatic manual generation)
single-line comments: // .....

페이지 5

Java의 comment는 C나 C++와 같은 형식입니다.
먼저, 슬래쉬 스타 /* 로 시작하여 스타 슬래쉬 */ 로 끝내는 멀티라인 코멘트가 있습니다.
특별히 슬래쉬 더블스타로 시작할 수도 있는데
이 경우는 Java의 utility 중 하나인 JavaDoc을 이용하여
API documentation 을 자동으로 만들어 주는 기능을 이용할 수 있습니다.
그러나 여기에 대해서는 더 이상 이 코스에서 언급하지 않겠으며
필요한 경우 다른 참고자료들을 찾아 보시기 바랍니다.

싱글라인 comment는 더블 슬래쉬 // 로 시작하면 되고
같은 줄의 끝까지 모두 comment로 간주 됩니다.

Basic structure of a Java program (5/5)

```java
/**
 * This is a JavaDoc comment to generate documentation
 */
public class HelloWorld1 {
    public static void main(String[] args) {
        // Single statement
        System.out.println("Hello, World!");

        // Block of statements
        {
            System.out.println("This is a block.");
            System.out.println("It contains multiple statements.");
        }
    }
}
```


페이지 6

다음은 statement block에 관한 것인데
Left brace { 로 시작하여 right brace } 로 닫습니다.
이 block은 local variable들을 위한 새로운 scope를 만들면서
If, for, while 문 등에서 body를 이루는 multiple statement들을
묶어 주는 역할을 합니다.
물론 method 전체의 시작과 끝도 하나의 block으로
Class 전체의 시작과 끝도 하나의 block으로 간주할 수 있겠습니다.
variable의 scope에 대해서는 나중에 더 자세히 강의할 예정입니다.

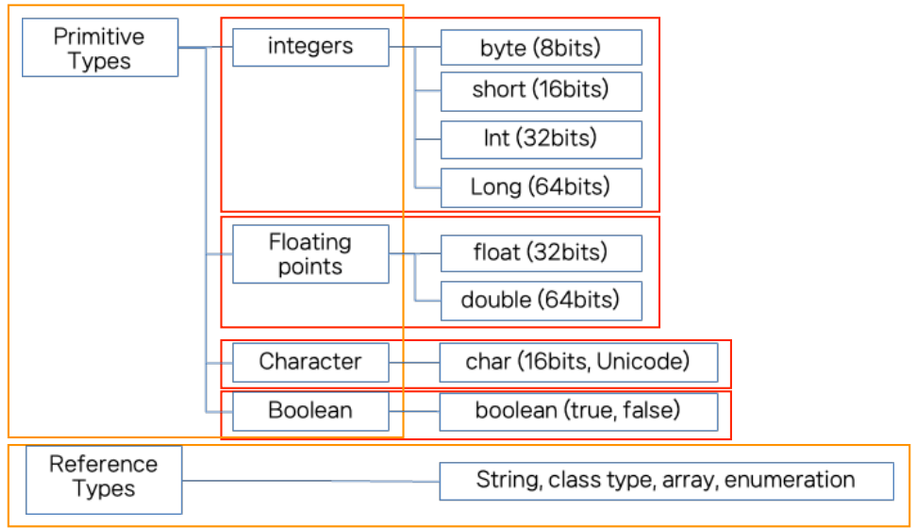
Identifiers

- Identifiers
  - The name of variables, constants, methods, parameters, classes, ...
- Identifier Rules
  - **Case sensitivity**: MyVariable ≠ myVariable
  - **Reserved words (keywords) cannot be used**: class, public, void, …
  - **General Naming Rules:**
    - Cannot start with numeric characters
    - Cannot use special characters except $ and _
  - **Naming Conventions**:
    - camelCase for variables and methods: myVariable
    - PascalCase for classes: MyClass

페이지 7

다음은 identifier, 즉, 식별자 입니다.
Identifier는 간단히 말해 프로그래머가 새로 지어내야 할 이름인데,
Variable, constant, method, parameter, class 등의 이름을 말합니다.
Java의 identifier에는 case sensitive rule이 적용됩니다.
즉, 알파벳 대문자와 소문자가 사용된 경우 다른 identifier로 인식합니다.
또, Java 문법에서 사용되고 있는 keyword는 identifier로 사용될 수 없습니다.
예를 들면 class, public, void 등 입니다.
Identifier 이름을 짓는데 반드시 따라야 하는 rule은
Numeric character로 시작할 수 없으며
달러 싸인 ($) 과 언더바 (_) 이외에 다른 special character는
사용될 수 없다는 두가지 뿐입니다.
그러나 일반적으로 Java 언어에서
identifier 이름을 짓는 convention이 있는데,
variable과 methods는 camelCase, 즉,
소문자로 시작하고, 단어가 바뀔 때 시작 문자를 대문자로 하는 것과
Class 이름은 PascalCase, 즉,
대문자로 시작하고, 단어가 바뀔 때 시작 문자를 대문자로 하는 것이
권장되고 있습니다.

Data Types



페이지 8

Java의 data type에는 크게 두 가지가 있는데,
첫번째는 Primitive type이고
두번째는 Reference type 입니다.
Primitive type은 정수, 실수 등으로 기본적으로 값 1개 만을 가지는 타입입니다.
따라서 Primitive type은 class로 취급되지 않습니다.
반면에 reference type은 String과 같은 모든 class들과
Array, enumeration 등의 type을 말합니다.
엄밀히 말하면 reference는 memory의 address와 같은 개념으로
class와 같은 복잡한 data가 모여있는 memory의 주소를
그 값으로 가지고 있다고 보면 되겠습니다.
Reference type에 대해서는 추후 더 자세히 학습할 것입니다.
Primitive type에는 정수형, 실수형, 캐릭터형, 불리안형의 네가지가 있는데,
정수형에는 8bits (즉, 1 byte) 의 byte type, 16bits (2 bytes)의 short type,
32bits (4 bytes)의 int type, 64bits (8 bytes)의 long type 의
4가지 type이 있습니다.
실수형에는 32bits의 float, 64bits의 double이 있습니다.
캐릭터형에는 16bits의 char 가 있습니다.
불리안형에는 true나 false 중 한가지 값을 가질 수 있는
boolean type이 있습니다.

Data Types: Example

```java
public class DataTypesExample {
    public static void main(String[] args) {
        int myNumber; // variable declaration
        myNumber = 10; // initialization
        int yourNumber = 10; // declaration with initialization
        float f1 = 3.151492f; // float type literal should be ended with the suffix 'f'
        double d1 = 3.151492; // double type literal doesn't have to have any suffix.

        System.out.println("myNumber=" + myNumber + " yourNumber=" + yourNumber +
                " f1=" + f1 + " d1=" + d1);
    }
}
```

```
OUTPUT:
myNumber=10 yourNumber=10 f1=3.151492 d1=3.151492
```


페이지 9

Data type에 대한 example 프로그램을 보겠습니다.
먼저 int type인 myNumber를 declare하였고
그 값을 10으로 초기화 하였습니다.
yourNumber의 경우처럼 declaration과 초기화를 한번에 하기도 합니다.
literal은 여기서 직접적으로 주어지는 숫자, 문자, 문자열 등을 말하는데요,
float type f1에 literal을 assign할 경우에 숫자 맨 뒤에 f를 붙여 double과 구분해야 합니다.
대신에 double type에 literal을 assign할 경우에는 그냥 숫자만 쓰면 됩니다.

Type Conversion

```java
public class TypeConversion {
    public static void main(String[] args) {

        int myInt = 10;
        double myDouble = myInt;    // Implicit conversion from int to double
                                    // range of (double) > range of (int)
        System.out.println("myInt(" + myInt + ") myDouble(" + myDouble + ")");

        myDouble = 9.78;
        myInt = (int) myDouble;     // Explicit conversion from double to int

        System.out.println("myInt(" + myInt + ") myDouble(" + myDouble + ")");
    }
}
```

```
OUTPUT:
myInt(10) myDouble(10.0)
myInt(9) myDouble(9.78)
```


페이지 10

type이 다른 variable의 value를
다른 type의 variable에 assign하는 경우를 살펴 보겠습니다.

myInt는 int type입니다.
이 myInt의 값을 double type인 myDouble에 assign 하였습니다.
이 때 아무런 다른 장치 없이 그냥 assign을 해도 되는 것은
double type의 data 범위가 int보다 훨씬 넓기 때문입니다.
이런 방식을 implicit conversion이라 합니다.
myInt의 value 10은 double로 assign 되면서 10.0 이 되었습니다.

반면에 myDouble을 myInt로 assign하는 경우에는
(Int) 라는 casting operator를 붙여 explicit conversion을 해 주어야 합니다.
double에서 int로 casting 되었을 경우
소숫점 이하의 값은 버려지고 정수부분만 남게 됩니다.
따라서 9.78이라는 double 값은 9라는 int값으로 변환됩니다.

Naming Constants

- Named constants having names:

```
public static final int INCHES_PER_FOOT = 12;
public static final double RATE = 0.14;
```

  - Cannot change the value in the program
  - Naming convention for constants: Use all uppercase letters, and designate word boundaries with an underscore character

페이지 11

Named constants는 특정한 값을 identifier로 나타내어
그 의미를 명확히 하면서
프로그램에서 숫자 대신 이름을 사용할 수 있도록 하는 것입니다.
이 예에서는 INCHES_PER_FOOT 라는 값이 12로 정의되었는데
1 foot가 12 inch라는 사실은 변하지 않기 때문에
이 값을 constant, 즉, 상수로 놓을 수 있는 것입니다.
RATE는 0.14 라는 값으로 고정되는데
금리와 같은 어떤 특정 비율을 나타내는 것으로 예측할 수 있습니다.
이와 같이 named constant는 한번 그 값이 정의되면
프로그램 내에서 다시 값을 바꿀 수 없습니다.
만일 바꾸려는 시도를 하면
Compile error가 나게 됩니다.
public static final 이라는 keyword들이 붙어 있는데
여기서 final 의 의미가 constant를 나타내며
값을 다시 바꿀 수 없다는 것을 나타냅니다.
Naming constant를 나타내는 identifier의 이름을 지을 때
일반적으로 대문자만을 사용하며,
단어간에는 언더바 (_) 를 사용합니다.

Strings

- A class used to handle text

```java
public class StringClass {
    public static void main(String[] args) {
        String greeting = "Hello, World!";
        String firstName = "John";
        String lastName = "Doe";

        // Concatenation
        String fullName = greeting + " " + firstName + " " + lastName;

        System.out.println(fullName); // Outputs "Hello, World! John Doe"
    }
}
```


페이지 12

String type은 Java가 제공하는 기본 package에 들어있는 class type 입니다.
text string을 다루기 위해 사용되는 reference type입니다.
세개의 String greeting, firstName, lastName들은
각각 "Hello World!", "John", "Doe" 로 initialize되었습니다.
String은 plus operator를 사용하여 서로 연결,
즉, concatenation 될 수 있습니다.
example에서 연결된 fullName String의 값이
화면에 프린트 되는 것을 볼 수 있습니다.

Concatenation of Strings

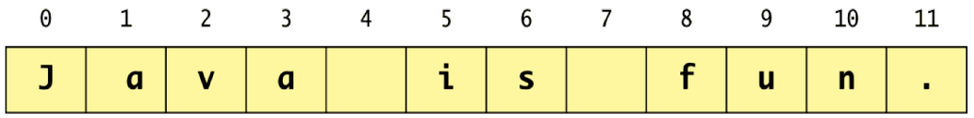
```
String str4 = “The answer is " + 42     // “The answer is 42”
int k = 35;
String str5 = “Yes ” + k;               // “Yes 35”
```


페이지 13

String concatenation의 경우 String들 간의 연결 뿐 아니라
이 프로그램 처럼 42와 같은 정수 literal을 연결하여
하나의 String으로 만들 수 있습니다.
또 int type k와 같이 primitive type이나 class type의 variable도
concatenate할 수 있습니다.
이것이 가능한 것에 대해서는 class를 학습하면서
좀 더 자세히 알아보도록 하겠습니다.

String Indexes

"Java is fun."



페이지 14

String은 다른 면에서 본다면 character들이 모여 있는 것으로 볼 수 있습니다.
그러나 String이 char type의 array (배열) 과 같다는 것은 아닙니다.
C 나 C++에서는 이 두가지가 일치하지만
Java의 String은 그렇지 않습니다.
그럼에도 Java의 String을 이루고 있는 각 문자들에는
index가 붙어 있습니다.
0부터 시작하는 index는 뒤에서 볼 String의 여러 method들에서
유용하게 이용됩니다.
유의할 점은 빈칸 (blank space) 도
하나의 character로 간주된다는 것입니다.

String Methods (1/3)

```java
public class StringMethods {
    public static void main(String[] args) {

        String str = "Hello, World!";

        int length = str.length(); // length of the string: 13
        char ch = str.charAt(0); // character at a specific index position: 'H'

        // substring from the given begin index to the end: "World!"
        String substr1 = str.substring(7);

         // substring from index1 to index2: "Hello"
        String substr2 = str.substring(0, 5);

        // str과 given String의 내용 비교 (reference, 즉, 주소 비교 아님) : true
        boolean isEqual = str.equals("Hello, World!”);

        // 대소문자 구분없이 내용 비교: true
        boolean isEqualIgnoreCase = str.equalsIgnoreCase("hello, world!");
```


페이지 15

String class에는 여러가지의 method가 존재합니다.
먼저 str String의 값이 “Hello, World!” 라 가정합시다.

length() 는 String의 길이를 return합니다.
앞 페이지의 index 측면에서 본다면
String의 끝 문자의 index + 1 이 length와 같습니다.

substring(int) 는 String에서 주어진 parameter index 부터
맨 끝까지의 부분 String을 return 합니다.
“Hello, World!” 의 경우 substring(7) 은
Index 7인 ‘W’ 부터 시작하여 끝까지 이니까
“World!” 를 return 합니다.

substring(0, 5)는 begin index 0부터 end index 5까지의
부분 String을 return 합니다.

equals(String other) 는 equals를 call한 String과
Parameter other 의 내용이 같은지를 test하여
true 또는 false를 return 합니다.

equalsIgnoreCase는 equals와 같으나
대소문자 구분없이 같다면 true를 return합니다.

String Methods (2/3)

```java
// dictionary order로 str > "Hello" 이면 positive, str < "Hello" 이면 negative
// str == "Hello" 이면 0을 return
int comparison = str.compareTo("Hello"); // Positive value
int comparisonIgnoreCase = str.compareToIgnoreCase("hello"); // Positive value

int index = str.indexOf("World");       // 처음 출현하는 World의 W의 index: 7
int lastIndex = str.lastIndexOf("o");   // 마지막 출현하는 o의 index: 8
boolean contains = str.contains("Hello"); // 주어진 substring을 포함하는가? true

String replacedStr = str.replace('o', 'a'); // "Hella, Warld!"
String replacedStr2 = str.replace("World", "Java"); // "Hello, Java!"
String replacedAllStr = str.replaceAll("l", "L"); // "HeLLo, WorLd!"
String replacedFirstStr = str.replaceFirst("l", "L"); // "HeLlo, World!"

String upper = str.toUpperCase(); // 대문자로: "HELLO, WORLD!"
String lower = str.toLowerCase(); // 소문자로: "hello, world!"
```


페이지 16

str.compareTo(String other) 는 사전식 배열에 따라
str이 other보다 더 작으면 마이너스 값을
같으면 0을, 더 크면 플러스 값을 return 합니다.

str.indexOf("World") 는 str String 내에 "World" 라는 substring이
있을 경우 가장 첫번째 substring의 시작 index를 return 합니다.

str.lastIndexOf("o") 는 str 내에 String "o" 라는 substring이 출현할 경우
가장 마지막에 출현하는 "o" 의 시작 index를 return 합니다.

str.contains("Hello") 는 str 내에 "Hello" substring 이 존재하면
true를 return합니다.

str.replace() 는 여러 가지의 version이 있는데
str에 출현하는 character, substring 등을
두번째 것으로 대치하는 역할을 합니다.

str.toUpperCase() 는 str을 모두 대문자로
str.toLowerCase() 는 str을 모두 소문자로 바꾸어 return 합니다.

String Methods (3/3)

```
        String trimmedStr = str.trim(); // 앞뒤 공백이 제거된 문자열
        String[] words = str.split(", "); // words[0] = “Hello”, words[1] = “World!”
        String joinedStr = String.join(", ", "Hello", "World"); // "Hello, World"

        String intStr = String.valueOf(123);  // integer 123을 String "123" 으로
        String boolStr = String.valueOf(true); // boolean true를 String "true"로

        boolean startsWith = str.startsWith("Hello"); // true
        boolean endsWith = str.endsWith("!"); // true
        boolean isEmpty = str.isEmpty(); // false
    }
}
```


페이지 17

str.trim() 은 String의 맨 앞과 맨 뒤에 존재하는 공백을 모두 없애는 역할을 합니다.
이 method는 web과 같은 UI 에서 text input을 받을 때 편리하게 사용됩니다.
String.join(a, b, c) 는 b, a, c 순서로 세 String을 concatenation합니다.
String.valueOf(123) 은 정수 123을 String “123” 으로 바꾸어 return 합니다.
마찬가지로 String.valueOf(true) 는 boolean value인 true를
String "true" 로 만들어 return 합니다.
str.startsWith(“Hello”) 는 str이 “Hello”로 시작하면 true를 return합니다.
str.endsWith("!") 는 str이 “!”로 끝날 경우 true를 return합니다.
str.isEmpty()는 str이 empty String일 경우 true를 return 합니다.

Escape Sequences

1) 프로그램에서 특별한 의도로 사용되는 문자
2) 눈에 안보이는 특수 문자

```
\" Double quote.
\' Single quote.
\\ Backslash.
\n New line. Go to the beginning of the next line.
\r Carriage return. Go to the beginning of the current line.
\t Tab. White space up to the next tab stop.
```

```
Ex) System.out.print(“Hey Guys\\\n\”Oh\t\’Yes!!”);

Hey Guys\
“Oh    ‘Yes!!
```


페이지 18

Escape Sequence는
첫번째, 프로그램에서 특별한 의도로 사용되는 문자를 프린트하고 싶을때
두번째, 눈에 안보이는 특수 문자를 지칭하여 프린트하고 싶을 때 사용합니다.
Double quote, Single quote, Backslash 는 특별한 의도로 사용된 문자들이고
New line, carriage return, tab 은 눈에 안보이는 특수 문자 입니다.
이 example에서는 backslash, new line, double quote,
Tab, single quote의 escape sequence를 사용하여 프린트 하였습니다.

String Processing

- A **`String`** object in Java
  - immutable, i.e., the characters it contains cannot be changed
- **`StringBuffer`**
  - Can be changed
- [NOTE] Possible to change the value of a **`String`** variable by using an assignment statement

```
String name = "Soprano";
name = "Anthony " + name;
```


페이지 19

Java String은 immutable 입니다, 즉, String의 내용은 바꿀 수 없습니다.
내용을 바꾸고 싶다면 String 대신 StringBuffer object를 사용합니다.
물론 String의 내용을 assignment를 이용하여 바꾸는 것은 가능합니다.
이 예에서는 name의 내용을 "Soprano" 로 assign 했다가
다시 "Anthony Soprano" 로 바꾸는 것을 보여주고 있습니다.
엄밀히 말하면 이 경우 name에 원래 할당되었던 memory가 유지되면서
그 memory의 내용이 바뀌는 것은 아닙니다.
assignment를 새로 할 때마다 새로운 memory space가 할당되기 때문에
String이 여전히 immutable하다는 원칙은 유지가 되는 것입니다.

Character Sets - ASCII

- ASCII: A character set used by many programming languages that contains all the characters normally used on an English-language keyboard, plus a few special characters
  - Each character is represented by a particular number
  - 1 byte (8 bits)

Ctrl	Dec	Hex	Char	Code
^@	0	00		NUL
^A	1	01		SOH
^B	2	02		STX
^C	3	03		ETX
^D	4	04		EOT
^E	5	05		ENQ
^F	6	06		ACK
^G	7	07		BEL
^H	8	08		BS
^I	9	09		HT
^J	10	0A		LF
^K	11	0B		VT
^L	12	0C		FF
^M	13	0D		CR
^N	14	0E		SO
^O	15	0F		SI
^P	16	10		DLE
^Q	17	11		DC1
^R	18	12		DC2
^S	19	13		DC3
^T	20	14		DC4
^U	21	15		NAK
^V	22	16		SYN
^W	23	17		ETB
^X	24	18		CAN
^Y	25	19		EM
^Z	26	1A		SUB
^[	27	1B		ESC
^\	28	1C		FS
^]	29	1D		GS
^^	30	1E	▲	RS
^-	31	1F	▼	US

Dec	Hex	Char
32	20	
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

Dec	Hex	Char
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

Dec	Hex	Char
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	⌂*

* ASCII code 127 has the code DEL. Under MS-DOS, this code has the same effect as ASCII 8 (BS). The DEL code can be generated by the CTRL + BKSP key.

페이지 20

ASCII character set은 기존의 컴퓨터 시스템에서 가장 많이 쓰이던 것입니다.
이 character set에는 영어 키보드 자판에 있는 모든 문자들과
일부 special character를 담고 있습니다.
이 문자들이 모두 128개 이기 때문에
ASCII set은 8bit로 충분히 나타낼 수 있습니다.
이러한 판단에 근거하여
Java 이전의 프로그래밍 언어들에서는
Character type의 크기를 8bit (1 byte) 로 정의한 경우가 많았습니다.
C, C++ 가 대표적인 경우 입니다.

Character Sets - Unicode

- Unicode: A character set used by the Java language
  - includes all the ASCII characters plus many of the characters used in languages with a different alphabet from English (ex. Korean)
  - 2 bytes (16 bits)

페이지 21

그러나 8bit 만으로 영어 알파벳 이외에
외국어 문자들이나 다양한 기호들을 모두 나타내는 것은 불가능하였기 때문에
Java에서는 한 문자를 8bit가 아닌 16bit인 Unicode로 나타내게 되었습니다.
따라서 여러분이 써 보면 아시겠지만
Java 프로그램에서는 한글로 된 variable 이름과 같은 것도 가능합니다.
물론 본 코스에서는 한글 변수 이름을 권장하지는 않습니다.