



JAVA Program 기본 구조

경북대학교
스마트IT과
배희호 교수



Computer가 일을 처리하는 방법

■ 명령대로 수행하는 Computer



Hello, Jane



Hello, Tom



Hello, Java라고
출력해라



```
System.out.println("Hello, Java");
```



Hello, Java

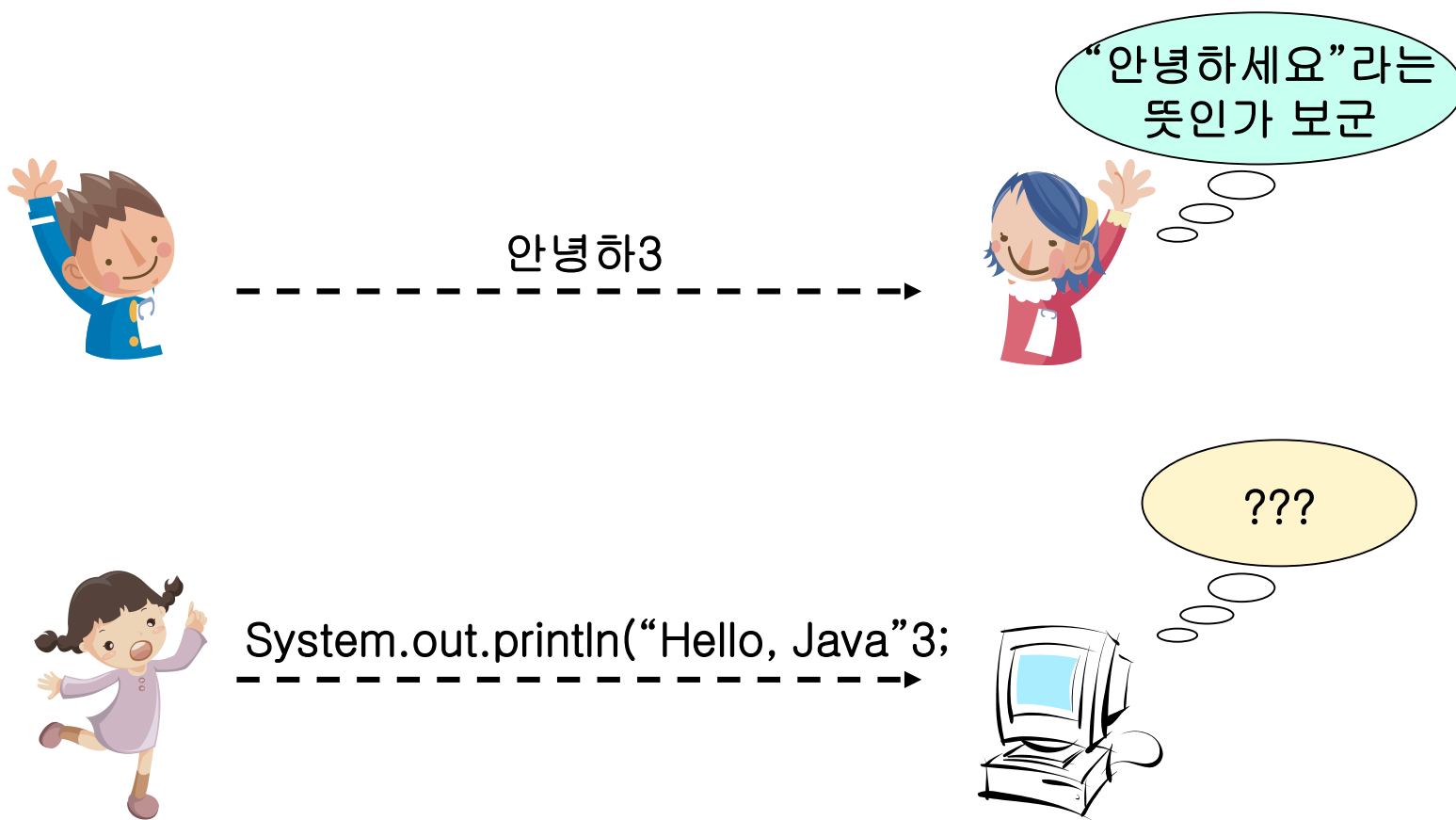
네, 시키는 대로
하겠습니다





Computer가 일을 처리하는 방법

■ 문법이 조금만 어긋나도 못 알아듣는 Computer





Unicode



- 기존의 8bit 문자 체계인 ASCII(American Standard Code for Information Interchange), EBCDIC(Extended Binary-Coded Decimal Interchange Code)는 **모든 언어에 필요한 문자를 수용할 수가 없음**
- 여러 나라의 문자를 모두 수용할 수 있게 하기 위해 16bit Code를 사용하여 문자를 표현하는 표준 문자 체계를 채택하여 65,535 문자를 수용할 수가 있음
- 이렇듯 많은 문자 Code가 수용 가능하므로, 수천 개의 문자 Code가 필요한 한자(漢字) 등의 문자를 포함하는 각 나라의 모든 문자를 할당 할 수 있음
- JAVA는 문자와 문자열을 포함하기 위해 내부적으로 Unicode를 지원하므로, 잠재적으로 국제 언어를 모두 포괄할 수 있음



UniCode



- 국제 표준으로 제정된 2 Byte계의 만국 공통의 국제 문자 부호계(UCS)를 줄여서 부르는 말
- 미국의 Apple 컴퓨터 회사, IBM사, MicroSoft사 등이 제안
 - 이를 추진하기 위하여 설립된 유니코드(Unicode)사가 1990년에 발표한 유니코드 버전을 ISO/IEC JTC 1에서 1995년 9월에 국제 표준으로 제정



UniCode



- 공식 명칭 : ISO/IEC 10646-1 (Universal Multiple-Octet Coded Character Set)
 - 문자 하나하나에 부여되는 Data 값을 모두 16bit로 통일
 - 문자 간 호환성을 제고함으로써 Computer에 의한 Data의 교환이 용이
 - 즉 현재 각국에서 사용 중인 Code의 1 문자당 값이 영어는 7bit, 비 영어는 8bit, 한글이나 일본어는 16bit 등으로 각기 다른 것을 모두 16bit로 통일
 - ISO/IEC 10646-1의 문자판(plane)에는 전 세계에서 사용하고 있는 **26개 언어의** 문자와 특수 기호에 대해 일일이 Code값을 부여



Unicode



- 최대 수용 문자 수는 65,536자
 - 여기에 포함된 한글 코드 체계
 - 옛 한글의 자모를 포함한 한글 자모 240자(HANGUL JAMO, 11열)
 - 현재 한국 표준인 KSC 5601의 조합형 한글 자모 94자(HANGUL COMPATIBILITY, 31열)
 - 현재 한글에서 구현할 수 있는 최대 글자 수 1만 1,172자를 가나다순으로 배열해 놓은 완성형(HANGUL, AC열~D7열) 등 3종으로 되어 있음
- 각국의 문자를 2Byte로 수용하기 위해 한·중·일·대만의 한자를 통합
 - 즉 한국, 일본, 중국, 대만에서 쓰고 있는 님은 한자에 대해 동일한 2Byte의 코드를 할당



JAVA 기본 구성 요소



■ 문자 집합

■ JAVA Program에서 사용되는 문자들

- 영문자 ('A' ~ 'Z')의 대/소문자
- 16Bit Unicode로 표현
- ASCII Code와 EBCDIC Code 문자를 포함

■ Space

- 공백은 Keyword와 Keyword 또는 Keyword와 식별자간을 서로 구분해 주기 위해서 사용됨
 - Keyword와 식별자는 반드시 하나 이상의 공백으로 분리 되어야 함
- JAVA에서 Space, Tab('\t'), new Line('\n')이 공백의 역할을 할 수 있음



JAVA 기본 구성 요소



■ 대소문자 구분

- 관례적으로 **class(클래스)** 이름의 첫 글자는 대문자로 작성
- Method(메소드)와 Object(객체) 이름은 첫 글자는 소문자로 새로운 단어의 글자는 이해 쉽게 대문자로 작성
- 기본형 변수는 일반적으로 첫 글자는 소문자로 시작하고 새로운 낱말의 시작은 대문자로 표기하여 쉽게 이해될 수 있도록 함
- 예) `int myAge;`

■ 문장 구분자 ; (semicolon)

■ 문장 그룹(복합 문장) { } (brace)





JAVA 기본 구성 요소



■ Token

- Program을 구성하는 구문적인 의미의 최소 단위
- 문장을 이루는 기본 단위인 단어

■ Token의 종류

- 언어 설계 시 미리 정의된 Token
- 사용자가 정의한 Token

토큰 public, class, Java, {, int, total, ;, }

```
public class Java {  
    int total;  
}
```



JAVA 기본 구성 요소



- 미리 정의된 Token
 - 언어 자체에서 제공되는 Token
 - Comment(주석, 설명)
 - 작성한 Program에 대한 설명을 기술
 - Compile시 무시됨(제거됨)
 - Keywords(키워드)
 - 언어 설계 시 그 기능과 용도가 미리 지정
 - JAVA에서는 모든 Keyword는 소문자로 구성
 - Separators(구분자)
 - 문장을 구분
 - Operators(연산자)
 - 수식을 구성



JAVA 기본 구성 요소



- 사용자 정의 Token
 - 이름을 나타내는 Identifier(식별자)
 - 식별자 : A~Z, a~z, _, \$로 시작
 - 값을 표현하는 Literal(리터럴)
 - 리터럴에는 자료의 형태에 따라 정수형, 실수형, 논리형, 문자형, 문자열이 존재



Comments



- Program 실행에는 전혀 영향을 주지 않고, Program의 이해를 돕기 위한 설명문(주석문)
- Programmer가 Program이나 명령어를 해설하기 위한 용도로 적은 문장
- 주석문 내의 주석문은 Error로 처리



Comments



■ 주석문을 표현하는 3가지 방법

■ Block 주석(`/* TEXT */`)


- 여러 줄에 걸쳐 주석 문을 만들고자 할 때 사용
- 주석의 시작과 끝을 `/*`와 `*/`로 표시

■ Line 주석(`// TEXT`)

- `//`에서 줄의 끝까지가 주석
- 한 줄로 구성된 주석만 가능

■ Document 주석(`/** DOCUMENTATION */`)

- `/**`에서 `*/`까지가 주석이 됨
- 선언문 앞에서만 사용 가능
- JDK에 포함된 javadoc Program이 `/**`에서 `*/`까지의 내용을 가지고 주석을 추출하여 자동으로 Source를 설명하는 HTML 문서를 만들어 줌



Your comment here...



Comments

- Document 주석 (/** DOCUMENTATION */)
 - 변수나 메소드 또는 클래스의 선언 바로 앞에서 사용
 - HTML 문서 생성시에 해당 주석 문장을 보여줌
 - Javadoc 사용
 - javadoc '패키지명' *.java javadoc을 통해 소스 파일을 API 문서로 만들 때, /**와 */ 사이의 문자들을 주석으로 처리

/**

이 주석은 시스템에서 자동적으로 발생하여 주는 주석이므로 프로그래머는 가급적 사용하지 말아 주시기 바랍니다.

*/



Comments



■ 주석을 이용한 코드의 예제

```
/**  
 * <pre>  
 * 1. 설명 : 자신을 제외한 약수의 총합을 구한다.  
 * 2. 로직 : 1부터 n-1까지의 모든 약수를 합한다.  
 * @param n 약수의 총합을 구하려는 정수  
 * @return 약수의 총합  
 * </pre>  
 */
```

```
public int sumDivide(int n) {  
    int total = 1;           // 1은 항상 포함  
    for (int i = 2; i < n; i++) {  
        if (n % i == 0)  
            total += i;      // 약수의 합  
    }  
    return total;  
}
```




Comments

- API 문서 작성 주석문
 - 개발자가 만든 클래스의 문서 작성 시 사용하는 주석문
 - 작성법
 - javadoc.exe 명령어를 이용

```
javadoc -d. 소스파일.java
```



Comments



■ API 문서작성 주석문

■ 주석에 @태그를 적용해서 작성할 수 있음

문서 태그	설명
@author	클래스나 인터페이스의 제작자 표시 예) @author kyeomstar
@version	문서의 버전 예) @version 1.2
@return	메소드가 void형이 아닌 경우 return value type을 기술
@exception	메소드가 발생 시킬 수 있는 예외를 기술
@see	추가 또는 관련 내용 참고 예) @see 클래스#메서드(아규먼트 타입) @see java.util.Vector#addElement(java.lang.Object)
@since	언제부터 사용하였는가 예) @since version 1.1
@param	매개변수에 대한 설명 예) @param int 돈 액수를 입력
@throws	@exception Tag와 동일
@deprecated	다음버전에서 폐기된 메소드를 알림
@serial	기본적으로 직렬화 할 수 있는 클래스의 멤버를 설명



Comments

■ API 문서작성 주석문

```
1  /**
2  * MyDate class는 날짜에 대한 클래스입니다..
3  *
4  * @author kyong yeol IN
5  * @version 2.0 April 2007
6  *
7  * @see 소스코드
8  *
9  */
10
11 public class MyDate{
12     int year;
13     int month;
14     int day;
15
16     /** 생성자입니다.
17     */
18     public MyDate(int new_year,int new_month, int new_day){
19         year = new_year;
20         month = new_month;
21         day = new_day;
22     }
23
24     /** 년도를 세팅합니다
25     */
26     public void setYear(int new_year){
27         year = new_year;
28     }
29     /**월을 세팅합니다.
30     */
31     public void setMonth(int new_month){
32         month = new_month;
33     }
```

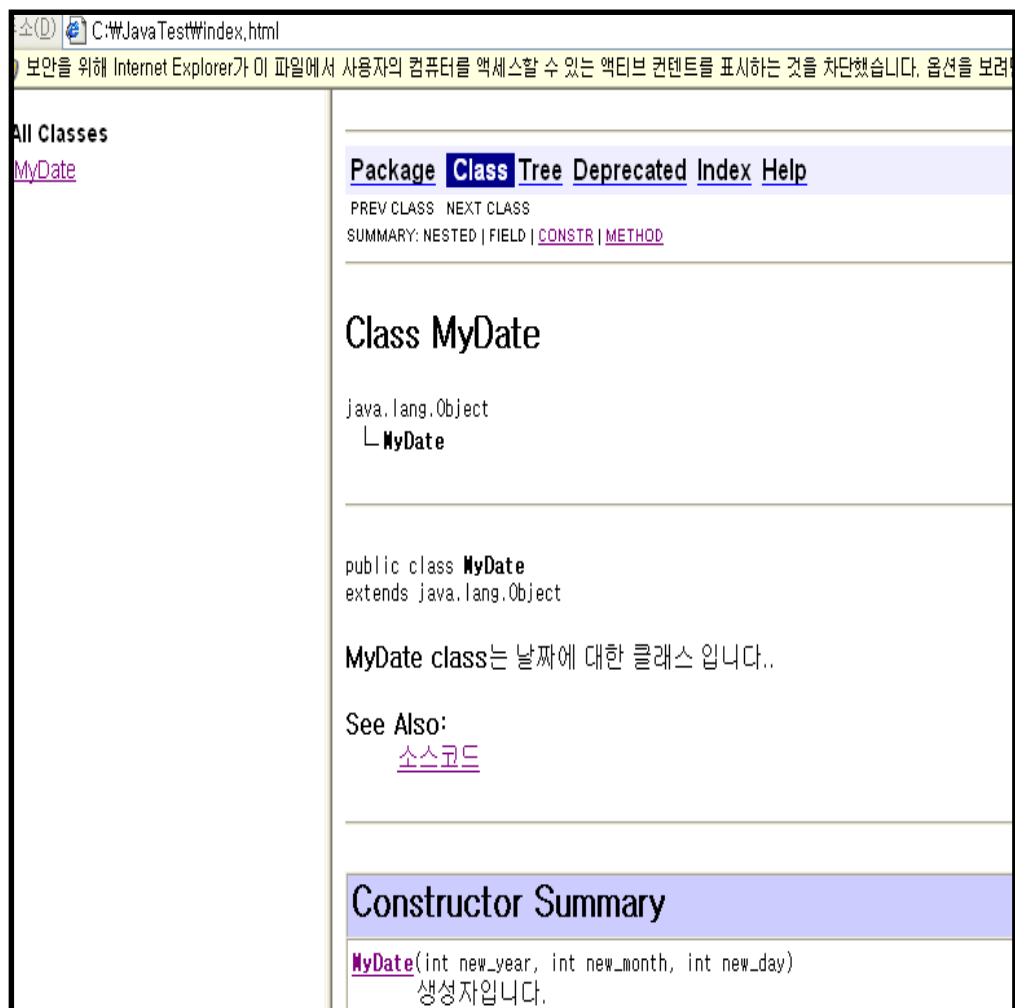
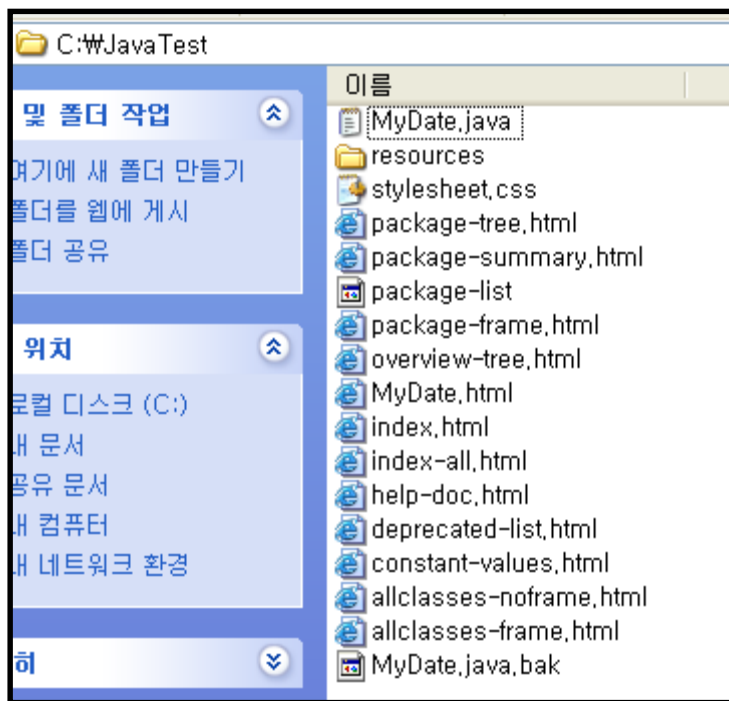
```
c:\JavaTest>javadoc -d . MyDate.java
Loading source file MyDate.java...
Constructing Javadoc information...
Standard Doclet version 1.6.0
Building tree for all the packages and classes...
Generating .\MyDate.html...
Generating .\package-frame.html...
Generating .\package-summary.html...
Generating .\package-tree.html...
Generating .\constant-values.html...
Building index for all the packages and classes...
Generating .\overview-tree.html...
Generating .\index-all.html...
Generating .\deprecated-list.html...
Building index for all classes...
Generating .\allclasses-frame.html...
Generating .\allclasses-noframe.html...
Generating .\index.html...
Generating .\help-doc.html...
Generating .\stylesheet.css...

c:\JavaTest>
```



Comments

■ API 문서작성 주석문





Javadoc



```
MS 한글 MS-DOS
8 x 12
디렉터리 C:\temp\W02\장
HELLOW~1 HTM          217  98-01-07   7:47 HELLOWORLD.HTML
      1개 파일                217 바이트
      0개 디렉터리          1,978.51 MB 사용 가능

C:\temp\W02\장>javadoc AppletHelloWorld.java
Loading source file AppletHelloWorld.java...
Constructing Javadoc information...
Building tree for all the packages and classes...
Building index for all the packages and classes...
Generating overview-tree.html...
Generating index-all.html...
Generating deprecated-list.html...
Building index for all classes...
Generating allclasses-frame.html...
Generating index.html...
Generating packages.html...
Generating AppletHelloWorld.html...
Generating serialized-form.html...
Generating package-list...
Generating help-doc.html...
Generating stylesheet.css...

C:\temp\W02\장>
```



Javadoc



Class AppletHelloWorld - Microsoft Internet Explorer

파일(F) 편집(E) 보기(V) 이동(G) 즐겨찾기(A) 도움말(H)

뒤로 앞으로 멈출 새로고침 시작 검색 즐겨찾기 목록보기 채널 전체화면 메일 글꼴 인쇄

주소 C:\temp\W02장\WAppletHelloWorld.html

연결 골드뱅크 대우증권 동아일보 조선일보 매일경제 MBC 중앙일보

Class [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) [DETAIL: FIELD](#) | [CONSTR](#) | [METHOD](#)

Class AppletHelloWorld

```
java.lang.Object
|
+--java.awt.Component
|   |
|   +--java.awt.Container
|       |
|       +--java.awt.Panel
|           |
|           +--java.applet.Applet
|               |
|               +--AppletHelloWorld
```

```
public class AppletHelloWorld
extends java.applet.Applet
```

내 컴퓨터



/** */ 주석문



```
APPLETHELLOWORLD.JAVA - 메모장
파일(F)  편집(E)  찾기(S)  도움말(H)

import java.awt.Graphics;

public class AppletHelloWorld extends java.applet.Applet
{
    /** 이것이 보일까 */
    public void paint(Graphics g)
    {
        g.drawString("Hello World!", 10, 10);
    }
}
```



Separator



- Separator는 문장을 구분하고 띄어쓰기 역할
- 가장 많이 사용되는 구분자는 **세미콜론(;)**과 **кома(,)**
 - **Semicolon(; , 세미콜론)**은 문장을 종결할 때 사용하며, 문장과 문장을 구분하여 줄 (C언어와 동일)
 - **Comma(кома)**는 변수 선언에서 연속적인 식별자들을 구분하거나, for문 내부에서 문장들을 나열하기 위해서 사용됨
 - 예) 코마를 사용한 for문

```
for (a = 1, b = 3; a < 10; a++, b++)  
    sum = a + b;
```
- Block은 중괄호({, })로 묶여진 부분(C언어와 동일)
- Whitespace는 space, tab, new line을 총칭하는 말(C언어와 동일)



Separator

- 구분 문자(separator)는 문장을 구분하고 띄어쓰기 역할
- JAVA의 구분자는 C의 구분자와 유사함

구분 문자	사용 용도	예
[]	배열 참조형 선언, 배열 첨자	<code>int[] test = new test[5];</code>
{ }	클래스, 메소드, 정적 초기화 블록, 제어 블록	애플리케이션 및 애플릿 작성한 내용 참고
()	매개변수, 실행 우선순위, 형변환	<code>(matrix + 3) / 4</code> <code>total(3, 4)</code>
;	실행문의 마지막	
:	라벨 선언	
.	패키지 이름 구분, 객체의 멤버를 참조	<code>System.out.println("안녕");</code>
,	인터페이스 선언, 변수명 선언, for 초기 값 증가값	
여백	구성요소 구분(공백, 탭, 개행), 들어쓰기	



Separator



■ Semicolon(;)

- JAVA에서 한 문장이 끝날 때 반드시 끝에 semicolon(;)을 붙여야 함. 붙이지 않으면 하나의 명령으로 인식하지 못하며 ;으로 끝날 때까지 한 명령으로 인식

```
System.out.println( "a" + name + "b" );
```

- 위 예제는 하나의 Coding이 끝난 후에 ;을 했기 때문에 정상적

```
System.out.println( "a" +  
name + "b" );
```

- 위 예제는 한 줄이 끝나도 ;이 없기 때문에 다음 줄로 내려가 ;을 만날 때까지 하나의 명령으로 인식



Separator



■ Block({ })

- Block이라고 하는 것은 curly brace {, }로 묶여진 부분을 말함. JAVA에서는 범위에 해당하는 것은 반드시 {}로 묶어야 함. 제어구조에서 범위가 한 줄인 경우에는 생략할 수 있음

```
class A {  
    int a ;  
    int b ;  
}
```

- A라는 클래스를 지정한 후 A 클래스는 두 줄을 포함하고 있음



Separator



```
public static void main( String [] args ){  
    System.out.println( "a" );  
}
```

- main 메소드는 한 줄을 포함하고 있음

```
if( a > 0 )  
    System.out.println( "a" );
```

- if문의 true 문장은 한 줄이기 때문에 {}를 생략



Separator



■ whitespace

- whitespace라고 하는 것은 space, tab, new line character(흔히 enter key라고 함)를 총칭
- JAVA에서는 이 3개의 문자가 source code안에 얼마든지 포함되어도 문제 없음

```
public    static void main( String [] args ){  
  
    System.out.println ( "a"    + b ) ;  
}
```

- 얼마든지 넣어도 상관은 없지만 보기에 안좋으므로 잘 넣어야 함

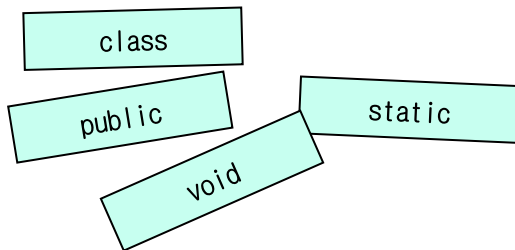


JAVA Program의 구성요소

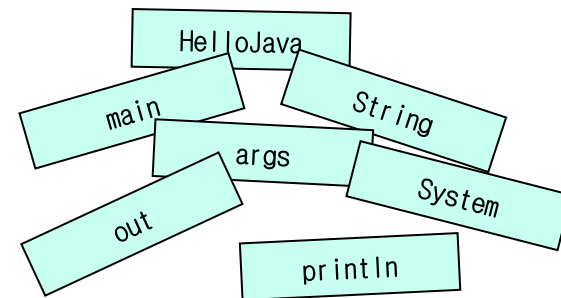
■ 단어

- 키워드(keyword), 식별자(identifier), 상수를 표현하는 단어

```
public class HelloJava {  
    public static void main(String args[]) {  
        System.out.println("Hello, Java");  
    }  
}
```



키워드(keyword)



식별자(identifier)



Reserved Words



■ Literal(리터럴)

- 즉시 Data라고 번역할 수 있으며, 변수의 특정 값을 나타내는 문자열을 말함

■ 예)

- true, false, 12, 121.8, 'x',
- “Java is a good language”

■ Reserved word(예약어)

- JAVA 언어에서 정의한 목적으로만 사용되는 단어
- 사용자가 식별자(Identified, 변수의 이름, 메소드의 이름 등)로 정의할 수 없음
- JAVA System에 의해서 예약되었기 때문에 다른 목적으로 사용될 수 없다는 의미임

■ 예)

- if, for, class 등의 명령어로 사용되는 단어들



Futuristic Innovator

京福大學校
KYUNGBOK UNIVERSITY



Reserved Words



- Program 구문 또는 구조를 나타내는 미리 약속된 식별자
- 다른 식별자로 사용할 수 없음
- 예약어는 명령어와 같이 내부에서 기능으로 사용하는 이름
 - JAVA System에서 사용하는 단어로 사용자는 식별자로서 이용할 수 없음
 - 예) static, void, public, int, byte, long 등
- Keyword
- 총 50개의 예약어
- Programmer가 실수로 예약어를 식별자로 사용 하였을 경우, JAVA Compiler는 **Error Message**를 출력함



Reserved Words



abstract	continue	for	new	switch
assert	default	if	package	synchronized
boolean	do	goto	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while



Reserved Words



분류	키워드
소스 파일 구성	import, package
클래스/인터페이스 선언, 인스턴스 생성	class, interface, new
기본 데이터형	boolean, char, byte, short, int, long, float, double
데이터 값	true, false, null
클래스 상속	extends, implements
생성자	super, this
연산자	instanceof
접근제한자	public, protected, private
기타제한자	abstract, static, final, native, synchronized
제어문	if, for, while, do, else, break, continue, default, return, switch, case
예외처리	try, catch, finally, throws, throw
기타	transient, goto, const, volatile



Reserved Words

- const, goto 등도 정의는 되어 있지만 실제로는 사용하지 않는 Keyword (권고)
 - const는 C언어에서 사용되는 것으로 상수를 정의할 때 사용(JAVA에서 final)
 - goto는 옛날 GW-Basic배울 때 많이 썼던 goto문!, 절차지향 언어에서 주로 사용
- JAVA 번역기에 의해 기능과 용도가 지정되어 있는 단어
- 사용자가 임의로 다른 목적이나 의미로 바꾸어 사용 불가능
- 상수 값 true, false, null은 소문자



Identifier



- Programmer가 정의할 수 있는 프로그램 요소(element)의 이름
- 식별자는 변수, 상수, 배열, 문자열, 사용자가 정의하는 Class나 Method 등을 구분할 수 있는 이름
- 영어의 대소문자, 숫자, 언더라인 문자(_), \$기호 등을 사용하여 식별자를 구성할 수 있음
- 숫자가 식별자의 첫 문자로 사용되어서는 안 됨
- JAVA에서는 **대소문자를 구분하므로** intValue와 IntValue는 서로 다른 식별자이므로 주의를 요함
- 길이에겐 제한을 두지 않음
- 예약어는 식별자로 사용할 수 없음



Identifier

■ 식별자 작성 원칙

- ‘@’, ‘#’, ‘!’와 같은 특수 문자, 공백 또는 탭은 식별자로 사용할 수 없으나 ‘_’, ‘\$’는 사용 가능
- ‘_’ 또는 ‘\$’를 식별자 첫 번째 문자로 사용할 수 있으나 일반적으로 잘 사용하지 않음
- 공백 문자는 포함할 수 없음
 - 예) ab cd, _b 12 등
- 16 bit Unicode 사용 가능(한글 사용 가능)
- JAVA 언어의 Keyword는 식별자로 사용 불가
- 식별자의 첫 번째 문자로 숫자는 사용 불가
- 불린(boolean) 리터럴(true, false)과 널 리터럴(null)은 식별자로 사용 불가
- 대소문자 구별
 - 예) Test와 test는 별개의 식별자



Identifier



■ 사용 가능한 예

```
int number;
char student_ID;           // '_' 사용 가능
void $func() { }           // '$' 사용 가능하나 첫글자로 잘 사용 안함
class Monster3 { }         // 숫자 사용 가능
int whatsyournamemynameiskitae; // 길이 제한 없음
int barChart; int barchart; // 대소문자 구분. barChart와 barchart는 다름
int 가격;                  // 한글 이름 사용 가능
```

■ 잘못된 예

```
int 3Chapter;              // 식별자의 첫문자로 숫자 사용 불가
class if { }               // 자바의 예약어 if 사용 불가
char false;                // false 사용 불가
void null() { }            // null 사용 불가
class %calc { }            // '%'는 특수문자
```



Identifier

- 식별자 이름 붙이는 관습
 - 기본 : 헝가리안 이름 붙이기
 - 클래스 이름
 - 첫 번째 문자는 대문자로 시작
 - 여러 단어가 복합될 때 각 단어의 첫 문자만 대문자

```
public class HelloWorld { }  
class Vehicle { }  
class AutoVendingMachine { }
```

- 변수, 배열, 문자열, 메소드 등의 이름
 - 첫 번째 문자는 소문자로 시작
 - 첫 단어 이후 각 단어의 첫 문자는 대문자로 시작

```
int iAge;           // iAge의 i는 int의 i를 표시  
boolean bIsSingle;  // bIsSingle의 처음 b는 boolean의 b를 표시  
String strName;     // strName의 str은 String의 str을 표시  
public int iGetAge() { } // iGetAge의 i는 int의 i를 표시
```



identifier



■ 내장 객체(상수) 이름

- 모든 문자를 대문자로 표시
- 둘 이상의 단어가 묶여있는 경우 _(언더바)를 사용한다

```
final static double PI = 3.141592;  
int MESSAGE_LENGTH = 100;
```

종류	사용 방법	예
클래스명	각 단어의 첫글자는 대문자로 한다.	StaffMember, ItemProducer
변수명, 메소드명	소문자로 시작되어 2번째 단어의 첫글자는 대문자로 한다.	width, payRate, acctNumber, getMonthDays(), fillRect(),
상수	상수는 모든 글자를 대문자로 한다.	MAX_NUMBER



Identifier



- 다음은 자바에서 사용 가능한 식별자의 예

Counter	_value	Average_Score
identifier	UserName	User_2_Name
_isdel	\$money	

- 다음은 사용할 수 없는 식별자의 예

ship-price	10times	Yes/No	2identifier
*variable	%lengthOfArray		#abcde
@hello			



Identifier

- class, method, field의 첫 글자는 “\$”, “_”, 영문 대소문자여야 함
 - 예) \$abc, _b12, a12 등
- 글자 수에는 제한이 없음
- 특수문자(@, #, %, ^, &, *, !, ? 등)는 사용할 수 없음
 - 예) ab&dc, k(bb) 등
- 숫자는 첫 글자가 아닐 경우 사용이 가능
 - 예) ab12, \$Rt_2 등
- 예약어는 사용할 수 없음
 - 예) this, for, if, abstract 등
- JAVA에서는 대소문자를 구분하므로 intValue와 IntValue는 서로 다른 식별자이므로 주의를 요함



보기 좋은 Program 작성법

- ① 프로그램의 소스(source)는 보기가 좋고, 이해하기 쉽게
- ② 프로그램은 예상한대로 수행되어 원하는 결과를 얻는 신뢰성(reliability)이 있어야 함
- ③ 프로그램 실행을 위한 기억장소는 가능한 적게 사용하는 것이 좋음
- ④ 프로그램 실행 속도는 가능하면 빨라야 함
- ⑤ 이식성(portability)이 높아야 함



Program을 읽기 좋게 하는 방법

- 읽기 쉬운(readability) 프로그램은 이해하기 쉽고 수정 또는 변경하기 쉬움
- 식별자를 작성시 의미 있는 단어를 사용
int a; (X) int width; (O)
- 하나의 문장은 한 라인(line)에 작성
- 공백 라인을 적절히 사용하여 프로그램을 보기 쉽게 구분
- 들여 쓰기(indentation)을 적절히 사용
- 설명문(comment)을 사용, 친절한 설명을 기술
 - Program의 시작부분에 프로그램의 목적과 작성자, 작성 일, 주의사항 등을 명기
 - Program에서 설명이 필요한 부분에는 적당한 설명문을 달아 줌



Program을 읽기 좋게 하는 방법

- 식별자를 선정할 때 같은 글자가 반복되거나 유사한 문자를 연속 혼용해서 사용하지 않음
 - 예) 틀리기 쉬운 문자에는 1과 l, 0과 O, u와 v, n과 m, j와 i 등
 - $\text{value1} = \text{value1} + \text{value}l;$
 - $\text{term0} = \text{termO} + 6 - \text{term0};$
 - $\text{mnun} = \text{mnvn} * uv + vu;$
- 마지막 글자 하나만 틀린 길이가 긴 식별자의 사용은 피함
 - 예) positionx , positiony 보다는 x_pos , y_pos 를 사용
- 임시변수는 통일해서 사용
 - 예) i, j, k 는 인덱스 또는 제어변수로 사용
 ch 는 문자형 변수



Program을 읽기 좋게 하는 방법

- 주석문은 Program의 내용과 일치하도록 작성

- 예) 세금계산 공식에서

tax = price * rate; /* price에 rate를 곱한다. */

보다는

tax = price * rate; /* 세금계산 공식 */

으로 작성하는 것이 프로그램을 이해하기 쉽게 함

- 하나의 함수는 최대 2~3 페이지 내로 만들어 간단하고 명료하게 설계
- Programming 테크닉에만 관심을 갖지 말고, Program을 어떻게 하면 효율적이고 이해하기 쉬운 Code를 작성할 수 있을 지에 관한 노력에 힘을 쏟아야 함



Program 작성 규칙

- JAVA Program 작성할 때 문법(syntax rule)에만 위배되지 않으면 특별한 제한이 없는 자유 형식(free format)

① 이항 연산자의 양쪽 항에는 공백 (단항 연산자는 붙임)
a = 3; a += 7; x = y + 3; (이항 연산자)
+x; test--; ++a; (단항 연산자)

② 문장을 분리하여 주는 세미콜론(;)은 앞의 문장에 붙임
System.out.println(" coding rule ");
int a; a = 30;

③ 분리자(separator)인 콤마(,)나 세미콜론(;) 다음에는 공백
int a, b, c;
for (i = 1; i < 3; i++)



Program 작성 규칙

- ④ 메소드를 나타내는 소괄호(())와 배열을 표시하는 대괄호([])는 메소드나 배열 이름에 붙여 씀

```
char name[15];           (배열)  
main( )                  (메소드)
```

- ⑤ 제어문에 사용하는 소괄호(())는 공백을 두어 메소드와 구별

```
while ( )    for ( ; ; )    if ( )    switch ( )
```




Program 작성 규칙



- ⑥ 메소드의 본체나 복문을 나타내는 중괄호({ })는 다음과 같이 서로 열(column)을 맞추어 작성

```
main( ) {  
  
    ....  
while (1) {  
    ....  
}  
}
```

```
main( ) {  
  
    ....  
while (1) {  
    ....  
}  
}
```



Program 작성 규칙

⑦ 한 줄에 최대 248자까지 허용하지만 한 줄에 한 문장만 작성

①. `main() { printf ("Welcome to C world !\n"); }`

②. `main()
{
 printf("Welcome to C world !\n");
}`

⑧ 빈 줄(blank line)을 사용하여 기능별로 구분

```
main( )  
{  
    int number;  
    /* 빈줄을 사용하여 선언문과 실행문을 구분 */  
    number = 5;
```



Program 작성 규칙

- ⑨ Program의 구조를 명확하게 하기 위하여 **행 들여놓기 (indentation)**를 사용하여 블록 단위로 구분

```
void main(void)
```

```
{
```

```
    int index;
```

```
    for (index = 1; index < 10; ++index)
```

```
    {
```

```
        if (index ==
```

```
            break;
```

```
        printf("%d\\n", index);
```

```
    }
```

```
}
```

행 들여놓기는 프로그램을 작성하는 사람의 보조수단이지 시스템은 전혀 상관이 없다

- ⑩ Program 작성 당시에 주석문을 철저하게 기입





Program 작성 규칙

■ 들여쓰기(indentation)

```
import java.util.Scanner;
public class SumAndAverage {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int sum = 0;
        for (int i = 0; i < 10; i++) {
            sum += input.nextInt();
        }
        System.out.println("정수의 합 = " + sum);
        System.out.printf("평균 = %.2f\n", (double) sum/10);
    }
}
```



Program 작성 규칙



■ 들여쓰기(indentation)

```
int count = 0;
for (int i = 0; i <= upperBound; i++){
    System.out.print(i + '\t');
    count++;
    if (count == 10)
        System.out.print(i + '\t');
        count = 0;
}
```





변수 표기 방법



■ Hungarian Notation

- 헝가리 출신 MicroSoft사의 개발자 Charles Simony에 의해 만들어져 사용되는 표기법

■ 선언 방법

- 변수를 선언할 때 형을 구분할 접두사를 이용하여 복합어를 만듦
- 변수의 자료형 - 알기 쉬움
- 예) 변수 선언 `int iNumber;`
`boolean bBool;`
`char chtest;`



변수 표기 방법



■ Hungarian Notation 단점

- 변수가 선언이 안되었는데도 변수의 타입과 종류를 알 수가 있음
- 가독성이 떨어짐
- 개발 도구의 발전으로 타입의 확인이 쉬워졌음
- 해당 변수의 타입이 바뀌면 변수의 이름까지 바꾸어야 함

■ Hungarian Notation 장점

- 변수 이름만으로 그 변수가 무슨 일을 하는지 알 수 있음
- 함수 이름에 대하여 함수 이름 또한 변수 이름과 거의 같음

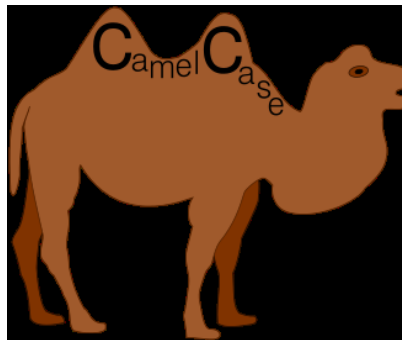


변수 표기 방법



■ Camel Case

- 단어와 단어 사이를 대문자로 구분하는 방법으로, 낙타의 혹을 닮아서 붙여진 이름
- 변수명이 소문자로 시작하며, 복합어일 경우 중간에 시작하는 새로운 단어는 대문자(단봉 낙타 표기법)
- 예)
 - `customerName`, `guestbook`, `backgroundColor`, `typeName`, `iPhone`





변수 표기 방법

■ Camel Case 규칙

- 클래스, 변수, 메소드의 이름을 지을 때 대부분 그 기능을 알 수 있도록 이름을 생성
- 하나 또는 여러 개의 단어의 조합으로 클래스, 변수, 메소드가 어떤 기능을 하는지 알 수 있도록 이름을 생성
- JAVA 프로그래머들이 일반적으로 클래스나 변수, 메소드의 이름을 정의하면서 쓰는 규칙은 **Camel Case 규칙**을 따라 이름을 생성





변수 표기 방법

■ Camel Case 규칙

- 첫 문자는 대문자로 시작
- 둘 이상의 단어가 묶여있는 경우 각 단어의 첫 글자를 대문자로 한다
- 클래스의 이름은 Camel Case의 기본 규칙을 따라 작성하는 것이 일반적

```
class NewClass  
class SimpleClass
```

- 메소드와 변수의 경우는 클래스와 구분하기 위해 Camel Case의 기본 규칙을 약간 변형해서 사용
 - 첫 번째 문자는 대문자가 아닌 소문자로 시작



변수 표기 방법

■ Camel Case 규칙

- 변수와 메소드에 적용되는 규칙을 정리하면 아래와 같음
 - 첫 문자는 소문자로 시작
 - 둘 이상의 단어가 묶여있는 경우 각 단어의 첫 글자를 대문자로 한다

```
int num;           // 변수  
int nextInt();     // 메소드
```

■ 상수는 변수와 구분되도록 다른 규칙을 적용

- 모든 문자는 대문자로 작성
- 둘 이상의 단어가 묶여있는 경우 _(언더바)를 사용

```
final int PI = 3.14;  
final int MESSAGE_LENGTH = 100;
```



변수 표기 방법



- Pascal Case

- 쌍봉 낙타 표기법
- 첫 단어를 대문자로 시작하는 Camel 표기법
- 띄어쓰기 대신 대문자로 단어를 구분하는 표기 방식
- 예)
 - BackgroundColor, TypeName, PowerPoint



변수 표기 방법



■ Snake Case

- 변수를 정의하는 단어와 단어 사이를 밑줄(_)을 사용하여 표시
- 예)
 - name_of_variable, background_color, type_name
- 주로 Snake Case를 사용하는 언어들
 - Perl
 - PHP 변수명, 함수명, 메소드명
 - Python 변수명, 함수명, 메소드명
 - Ruby
 - Rust 함수명



변수 표기 방법(관례)

■ 클래스 이름

- 명사형 단어 사용
- 첫 자는 대문자로 시작하고 두개의 단어 이상이 결합할 때는 첫 자는 대문자로 사용

Customer, AccountTest, AddressBook

■ 메소드 이름

- 메소드 이름은 주로 동사형을 사용
- 첫 자는 소문자로 시작하고 두개의 단어 이상이 결합할 때는 첫 자는 대문자로 사용

print, getName, setLocation



변수 표기 방법(관례)

■ 변수 이름

- 변수 이름은 주로 명사를 사용
- 첫 자는 소문자로 시작하고 두 개의 단어 이상이 결합할 때는 첫 자는 대문자로 사용

accountName, balance, price



JAVA Program 구조

```
package 패키지 이름;
```

```
import 패키지 이름/클래스 이름;
```

```
interface 인터페이스 이름 [extends 인터페이스 이름]
```

→인터페이스 헤더

```
{  
    // 상수 선언  
    // 추상 메소드 선언  
}
```

→인터페이스 바디

```
class 클래스 이름 [extends 클래스 이름]  
    [implements 인터페이스 이름]
```

→클래스 헤더

```
{  
    // 멤버 변수 선언  
    // 메소드 선언  
}
```

→클래스 바디



화면에 출력하기

```
1 : public class Output
2 : {
3 :     public static void main(String[] args)
4 :     {
5 :         System.out.print("Java2 ");
6 :         System.out.println("Programming");
7 :         System.out.println("자바2\n프로그래밍");
8 :     }
9 : }
```

System.out.print	화면에 출력(위치 유지)
System.out.println	화면에 출력 후 줄 바꿈



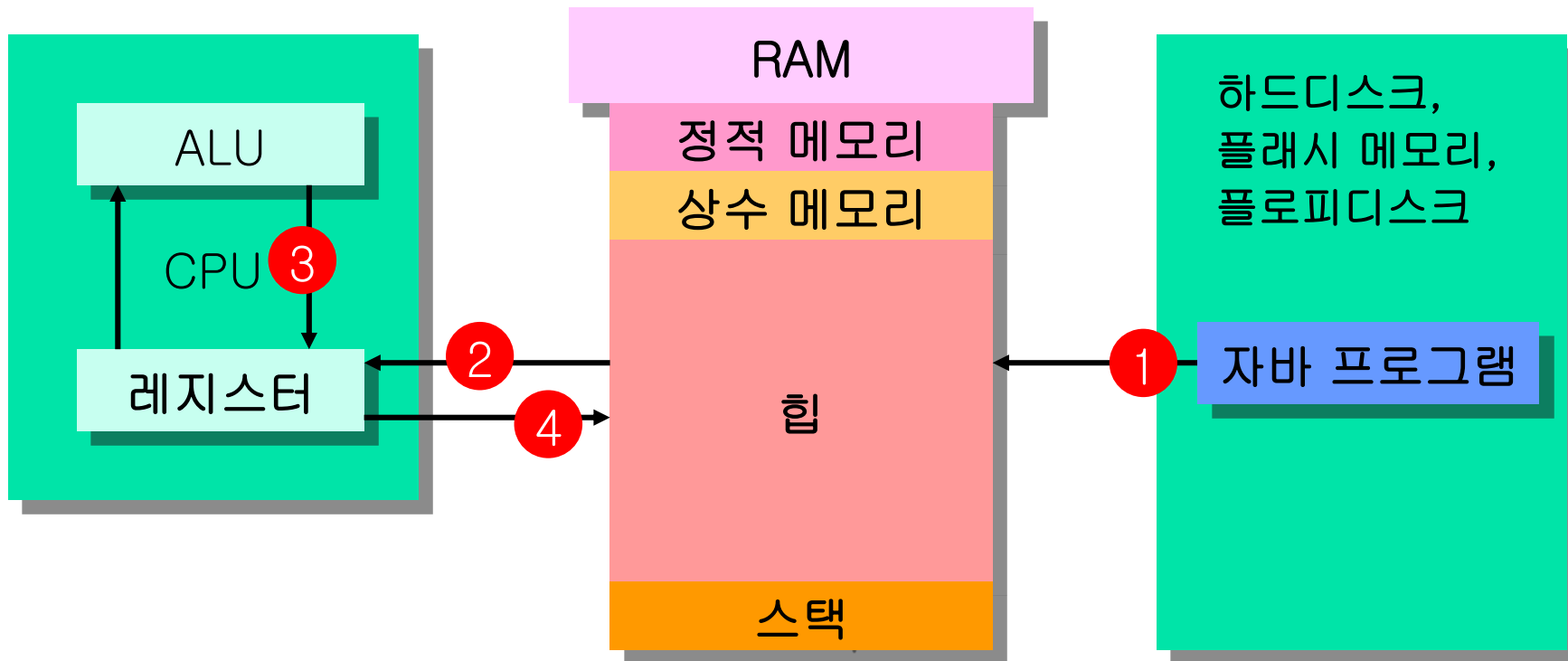
Futuristic Innovator

京福大學校
KYUNGBOK UNIVERSITY



Data 저장

Program 실행 과정



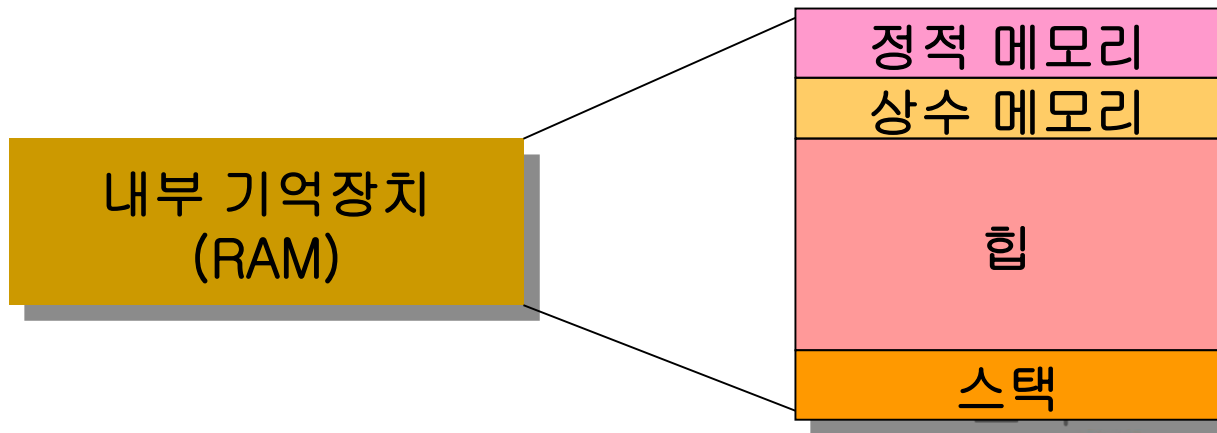
- 1** 프로그램 로딩 **2** 페치(fetch) **3** 연산 **4** 메모리 저장



기억 장치 구성



레지스터	CPU 내부의 고속 기억장치
내부 기억장치	RAM과 같이 CPU 버스와 직접 연결된 메인 메모리
스택(stack)	메소드와 변수를 위한 내부 기억장치(RAM)영역
힙(heap)	객체 할당을 위한 내부 기억장치 영역
정적 메모리	공유 메소드나 공유 변수를 위한 내부 기억장치 영역
상수 메모리	변경되지 않는 값을 위한 내부 기억장치 영역
외부 기억장치	하드디스크, 플로피디스크, 플래시메모리 같은 보조 기억장치





내부 기억 장치



스택(stack)	메소드와 변수를 위한 내부 기억장치(RAM)영역 최종 위치만을 가지고 운영되는 데이터 형태(LIFO) 가변적인 메소드 호출과 메소드 내부에서 선언된 지역변수 저장
힙(heap)	객체 할당을 위한 내부 기억장치 영역(스택 메모리를 제외한 영역) 객체 생성될 때 필요한 데이터 영역 - 랜덤 할당/해제 책꽂이에 비유 - 필요한 만큼 이용하다 필요 없으면 해제
정적 메모리	공유 메소드나 공유 변수를 위한 내부 기억장치 영역 코드 영역 : 위치 결정되면 변하지 않는 명령어 집합 영역 데이터 영역 : 클래스 당 하나씩 할당, 객체 생성과 상관 없이 메모리 차지
상수 메모리	변경되지 않는 값을 위한 내부 기억장치 영역