제10장 프로그래밍 언어 활용

제1절 구조적 프로그래밍과 객체지향 프로그래밍

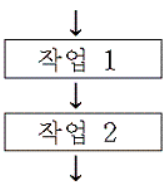
1. 구조적 프로그래밍(Structured Programming)

- 구조적 프로그래밍에 대한 개념은 프로그래밍 내에「GO TO 문」을 사용하므로서 발생되는 문제점을 없애려고 시작되었다. 따라서 GO TO문을 가능한 사용하지 않고 프로그래밍하는 것을 구조적 프로그래밍의 기본이라 할 수 있다.

(1) 구조적 프로그래밍의 세부 개념

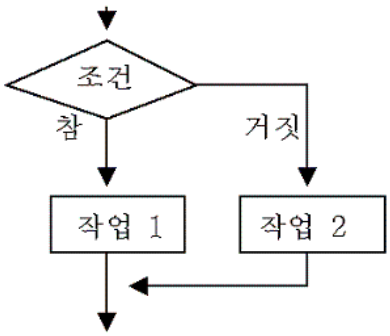
① GO TO 문을 가능한 사용하지 않고 프로그램을 작성한다.
② 논리구조는「순차, 반복, 선택」만을 사용하여 프로그램을 작성한다.
③ 프로그램 설계는 위에서 아래로 하향식 기법으로 하고, 처리 내용은 기능별로 분할하여 모듈 단위로 구성한다.
④ 각 모듈은 하나의 입구와 출구를 가지게 하며, 모듈별로 가능한 독립적이 되도 록 한다.
⑤ 프로그램의 외관적인 형태도 구조적이 되도록 코딩한다.

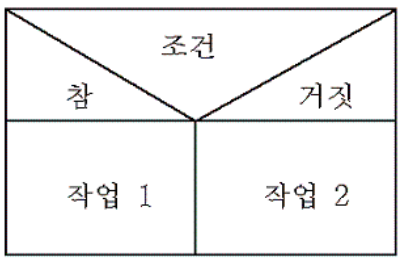
2) 구조적 프로그래밍의 논리구조
① 순차구조 : 하나의 작업이 수행되고 순차적으로 다음 작업을 진행한다.



(N-S 차트)

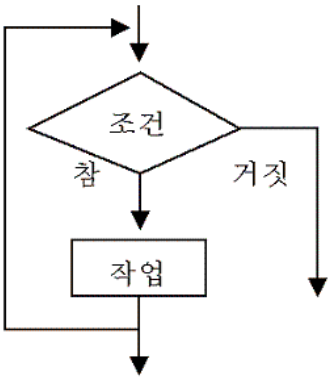
② 선택구조 : 조건에 따라 하나의 작업을 선택해서 진행한다.

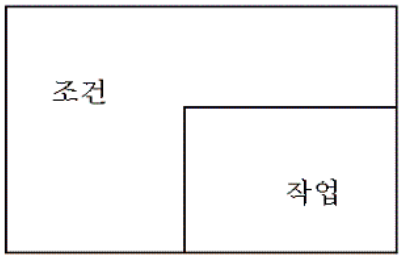




(N-S 차트)

③ 반복구조 : 조건에 따라 특정 작업을 반복 처리한다.





(N-S 차트)

(3) 구조적 설계의 효과

① 기존의 방식에 비하여 보다 많은 규칙성을 부여함으로써, 설계 시간이 단축되고 프로그램의 정확도가 높아진다.
② 기본적인 논리 구조만을 가지고 설계함으로써, 프로그램의 구조를 보다 간결하게 표현할 수 있다.
③ 프로그램을 모듈화함으로써 오류 수정 및 삽입과 삭제가 용이하다.
④ 작업의 흐름과 코딩 순서가 일치하므로, 논리 흐름을 쉽게 이해할수 있다.

2. 객체지향 프로그래밍

(1) 객체지향의 개요

① 1966년 Simula 67 프로그래밍 언어를 개발하면서, 시스템의 한 구성원으로서 한 행위를 행할 수 있는 하나의 단위로 객체라는 개념을 사용했다.
② 객체 지향 기법에서의 시스템 분석은 문제 영역에서 객체를 정의하고, 정의된 객체들 사이의 상호작용을 분석하는 것이다.
③ 객체 지향 기법은 복잡한 시스템의 설계를 단순하게 한다. 시스템은 하나 또는 그 이상의 규정된 상태를 갖는 객체들의 집합으로 시각화될 수 있으며, 객체의 상태를 변경시키는 연산은 비교적 쉽게 정의된다.
④ 소프트웨어 설계 개념의 추상화, 정보은닉, 그리고 모듈성에 기초한다.
⑤ 상속성, 상향식 방식, 캡슐화, 추상데이터형을 이용한다.
⑥ 하나의 객체지향 프로그램은 여러개의 객체들로 구성되며, 각 객체는 소수의 데이터와 이 데이터 상에 수행되는 소수의 함수들로 구성된다.
⑦ 객체 지향 시스템에서는 객체라는 개념을 사용하여 실세계를 표현 및 모델링하며, 객체와 객체들이 모여 프로그램을 구성한다. 전체 시스템은 각각 자체 처리 능력이 있는 객체로 구성되며, 객체들 간의 상호 정보 교환에 의해 시스템이 작동한다.

(2) 객체지향의 기본 개념

① 객체(Object)
㉠ 데이터와 그것을 사용하는 연산을 하나의 모듈로 구성한 것으로, 개별 자료 구조와 프로세스들로 구성된다.
(데이터(속성) + 연산(메소드) → 캡슐화)
㉡ 객체는 인터페이스인 공유부분을 갖으며, 상태(state)를 가지고 있다
㉢ 객체는 하나의 실체로 그 실체가 지닌 특징과 그 실체가 할 수 있는 행동방식으로 구성된다.
㉣ 프로그램 상에서 각 객체는 필요로 하는 데이터와 그 데이터 위에 수행되는 함수들을 가진 작은 소프트웨어 모듈이다.

② 속성(attribute)
㉠ 객체가 가지고 있는 특성으로, 현재 상태(객체의 상태)를 의미한다.
㉡ 속성은 개체의 상태, 성질, 분류, 식별, 수량 등을 표현한다.

③ 클래스(class)

㉠ 개요
ⓐ 동일한 속성, 공통의 행위, 다른 객체 클래스에 대한 공통의 관계성, 동일한 의미를 가지는 객체들의 집합으로 모든 객체는 반드시 클래스를 통해서 정의될 수 있다
ⓑ 클래스라는 개념은 객체 타입으로 구현된 소프트웨어를 의미한다. 클래스는 동일한 타입의 객체들의 메소드와 변수들을 정의하는 템플릿(templete)이다.
ⓒ 하나 이상의 유사한 객체들을 묶어 공통된 특성을 표현한 데이터 추상화를 의미한다.
ⓓ 클래스내의 모든 객체들은 속성의 값만 달리할 뿐, 동일한 속성과 행위를 갖게 된다.

㉡ 추상 클래스(abstract class)
ⓐ 서브 클래스들의 공통된 특성을 하나의 슈퍼 클래스로 추출하기 위한 목적으로 생성된 클래스로 재사용 부품을 이용하여 확장할 수 있는 개념이다.
ⓑ 일반 클래스와는 달리 객체를 생성할 목적을 가지고 있지 않으며 또한 생성할 수도 없다. 점진적 개발이 용이하다.

④ 메시지
㉠ 한 객체가 다른 객체의 모듈을 부르는 과정으로, 외부에서 하나의 객체에 보내지는 행위의 요구이다.
㉡ 인터페이스를 통해 전달되며 객체상에 수행되어야 할 연산을 기술한다.
㉢ 일반 프로그래밍 과정에서 함수 호출에 해당된다.
㉣ 메시지의 구성요소 : 메시지를 받는 객체(수신개체), 객체가 수행할 메서드 이름(함수이름), 메서드를 수행하는데 필요한 인자(매개변수)

⑤ 메소드
㉠ 연산은 객체가 어떻게 동작하는지를 규정하고 속성의 값을 변경시킨다.
㉡ 연산은 메시지에 의해 불리어질 수 있는 제어와 절차적 구성요소이다.

⑥ 다형성(polymorphism)
㉠ 같은 메시지에 대해 각 클래스가 가지고 있는 고유한 방법으로 응답할 수 있는 능력을 의미한다.
㉡ 두 개 이상의 클래스에서 똑같은 메시지에 대해 객체가 서로 다르게 반응하는 것이다.
㉢ 다형성은 주로 동적 바인딩에 의해 실현된다.
㉣ 각 객체가 갖는 메소드의 이름은 중복될 수 있으며, 실제 메소드 호출은 덧붙여 넘겨지는 인자에 의해 구별된다.

⑦ 상속성
㉠ 새로운 클래스를 정의할 때 처음부터 모든 것을 다 정의하지 않고 기존의 클래스들의 속성을 상속받고 추가로 필요한 속성만 추가하는 방법이다.
㉡ 높은 수준의 개념은 낮은 수준의 개념으로 특정화된다.
㉢ 상속은 하위 계층은 상위 계층의 특수화(specialization) 계층이 되며, 상위 계층은 하위 계층의 일반화(generalization) 계층이 된다.
㉣ 객체지향 프로그래밍 언어에서 상속이란 개념은 간단히 클래스를 더 구체적인 클래스로 발전시킬 수 있는 도구이다.
㉤ 클래스 계층은 요구된 속성들과 연산들이 새로운 클래스에 의해 상속받을 수 있게 재구성될 수 있으며, 새로운 클래스는 상위 클래스로부터 상속받고 필요한 것들이 추가될 수 있다.
㉥ 상속을 받은 하위 클래스는 상위 클래스의 속성과 메소드를 자기의 특성에 맞게 수정하거나 확장할 수 있다는 overriding과 연관된다.

⑧ 캡슐화
㉠ 객체를 정의할 때 서로 관련성이 많은 데이터들과 이와 연관된 함수들을 정보처리에 필요한 기능을 하나로 묶는 것을 말한다.
㉡ 객체의 내부적인 사항과 객체들 간의 외부적인 사항들을 분리시킨다.
㉢ 사용자에게 세부 구현사항을 감추고 필요한 사항들만 보이게 하는 방법으로, 객체의 사용자로 하여금 내부 구현 사항으로 접근을 방지한다.
㉣ 데이터구조와 이들을 조작하는 동작들은 하나의 개체인 클래스에 통합되므로, 컴포넌트 재사용을 용이하게 해준다.
㉤ 내부에 변수, 외부에는 메소드들이 변수들을 둘러쌓아 보호한다. 즉 데이터와 절차의 세부적인 구현사항은 외부세계로부터 감추어져 있어 변경연산 시 부작용을 감소시켜준다.

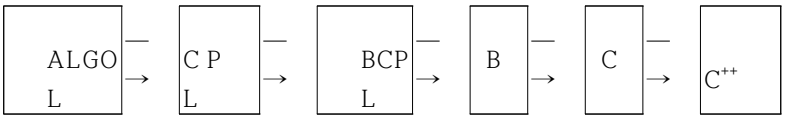
⑨ 정보은닉(Information Hiding) : 객체의 상세한 내용을 객체 외부에 철저히 숨기고 단순히 메시지만으로 객체와의 상호작용을 하게 하는 것. 외부에서 알아야 하는 부분만 공개하고 그렇지 않은 부분은 숨김으로써 대상을 단순화시키는 효과가 있다. 정보은닉은 높은 독립성, 유지보수성, 향상된 이식성을 제공한다.

제2절 C언어

1. C 언어의 개요

(1) C 언어의 역사

① C 언어는 Bell 연구소에서 UNIX라는 운영체제에 사용하기 위한 시스템 프로그래밍 언어로 70년대 초 Dennis Ritchie에 의해 개발되었다.
② C언어의 뿌리는 최초의 구조적 언어인 ALGOL언어이며 Dennis Ritchie는 동료인 켄 톰슨(Ken Thompson)이 만든 B언어를 개량하여 C언어를 만들었다.
③ UNIX는 처음에 어셈블리어로 작성되었으나 곧 C 언어로 다시 작성되었고, UNIX가 세상에 알려지면서 어셈블리어가 아닌 언어로 능력있고 훌륭한 운영체제가 개발될 수 있다는 사실을 입증하는 결과를 가져왔다.



(2) C 언어의 특징

① C 프로그램은 함수의 집합으로 구성된다.
- 각 루틴의 특성에 맞추어 각각의 함수를 만들어 두면 다른 응용프로그램을 작성할 때도 그대로 이용할 수 있다.
② 이식성(Portable)이 높은 언어이다.
- C로 작성된 프로그램은 거의 수정없이 다른 컴퓨터 시스템에서 컴파일 되고 실행된다.
③ 예약어(Reserved Word)가 간편하다.
- 기본적인 몇가지의 예약어로 다양한 종류의 작업을 처리할 수 있는 프로그램을 개발할 수 있다.
④ 융통성과 강력한 기능을 갖고 있다.
- 과학 기술 및 업무용 프로그램뿐만 아니라 오락, 문서작성기, 데이터베이스 등을 만드는데 사용될 수 있고, 심지어는 운영체제와 또 다른 언어의 컴파일러를 개발하는데 사용될 수 있는 언어이다.
⑤ 구조적 프로그램이 가능하다.
- 아무리 복잡한 논리구조도 『goto』 명령을 사용하지 않고 처리할 수 있다.

(3) 기본 구조

① 헤드 부분
㉠ 외부파일 편입(#include문)
㉡ 매크로 정의(#define문)
㉢ 전역변수 및 사용자 정의함수 선언

② 몸체 부분
㉠ 함수 main()은 C프로그램에서 예약된 유일한 함수로 프로그램 실행시 가장 먼저 수행되는 함수 (모든 프로그램은 main 함수부터 실행 시작)
㉡ main() 함수의 위치는 프로그램 내의 어디에나 위치할 수 있고 반드시 한 번 기술되어야 함

③ 사용자 정의 함수
㉠ 처리할 내용에 맞게 함수를 정의하고 경우에 따라서는 또 다른 함수를 호출할 수 있다.
㉡ 실제 프로그램에서는 사용자 정의 함수가 여러 개 나열되어 완전한 프로그램이된다.
㉢ 함수 내부에서는 또 다른 함수를 정의할 수 없다.

(4) 전처리문(Preprocessor Statement)
① 컴파일하기 전에 원시프로그램에 사용된 전처리문을 전처리기(Preprocessor)가 확장시킨다.
② 전처리문의 종류

전처리문	기 능
#include	외부파일을 원시프로그램에 편입
#define	매크로 정의 (함수 및 상수)
#undef	정의된 매크로를 취소
#if-#endif	조건에 따른 컴파일

```
#define VAT 0.2
main(){
  int a;
  a = VAT * 100;
  printf("%d", a);
}
```

[원시 프로그램]

```
main() {
  int a;

  a = 0.2 * 100;
  printf("%d",a);
}
```

[확장된 원시프로그램]

2. C언어의 구성 요소

(1) 예약어(Reserved Word)

구 분	종 류
자 료 형	char, int, float, double, enum, void, struct, union, short, long, signed, unsigned 등
기억분류	auto, register, static, extern
제어구조	if~else, for, while, do~while, switch~case~default, break, continue, return, goto
연 산 자	sizeof

(2) 명칭(Identifier)

① 예약어만을 명칭으로 사용할 수 없다.
② 영문자, 숫자, 밑줄(_)을 사용하여 명칭을 구성할 수 있다.
③ 숫자로 시작해서는 안된다.
④ 대문자와 소문자는 구별된다.

(3) 자료표현

구분	종 류		표 현 방 법	예
숫자	정수	10진수	일반적인 정수형 표현	25, -356
		8 진수	숫자 앞에 0을 붙인다.	075, 0653, 0111
		16진수	숫자 앞에 0X를 붙인다.	OX41, OXFF
	실 수		소수점이 있거나 숫자 끝에 f를 기술한다	3.1415, 6.3, 74f
문자	문 자		단일 따옴표로 묶는다.	'A', 'K'
	문 자 열		이중 따옴표로 묶는다.	“KOREA”, “B”

(4) Escape sequence

Escape sequence는 문자를 표현하는 한 가지 방법으로 역 슬래쉬(₩) 다음에 특정 기호를 기술하여 하나의 문자를 표현하는 것이다.

종 류	의 미	비 고
'₩n'	커서를 다음 행으로 이동	New line
'₩r'	커서를 현재 행의 첫 번째로 이동	Return
'₩t'	커서를 다음 탭 위치로 이동	Tab
'₩b'	커서를 앞으로 한칸 이동	Back space
'₩f'	인쇄용지를 1쪽 이동(Page skip)	Form feed
'₩a'	소리 발생('삑'하고 경고음이 발생)	Beep
'₩''	단일 따옴표(')를 지칭	
'₩"'	이중 따옴표(")를 지칭	
'₩₩'	역슬래쉬(₩)를 지칭	
'₩수'	수에 해당하는 ASCII문자(수는 8진수로 인식됨)	
'₩x수'	수에 해당하는 ASCII문자(수는 16진수로 인식됨)	

(5) 자료형(Data Type)

구 분	자 료 형	비트수	허용 범위
문자형	char unsigned char	8 8	-128 ~ 127 0 ~ 255
정수형	short int unsigned int long unsigned long	16 가변적 가변적 32 32	-2^{15} ~ 2^{15} - 1 -2^{31} ~ 2^{31} - 1 0 ~ 2^{32} - 1
실수형	float double	32 64	
열거형	enum	16	-2^{15} ~ 2^{15} - 1
void형	void	함수와 포인터에서 이용	

3. C 언어의 연산자(Operator)

(1) 연산자

① 산술연산자(Arithmetic Operator)

구 분	연산자	기 능
이항연산자	+, -, *, / %	사칙연산을 수행 정수연산으로 나눗셈의 나머지를 구함
단항연산자	- ++ --	대상 자료의 부호를 바꾼다. 1 증가시킨다. 1 감소시킨다.
대입연산자	= += -= *= /= %=	오른쪽의 결과를 왼쪽 변수에 대입 오른쪽의 결과를 왼쪽 변수에 가, 감, 승, 제를 한다. 오른쪽으로 왼쪽을 나눈 나머지를 구함

ex1)

```
#include <stdio.h>
int main(void)
{     int i=100, j=200;
      printf("i : %d , j : %d \n", ++i, j++);
      printf("i : %d , j : %d \n", i, j);
      printf("i : %d , j : %d \n", --i, j--);
      printf("i : %d , j : %d \n", i, j);

      i=j++;
      printf("i : %d , j : %d \n", i, j);
        return 0;
}
```

<실행결과>

```
i : 101 , j : 200
i : 101 , j : 201
i : 100 , j : 201
i : 100 , j : 200
i : 200 , j : 201
```


② 관계 및 논리연산자

구 분	연산자	기 능
관계연산자	== != >, >=, <, <=	좌우가 같은가를 비교한다. 좌우가 다른가를 비교한다. 좌우의 대소 관계를 비교한다.
논리연산자	! && ||	NOT 연산을 수행(부정) AND 연산을 수행(논리곱) OR 연산을 수행(논리합)

③ 비트연산자(Bitwise Operator)

구 분	연산자	기 능	예
이동연산자	>> <<	비트 값을 우측으로 이동 비트 값을 좌측으로 이동	r = a >> 3; r = a << 3;

비트연산자	&
	|
	^
	~
		비트 논리곱(AND)
		비트 논리합(OR)
		비트 배타적 논리합(XOR)
		반전 (NOT, 1의 보수)
			r = a & b;
			r = a|b;
			r = a^b;
			r = ~a;

ex1)

```
char c = 3;        /*  c = 00000011  */
c << 1                    = 00000110          = 6
c << 2                    = 00001100          = 12
c << 3                    = 00011000          = 24
c << 4                    = 00110000          = 48
```

ex2)

```
char c = 48;       /*  c = 00110000  */
c >> 1                    = 00011000          = 24
c >> 2                    = 00001100          = 12
c >> 3                    = 00000110          = 6
c >> 4                    = 00000011          = 3
```

④ 조건연산자

[형식] 조건 ? 표현1 : 표현2 ;
→ 조건이 참이면 표현1을 수행, 거짓이면 표현2가 수행된다.

ex) ① a = 12; b = 7 ;
d = (a>b) ? a+b : a-b;
d = 19
② a = 8; n = 10;
y = (a>9) ? n++ : n--;
y = 10

4. 제어 구조

(1) if ~ else : 선택문

(2) switch~case

```
[형식]      switch(수식)
        {
            case 값1  :  처리 1
                         break ;
            case 값2  :  처리 2
                         break ;
               ⋮
            default   :  처리 n
        }
```

① 수식의 값과 일치하는 경우의 값이 있는『처리』를 수행하고, 수식의 값과 일치하는 값이 없으면 default의『처리 n』을 수행한다.
② break를 만나면 switch 블록을 탈출한다. 즉, break가 없으면 수식의 값에 해당하는 경우부터 break가 있는 곳까지 수행한 후 switch 블록을 탈출한다.

ex1)

```
void  main( ) {
            int  a, n = 10;
                 a = 3;
            switch ( a ) {
                 case 1 : n = n + 1;
                          break;
                 case 2 : n = n + 2;
                          break;
                 case 3 : n = n + 3
                          break;
                 default: n = n + 4;
                          break;
            }
            printf("실행 결과 : %d₩n", n);
      }
```

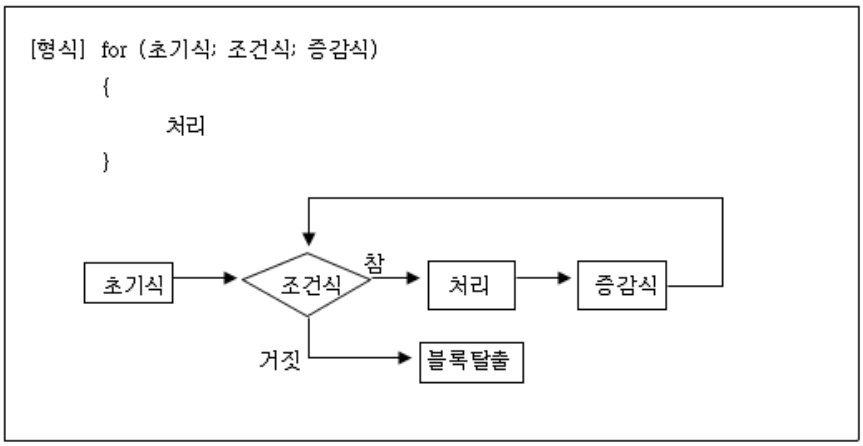
<실행 결과> 실행 결과 : 13

ex2)

```
int c=0;
switch(2) {
     case 1: c=c+1;
     case 2: c=c+2;
     case 3: c=c+3;
     case 4: c=c+4;
}
```

<실행 결과> 9

(3) for문



① 조건식이 거짓이면 블록을 탈출한다.
② for 블록의 실행순서는 다음과 같이 반복된다.

③ 중첩된 for문

```
n = 0 ;
for(i = 0 ;  i < 2 ;  i++)
{
  for( j = 0 ;  j < 3 ;  j++)
      {
          n++;
      }
}
```

ex1)

```
main( ){
    int i;
    for (i=0; i<=10; i+=2){
        printf("%d",i);
    }
}
```

<실행 결과> 0 2 4 6 8 10

ex2)

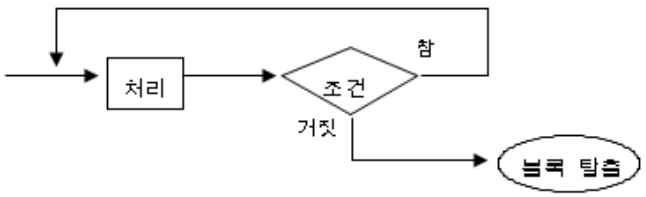
```
#include <stdio.h>
    void main(void){
        static int a[5]={10, 20, 30};
        int i, h=0;

        for (i=1; i<5; i++)
              h+=a[i];
        printf("%d /n", h);
    }
```

<실행 결과> 50

(4) do~while문

[형식] do {
처리
}while(조건식) ;



ex1)

```
void main()
  {
        int   i=0, hap=0;

        do {
              i++;
              hap += i;
        } while(i<100);

        printf("실행결과 : %d₩n", hap);
  }
```

<실행 결과> 5050

ex2)

```
int sum = 0, i = 0;
do{
    sum += i++;
}while(i<=20);
```

(5) break / continue문

```
while(조건식)
 {
        ⋮
     continue ;
        ⋮
     break ;
        ⋮
 }
```

① continue : for, while, do~while에서 블록의 조건식으로 복귀하고자 할 때 사용한다.
② break : for, while, do while, switch에 블록을 강제적으로 벗어나고자 할 때 사용한다.

ex)

```
#include <stdio.h>
void main() {
  int i, j, sum = 0;
  for (i = 1; i < 10; i++) {
    for (j = 1; j < 10; j++) {
      if (j%3 == 0) continue;
      if (i%4 == 0) break;
       sum++;
    }
  }
printf("%d\n", sum);
}
```

<실행 결과> 42

5. 함수(Function)

(1) 표준함수

C 컴파일러에 구비되어 있는 함수로 사용자가 정의하지 않고 사용할 수 있다.

① 단일문자 입출력함수(getchar() / putchar())
② 문자열 입출력 함수(gets() / puts())
③ 형식지정 입출력함수(scanf() / printf()) : 형식지정은 % 기호 다음에 여러 가지『변환문자』를 사용하여 지정한다. 즉, % 다음에 있는 변환문자에 따라 해당하는 인수의 입출력 기능이 달라진다.

변환기호	기 능
%c	인수를 단일문자로 변환시킨다.
%d	인수를 부호 있는 10진수로 변환시킨다.
%u	인수를 부호 없는 10진수로 변환시킨다.
%o	인수를 8진수로 변환시킨다.
%x	인수를 16진수로 변환시킨다.
%s	인수를 포인터형으로 변환시킨다(문자열 입출력).
%f	인수를 실수형으로 변환시킨다.

㉠ scanf()

- 표준입출력장치로 부터 지정된 형식에 맞게 자료를 읽어 들인다.

[형식] scanf("형식", 인수리스트);

▸형식에 사용된 변환기호(%c, %d 등)가 입력받을 자료형이 된다.
▸scanf()는 변수의 주소를 인수로 사용한다. 따라서 일반변수 앞에는 &를 붙이고, 배열명이나 포인터에는 &를 붙이지 않는다.

㉡ printf()

- 표준출력장치에 지정된 형식에 맞추어 자료를 출력한다.

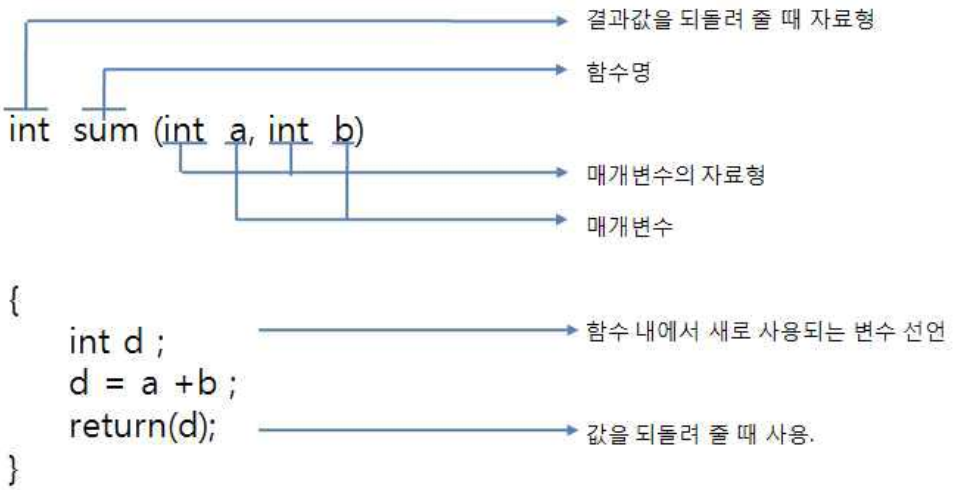
[형식] printf("형식", 인수리스트);

▸printf() 함수는『변수, 상수, 수식 등의 값을 인수』로 사용한다.

(2) 사용자 정의함수

사용자가 프로그램에서 직접 정의하여 사용하는 함수로 앞에서 설명한 표준함수와 동등하게 취급된다.

① 함수의 정의



② 함수의 사용

㉠ 함수를 사용할때는 함수의 원형선언, 함수의 호출, 함수의 정의로 구성되어야 함

㉡ 함수의 구성위치의 예

```
#include <stdio.h>
int sum(int a, int b);        →함수의 원형 선언.
```

```
void main() {
int x, y, c;
scanf("%d %d", &x, &y);
.....
c=sum(10, 20);                →함수의 호출.
printf("%d', c);
}
```

```
int sum(int a, int b) {       →함수의 정의.
int d;
d=a+b;
return(d);
}
```

6. 배열과 포인터

(1) 배열(Array)

① 변수의 확장에 해당하는 것으로 유사한 성격, 즉 동일한 자료형으로 이루어진 여러 개의 자료를 처리할 때 사용.

② 변수명을 모두 기억해야 하는 번거로움을 피할 수 있으며 매우 효율적인 자료처리가 가능하게 된다.

③ 1차원 배열 : 배열의 첨자가 하나만 있는 것으로 첨자 안에 표현된 개수는 배열의 크기를 나타내는 것으로서 배열 전체 구성요소의 개수를 나타냄

㉠ 배열선언

형식: 자료형 배열명[개수]

ex) 배열의 초기화

```
int  a[5] = { 1, 2, 3, 4, 5 } ;
```

1	2	3	4	5
a[0]	a[1]	a[2]	a[3]	a[4]

② 2차원 배열

㉠ 형식 : 자료형 배열명[행의수][열의수] (ex: int a[3][4])

㉡ 기능 : 배열명이 a이고 3행 4열로 된 12개의 요소를 가진 정수형 배열.

㉢ 첨자가 두개인 배열로 배열요소의 개수는 3*4는 12개

㉣ 배열선언과 초기화방법

```
ex1) int array[3][3]={1,2,3,4,5,6,7,8,9};
ex2) int array[3][3]={{1,2,3},{4,5,6},{7,8,9}};
ex3) int array[3][3]={{1,2,3}.
                      {4,5,6},
                      {7,8,9}};
```

(2) 포인터(Pointer)

포인터는 한마디로 주소(번지 ; Address)를 일컫는다. 기억공간의 주소값을 갖는 변수를 포인터 변수 또는 포인터라고 하며 *를 사용하여 포인터를 선언한다.

```
char  * ptr;
```

ptr : 기억공간의 주소값을 갖는다.(char형 포인터 값)

*ptr : 포인터 ptr이 가르키는 주소에 수록된 자료. 즉, 주소 ptr의 내용이다.
(포인터 앞에 *를 붙이면 내용물이 된다)

※ 포인터는 C 언어에서 제공되는 자료형에 모두 적용될 수 있다.

```
char  *p;
int   *q;
long  *r
float *f; 등
```

이들 포인터 p, q, r, f의 공통점은 주소값을 기억하고 주소를 직접 다루는 것이며, 가장 큰 차이점은 포인터가 가르키는 주소에서 시작하여 선언된 자료형이 갖는 바이트 수만큼씩 자료를 취급하는 것이다.

④ 포인터와 주소연산자(&) : 일반 변수가 위치하는 메모리의 주소를 구하기 위해서는 주소연산자 &를 사용한다. 즉 a라는 변수의 시작주소는 &a이다.

ex) 포인터와 주소연산자 사용

```
void        main(){
            int  a, b ;
            int  *p;

            a = 7 ;
            p = &a;
            b = *p;

    printf("%d\n", b);
}
```

〈실행 결과〉 7

7. 기억부류(Storage Class)

기억부류지정자에는『auto, register, static, extern』이 있는데, 이들은 변수를 메모리의 어느 영역에 지정할 것인지를 결정하게 된다.

(1) auto(자동변수)

① 자동변수는 함수 내부에서 선언하는 것으로 변수 앞에 기억부류지정자를 생략하면 자동변수로 간주된다.

② 기억장소 : Stack 영역

③ 함수가 실행될 때 생성되고, 함수가 종료되면 자동 소멸된다. 따라서 선언된 함수 내부에서만 사용할 수 있고, 메모리 절약 효과를 가져온다(지역변수)

(2) register(레지스터변수)

① 사용하지 않는 CPU의 레지스터를 변수의 기억장소로 사용하며, 고속처리에 이용된다. 특징은 자동변수와 같으나 주소참조 등은 불가능하다.

② 레지스터변수를 사용하는 이유는 프로그램의 실행속도를 조금이나마 늘리기 위함으로, 기억장치로의 자료 입출력보다 레지스터의 자료 입출력이 속도가 빠르기 때문에 반복문에서의 카운터 변수로 많이 사용되며 register변수로 선언됨.

(3) static(정적변수)

변수 앞에『static』을 기술하면 정의된 변수는 메모리상의『정적영역』에 위치하여 프로그램 종료시까지 변수의 값이 유지된다.

① 내부정적변수 : 함수 내부에서 정의한 변수로 통용 범위는 정의한 함수 내부이다.

② 외부정적변수 : 함수 외부에서 정의한 변수로 통용 범위는 자신을 정의한 모듈이다. 여기서 모듈이란 파일 단위의 원시프로그램을 뜻한다.

(4) extern

다른 모듈에 정의된 외부변수를 참조하려면 변수 앞에『extern』을 기술해야 한다. 즉, 외부변수는 정의된 모듈에만 일단 통용되기 때문이다.

ex1) 기억부류변수 선언

```
  static int sh ; ⟶ 외부정적변수 : extern문으로 다른 모듈에는 알릴 수 없고   자신이 정의된 모듈의 모든 함수에 사용할 수 있다.

  char ar[5] ;    ⟶ 외부변수 : extern문을 이용하여 다른 모듈에서도 사용할 수 있다.

  void main()
  {
      auto int i, j, h ; ⟶ 자동변수
      static int n[8] ; ⟶ 내부정적변수
      h = foo(5) ;
          printf("%d\n", h);
              ⋮
  }

  char foo(int k)
  {
      register int r ; ⟶ 레지스터 변수
                char  i, j, h ;  ⟶ 자동변수 ( 함수 main()의 i, j, h와는 무관)
                   ⋮
  }
```


제3절 Java 언어

1. 자바언어의 기본

(1) 자바의 개요

1) 자바의 유래

① 자바는 1990년대 초 미국 Sun Micro 사의 James Gosling이 가전 제품에 이용할 목적으로 파스칼을 모델로 개발하였으나 별다른 반응을 얻지 못했다. 인터넷이 급속도로 확산된 90년대 중반에야 가서 관심의 초점이 되었다.

② 자바는 오크(Oak)라는 언어로 부터 탄생되었다. 오크는 1991년 미국의 선 마이크로시스템즈사의 제임스 고슬링(James Gosling)이 가전 제품의 기능을 프로그램으로 제공하기 위해 개발하였다.

2) 자바의 특징

① Simple : 단순하다.
③ Distributed : 분산환경에 적합하다.
④ Interpreted : 인터프리터에 의해 실행된다. (하이브리드 방식)
⑤ Robust : 견고하다.
⑥ Secure : 안전하다.
⑦ Architecture neutral : 구조 중립적이다.
⑧ Portable : 이식성이 높다.
⑨ High-performance : 높은 성능을 가진다.
⑩ Multithreaded : 다중스레드를 제공한다.
⑪ Dynamic : 동적이다.

(2) 자바 언어의 기본구조

1) 자바 애플리케이션 분석

① 클래스
ⓐ 객체지향 프로그래밍에서 가장 기본이 되는 Class를 정의하는 키워드이다.
ⓑ 클래스의 이름은 관례적으로 첫 글자를 대문자로 쓴다.
ⓒ 보통의 경우 main()메서드가 포함된 클래스 이름이 프로그램의 이름이 된다.
ⓓ 클래스의 몸체는 { 과 } 로 나타내고, 그 안에 데이터와 메서드를 기술한다.

② main() 메서드
ⓐ 자바 애플리케이션에서 반드시 있어야 하는 특수 메서드이다.
ⓑ 실행시 자동으로 실행되는 유일한 메서드이다.
ⓒ 일반적으로 자바 애플리케이션은 main() 메서드 내에서 다른 클래스의 객체를 생성하고, 그 객체에 메시지를 보내어 원하는 결과를 얻는다.

③ 표준 입출력

ⓐ 자바에서의 표준 출력 : System.out
(표준 출력 메소드 : println(), print())

ⓑ 자바에서의 표준 입력 : System.in
(입력 메소드 : read())

2) 자바의 기본구조

① 키워드(예약어)

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

※ 예약어 중에서 const와 goto는 예약되어 있으나 사용할 수는 없다.

② 명칭(식별자;Identifier)

자바에서 식별자는 상수, 변수, 배열, 문자열, 사용자 정의 클래스나 메소드 등의 이름이 된다.

③ 자료형

ⓐ 기본형(Primitive Type) : 변수 자체가 값을 가지는 데이터형

구분	자료형	크기(byte)	범위	설명
정수형	byte	1	-128~127	부호있는 정수
	short	2	-32768~32767	
	int	4	-2^31~2^31-1	
	long	8	-2^63~2^63-1	
	char	2	/u0000~/uFFFF	유니코드 1문자
실수형	float	4	-3.40292347E38~+3.40292347E38	부동소수점수
	double	8	-1.79769313486231570E308 ~ +1.79769313486231570E308	
논리형	boolean	1	true/false	참/거짓

ⓑ 참조형(Reference Type)

- 참조하는 객체의 주소를 값으로 가진다. C의 포인터와 유사한 것으로 자바에서는 모든 객체를 참조형으로 취급한다.
- String : 문자열을 저장하는 클래스
- Array : 배열
- 기타 각종 클래스

④ 배열

자바에서는 배열을 객체로 취급하므로 배열을 사용하기 위해서는 배열 객체를 선언하고, 객체를 생성하여야 한다.

ⓐ 배열 선언

자료형 배열명[] 또는 자료형[] 배열명;

ⓑ 배열 객체 생성

배열명[] = new 자료형[크기]; → 배열 선언 후 객체를 생성하는 경우
자료형 배열명[] = new 자료형[크기]; → 배열 선언과 동시에 객체 생성

ex) 배열 선언 및 객체 생성 예

```
int a[];  //배열 선언
loolean b[] = null; //배열 선언과 초기화.
long c[] = new long[20]; //배열 선언과 동시에 객체 생성
String[] d = new String[30]; //배열 선언과 동시에 객체 생성. String에서
                                S는 반드시 대문자로 해야함.
```

ⓒ 배열 선언과 초기화

배열 선언과 동시에 초기화를 하면 배열 객체가 생성되어 진다.

int k[] = {1, 2, 3, 4, 5}; → 첨자는 0에서 4까지 사용 가능. 배열 크기는 기술하지 못한다.

⑤ 이차원 배열

ⓐ 배열 객체 생성 후 배열 요소 값을 지정

```
int arr[][] = new int[2][2];

arr[0][0] = 100;
arr[0][1] = 200;
arr[1][0] = 300;
arr[1][1] = 400;
```

	0	1
0	100	200
1	300	400

ⓑ 배열 선언과 동시에 초기화

```
int arr[][] = { {100, 200, 300}, {400} };
```

→ 열 크기가 다른 2차원 배열 구조가 된다.

	0	1	2
0	100	200	300
1	400		

⑥ 문자열(String)

ⓐ 자바는 문자열을 다루기 위해 클래스 String과 StringBuffer를 제공한다. 문자열은 객체로 취급된다.

ⓑ String : 생성된 객체의 내용을 변경할 수 없음.(상수 문자열)

ⓒ StringBuffer : 생성된 객체의 내용을 변경할 수 있음.

ⓓ String클래스와 StringBuffer클래스는 java.lang패키지에 포함되어 있음.

2. 자바언어의 구성요소

(1) 클래스의 구조

[형식]

```
[ 접근자 | 옵션 ] class 클래스이름 [ extends Superclassname ]
            [ implements Interface (, Interface) ] {
          클래스 정의 부분 (변수와 메소드 정의)
 }
```

※ { 와 } 사이에 멤버변수, 생성자 메소드 및 메소드를 기술

1) 접근자(Modifiers)와 옵션(Option)

① default(공백) 또는 package - 패키지 내부에서만 상속과 참조 가능
② public - 패키지 내부 및 외부에서 상속과 참조 가능
③ protected - 패키지 내부에서는 상속과 참조 가능, 외부에서는 상속만 가능
④ private - 같은 클래스 내에서 상속과 참조 가능
⑤ abstract - 객체를 생성할 수 없는 클래스
⑥ final - 서브 클래스를 가질 수 없는 클래스
⑦ static - 멤버 클래스 선언에 사용

2) 객체의 선언과 생성

```
클래스이름 객체이름 = new 생성자메소드;
```

① 작성한 클래스의 멤버 변수를 할당받고, 메소드를 실행하기 위해서는 클래스로부터 객체를 생성해야 한다.
② 속성의 접근 : 객체명.속성변수명
③ 메소드 호출 : 객체명.메소드명(매개변수)

ex)

```
class MethodEx {
        int var1,var2;
        public int sum(int a, int b){
                return a+b;
        }
        public static void main(String[] args){
                MethodEx me = new MethodEx();
                int res = me.sum(1000, -10);
                System.out.println("res="+res);
        }
}
```

(2) 멤버변수(Member Variable)

1) 객체의 속성을 정의하는 것으로 클래스의 메소드 밖에서 선언된 변수이다.
2) 멤버변수의 분류
① 객체 변수
㉠ 객체속성변수 : 기본 자료형의 값을 가지는 변수이다.
㉡ 객체참조변수 : 객체를 지정하는 변수이다.(주소를 가진다)
㉢ 객체속성변수와 객체참조변수는 객체참조변수(객체명)로 접근해야 한다.

② 클래스 변수
㉠ static으로 선언된 변수(전역변수의 개념)로 그 클래스로 부터 생성된 객체들이 공유한다.
㉡ 객체들 사이에 공통되는 속성을 표현하는데 사용될 수 있다.
㉢ 클래스 변수는 클래스명으로 접근한다.

③ 종단 변수
㉠ final로 선언된 변수로 상수 값을 가진다. 한번 초기화할 수 있으며, 그 후로는 새로운 값을 대입할 수 없다.
㉡ 프로그램에서 변하지 않는 상수 값을 선언할 때 사용한다.
㉢ 관례적으로 종단변수는 대문자로 한다.

ex) final int AAA = 250;

ex)

```
class Gogaek{
  String irum;
  int nai;
  long bunho;
  static long GoBunho=0;
  public Gogaek(){
       bunho = GoBunho++;
  }
}

class VarDemo{
  public static void main(String args[]){
    Gogaek gogaek1=new Gogaek();
    Gogaek gogaek2=new Gogaek();
    Gogaek gogaek3=new Gogaek();

      gogaek1.irum = "컴퓨터";

    System.out.println("고객1 이름 : "+gogaek1.irum);
    System.out.println("gogaek1의 id의 번호:"+ gogaek1.bunho);
    System.out.println("gogaek2의 id의 번호:"+ gogaek2.bunho);
    System.out.println("gogaek3의 id의 번호:"+ gogaek3.bunho);
    System.out.println("전체 고객 수 : "+Gogaek.GoBunho+"명");
  }
}
```

<실행 결과>

고객1 이름 : 컴퓨터
gogaek1의 id의 번호:0
gogaek2의 id의 번호:1
gogaek3의 id의 번호:2
전체 고객 수 : 3명

(3) 메소드(Method)

① 객체 메소드

㉠ static 선택항목을 갖지 않는 메소드로 객체를 통하여 접근한다.

㉡ 객체변수는 클래스로부터 생성된 객체에 별도로 할당되나, 객체 메소드는 프로그램 코드로서 다수의 객체가 접근하여 사용할 수 있다.

```
[public|private|protected][static|final|abstract] 반환값유형 메소드이름(매개변수)
        {
            정의할 메소드 내용을 기술
        }
```

[접근방법] 객체이름.메소드이름(매개변수)

② 클래스 메소드

㉠ static으로 선언한다.

㉡ 클래스 변수와 같이 클래스 이름을 통하여 접근한다.

㉢ 클래스 메소드 내에서는 클래스 변수만을 사용할 수 있다.

```
[public|private|protected] static [static|final|abstract|synchronized] 반환값유형 메소드이름(매개변수)
        {
            정의할 메소드 내용을 기술
        }
```

[접근방법] 클래스이름.클래스메소드이름(매개변수)

③ 종단 메소드

㉠ final로 선언한다.

㉡ 종단 메소드를 포함하는 클래스로부터 하위 클래스를 생성할 때, 하위 클래스에서 종단 메소드를 재정의(overriding)하여 사용할 수 없다.

④ 추상 메소드

㉠ abstract로 선언하며, public만을 사용할 수 있다.

㉡ 하나 이상의 추상 메소드를 포함한 클래스를 추상클래스라 한다.

㉢ 추상 메소드는 메소드의 실행문을 갖지 않으므로, 반드시 하위 클래스에서 재정의하여 사용해야 한다.

[구문]

```
public abstract 반환값유형 메소드이름(매개변수);
```


※ 메소드 중복(Overloading)

① 하나의 클래스에 이름은 같으나 매개변수의 자료형과 개수가 서로 다른 다수의 메소드를 사용하는 것이다.

② 중복된 메소드가 호출되면 매개변수의 형과 개수를 비교하여 적합한 메소드가 실행된다.

ex)

```
class Over{
        String foo(){
                return "인수가 없음";
        }
        int foo(int a){
                return a * a;
        }
        int foo(int a, int b){
                return a * b;
        }
        int foo(int a, int b, int c){
                return a * b * c;
        }
}
class Overlo{
    public static void main(String args[]){
                Over g = new Over();
         System.out.println(g.foo());
         System.out.println(g.foo(5));
         System.out.println(g.foo(4, 5));
         System.out.println(g.foo(2, 3, 4));
    }
}
```

<실행 결과>

인수가 없음

25

20

24

(4) 생성자(Constructor)

1) 생성자의 개요

① 객체가 생성될 때 객체의 초기화 과정을 기술하는 특수한 메소드이다.

② 생성자는 일반 메소드와 같이 명시적으로 호출되지 않고, 객체를 생성할 때 new 연산자에 의하여 자동으로 실행된다.

③ 반환하는 자료형이 없고, 이름은 반드시 클래스 이름과 동일해야 한다.

④ 매개변수 및 수행문을 포함할 수 있다.

2) 생성자 중복

① 하나의 클래스에 매개변수의 자료형과 개수가 서로 다른 다수의 생성자를 포함하여 다양한 객체를 생성하는 것이다.

② 생성자의 이름은 같지만 매개변수의 개수와 형을 다르게 한다.

3) this 예약어

① 생성자나 메소드의 매개변수가 멤버변수와 같은 이름을 사용하는 경우에 사용한다.

② this 예약어는 현재 사용 중인 객체 자기 자신을 의미한다.

ex)

```
class JavaTest{
    int value1;

    JavaTest(int value1){
        this.value1 = value1;
    }
}
```


(5) 상속(Inheritance)

1) 확장 클래스

① 클래스의 계층과 상속

㉠ 상위클래스나 하위클래스가 공통으로 가지는 멤버변수와 메소드들을 상위클래스에 선언하고, 하위클래스에서는 상속받아 재사용할 수 있도록 설계한다.

㉡ 자바의 최상위 클래스는 java.lang.Object 클래스로써 상속되는 상위 클래스가 지정되지 않은 경우, 묵시적으로 Object 클래스로부터 상속받는다.

㉢ 자바에서는 모든 클래스는 하나의 상위 클래스만을 가질 수 있다.

[형식]

```
class sub클래스 extends super클래스 {
      .....
      }
```

ex)

```
class SP {
   int a = 5;
}

class SB extends SP {
   int b = 10;
}
class InEx {
   public static void main (String args[]) {
     SB ob = new SB();
     System.out.println("a = " +ob.a);
     System.out.println("b = " +ob.b);
   }
}
```


2) 메소드 재정의(Overriding)

① 상위클래스에서 정의한 메소드와 이름, 매개변수의 자료형 및 개수가 같으나 수행문이 다른 메소드를 하위클래스에서 정의하는 것이다.

ex)

```
class A{
	int compute(int a, int b){
		return a + b;
	}
	public A(){
		System.out.println("최상위클래스");
	}
}
class B extends A{
	 int compute(int a, int b){
		return a * b;
	}
}
class C extends B{
	int compute(int a, int b){
		return a - b;
	}
}
class OverrideDemo{
    public static void main(String args[]){
		A ride1 = new A();
		B ride2 = new B();
		C ride3 = new C();
		System.out.println(ride1.compute(2, 3));
		System.out.println(ride2.compute(2, 3));
		System.out.println(ride3.compute(2, 3));
  }
}
```

<실행 결과>

최상위클래스
최상위클래스
최상위클래스
5
6
-1

3) 추상 클래스와 추상 메소드

① 추상 메소드와 추상 클래스는 반드시 키워드 abstract로 선언해야 한다.

② 실행문 없이 정의된 메소드를 추상 메소드라 하며, 하나 이상의 추상 메소드를 포함한 클래스를 추상 클래스라 한다.

③ 추상 메소드 : 메소드의 추상적인 기능만 선언하고 그 내용은 기술하지 않은 메소드이다.

④ 추상 클래스 : 클래스 내에 추상 메소드가 하나라도 있으면 추상 클래스이다.

⑤ 추상 클래스는 구현되지 않은 추상 메소드를 포함하므로 객체를 생성할 수 없다.

[형식]

```
abstract class 클래스이름 {
  [접근한정자] abstract 자료형 추상메소드이름();
}
```

ex)

```
abstract class Comp{
  abstract int  compute(int x, int y);
}
class Hap extends Comp{
  int compute(int a, int b){
       return a + b;
  }
}
class Gob extends Comp{
  int compute(int a, int b){
       return a * b;
  }
}
public class AbstractDemo{
   public static void main(String args[]){
     Hap hap = new Hap();
     Gob gob = new Gob();

     System.out.println(hap.compute(2, 3));
     System.out.println(gob.compute(2, 3));
   }
}
```

<실행 결과>

5

6

제4절 Python 언어

1. Python의 개요

- Python은 1991년 네덜란드의 귀도 반 로섬(Guido van Rossum)이 개발되었다.
- 범용 프로그래밍 언어로서 코드 가독성(readability)와 간결한 코딩을 강조한 언어이다.
- 인터프리터(interpreter) 언어로서, 리눅스, Mac OS X, 윈도우즈 등 다양한 시스템에 널리 사용할 수 있다.
- 웹서버, 과학연산, 사물인터넷(IoT), 인공지능, 게임 등의 프로그램 개발에 사용할 수 있다.

2. 기본 구조

(1) 연산자

+, -, *, /	사칙연산
%	나머지연산
//	나누기 연산 후에 소수점 이하 절삭
**	제곱연산

```
a = 10 % 3
print(a)
[실행결과] 1
```

```
10 / 3    -> 3.3333333333333

10 // 3    -> 3
```

(2) 문자열

① 문자열은 (' ') 와 (" ")을 활용하여 표현 가능하며, 여러 줄에 걸쳐있는 문장은 (""" """) 을 사용하여 표현한다.
② 하나의 문자열을 묶을 때 동일한 문장부호를 활용해야 한다.

aa='abcde' print(aa) abcde	aa[0] 'a'	aa[1] 'b'	aa[0:3] 'abc'

③ String method (문자열 메소드)
- capitalize() : 첫 글자는 대문자로, 나머지는 모두 소문자로 변환. ex) string.capitalize()
- title() : 각 단어의 첫 글자만 대문자로 변환. ex) string.title()
- upper() : 모두 대문자로 만들어 반환. ex)string.upper()
- lower() : 모두 소문자로 만들어 반환. ex) string.lower()

(3) list (리스트)

① 리스트는 C언어와 Java에서의 배열과 비슷한 모습을 보이지만, 리스트는 배열과 달리 정수, 실수, 문자열 등 여러 자료형을 혼합하여 저장할 수 있다.
② 리스트는 대괄호 []로 만들 수 있으며, 값에 대한 접근은 list[i] 와 같이 한다.

```
리스트명 = [value1, value2, ... ]
리스트명 = list([value1, value2, ... ])
```

ex) aa = [25, 'abc', 2.57]

(4) 숫자의 시퀀스(range)

range(n): 0<= x < n
range(n, m) : n <= x < m
range(n, m, s): n <= x < m (증가값: s)

(5) 입력/출력 함수

① input()
- 표준 입력 함수이며, 키보드로 입력받아 변수에 저장한다.

```
name = input('이름을 입력하세요 :')
이름을 입력하세요 :
--- 화면에 '이름을 입력하세요 :' 출력되고, 이름을 입력하면 name 변수에 저장된다.
```

② print()
- 표준 출력 함수이며, 화면에 결과를 출력할 때 사용한다.

print() 함수의 각 항목을 콤마(,)로 구분한다.
print(…, 변수, …., 수식, …., 값, ….)

a=10 b=20	print(a) 10	print(a+b) 30

sep을 이용한 출력

```
a1= '010'
a2= '1234'
a3= '5678'
print(a1, a2, a3, sep='-')
010-1234-5678
```

3. 제어 구조

(1) if문

① if <조건식> : 반드시 일정한 참/거짓을 판단할 수 있는 조건식과 사용

- 조건식이 참인 경우: : 이후의 문장을 수행
- 조건식이 거짓인 경우: else: 이후의 문장을 수행

- 복수 조건문: 2개 이상의 조건문을 활용할 경우 elif <조건문>: 을 활용
- 중첩 조건문: if 안에 if 실행

(2) for문

정해진 범위 내에서 순차적으로 코드를 실행한다.
범위가 이미 정해져 있기 때문에, 종료조건을 설정해주지 않아도 된다.

```
for 변수 in range(최종값):
    반복할 문장
```

ex) for i in range(5):
    sum += i;
--- 변수 i는 0에서 4까지 저장되며, 반복할 문장을 반복 수행한다.

(3) break, continue

- break : 반복문을 종료하는 표현
- continue : continue 이후의 코드를 수행하지 않고 다음 요소를 선택해 반복을 계속 수행

제5절 배치 프로그램 (Batch Program)

사용자와의 상호 작용 없이 여러 작업들을 미리 정해진 일련의 순서에 따라 일괄적으로 처리하는 것이다.

1. 배치 프로그램의 필수요소

① 대용량 데이터 : 대량의 데이터 가져오기, 전달하기, 계산하기 등의 처리 가능
② 자동화 : 심각한 오류 발생 상황을 제외하고는 사용자의 개입 없이 수행
③ 견고성 : 잘못된 데이터나 데이터 중복 등의 상황으로 중단되는 일 없이 수행
④ 안전성 / 신뢰성 : 오류 발생 시 오류의 발생 위치, 시간 등을 추적할 수 있어야 함
⑤ 성능 : 다른 응용 프로그램의 수행을 방해하지 않아야 되며, 지정된 시간 내에 처리 완료

2. 배치 프로그램 수행 주기

① 정기 배치
- 일, 주, 월과 같이 정해진 기간에 정기적으로 수행

② 이벤트성 배치
- 특정 조건을 설정해두고 조건이 충족될 때만 수행

③ On Demand 배치
- 사용자가 요청 시 수행

3. 배치 스케줄러 (Batch Scheduler)

- 일괄처리 (Batch Processing) 작업이 설정된 주기에 맞춰 자동으로 수행되도록 지원해주는 도구이다.
- 잡 스케줄러 (Job Scheduler) 라고도 불리며, 종류로는 스프링 배치, Quartz 등이 있다.

① 스프링 배치
- 스프링 프레임워크의 특성을 그대로 갖고 있으며, 데이터베이스와 파일의 데이터를 교환하는데 필요한 컴포넌트들 제공한다.

② Quartz
- 스프링 프레임워크로 개발되는 응용프로그램들의 일괄 처리를 위한 다양한 기능 제공한다.
- 수행할 작업과 수행시간을 관리하는 요소들 분리하여 일괄처리 작업에 유연성을 제공한다.

제6절 웹 저작도구

1. HTML(HyperText Markup Language)
(1) 웹 브라우저 상에 정보를 표시하기 위한 마크업 심볼 또는 파일 내에 집어넣어진 코드들의 집합이다.
(2) 홈페이지를 만들 수 있는 컴퓨터 언어
(3) 웹에서 사용하는 하이퍼텍스트 문서를 만들 수 있는 언어
(4) 기호 < >로 기능이 약속된 예약어로 이루어져 있다.
(5) 대소문자를 구분하지 않는다.

2. JavaScript
(1) 네스케이프 사에서 개발한 라이브 스크립트(Live Script)와 썬 마이크로 시스템사가 만든 자바 언어의 기능을 결합하여 만들어진 언어이며, 자바 언어에서 사용하는 문법을 따르고 있다.
(2) HTML의 텍스트 위주의 문제점을 해결하고, 동적인 데이터를 처리할 수 있다.
(3) HTML 문서 내에 자바 스크립트 코드를 그대로 삽입하며, 클래스와 상속의 개념은 지원하지 않는다.

3. ASP(Active Server Page)
(1) 서버 사이드 스크립트라는 특징이 있다.
(2) 웹브라우저에서 요청하면 웹서버에서 해석하여 응답해준다.
(3) 별도의 실행 파일을 만들 필요 없이 HTML 문서 안에 직접 포함시켜 사용한다
(4) 클라이언트에서 부가적인 작업이 존재하지 않고, 단지 HTML 문서를 받아 화면에 보여주는 작업만으로 클라이언트의 역할이 끝난다.
(5) ASP는 Windows 2000 Server, IIS, MS-SQL 과 결합되어 이용되는 것이 가장 일반적이다.
(6) 서버 입장에서는 ASP 코드를 수행한 결과 HTML 문서만 클라이언트로 전송하기 때문에 ASP 코드 및 ASP 코드로 작성된 다양한 정보가 클라이언트로 전달되지 않아서 보안성이 증대되는 효과도 있다.

4. JSP(Java Server Page)
(1) 서블릿(Servlet) 기술을 확장시켜 웹 환경에서 사용할 수 있도록 만듵 스크립트 언어이다.
(2) 웹브라우저에서 요청하면 웹서버에서 해석하여 응답해 주며, 자바의 대부분의 기능을 모두 사용할 수 있다.
(3) 별도의 실행 파일을 만들 필요 없이 HTML 문서 안에 직접 포함시켜 사용하며, 동적인 웹 문서를 빠르고 쉽게 작성할 수 있다.

5. PHP(Hypertext Preprocessor)
(1) 하이퍼텍스트 생성 언어(HTML)에 포함되어 동작하는 스크립팅 언어이며, 웹브라우저에서 요청하면 웹서버에서 해석하여 응답해 준다.
(2) 별도의 실행 파일을 만들 필요 없이 HTML 문서 안에 직접 포함시켜 사용하며, C, 자바, 펄 언어 등에서 많은 문장 형식을 준용하고 있어 동적인 웹 문서를 빠르고 쉽게 작성할 수 있다.
(3) ASP(Active Server Pages)와 같이 스크립트에 따라 내용이 다양해서 동적 HTML 처리 속도가 빠르며, PHP 스크립트가 포함된 HTML 페이지에는 .php, .php3, .phtml이 붙는 파일 이름이 부여된다.

6. Ajax(Asynchronous JavaScript and XML)
(1) 브라우저와 서버 간의 비동기 통신 채널, 자바스크립트, XML의 집합과 같은 기술들이 포함된다.
(2) 대화식 웹 애플리케이션을 개발하기 위해 사용되며, Ajax 애플리케이션은 실행을 위한 플랫폼으로 사용되는 기술들을 지원하는 웹 브라우저를 이용한다.
(3) Ajax 방식
① 웹 브라우저 ASP, PHP, JSP를 포함한 HTML 문서 요청을 하면 웹브라우저는 Javascript를 호출한다.
② Ajax 엔진은 이를 감지하여 웹서버에 HTTP 응답 요청을 보내고, 서버는 결과를 XML 형태로 만들어 Ajax 엔진에게 보내준다.
③ Ajax 엔진은 이 데이터에 HTML 형태로 사용자 화면에 출력해준다.

제11장 응용 SW 기초 기술 활용

1. 운영체제

(1) 운영체제의 정의
① 컴퓨터의 제한된 각종 자원을 효율적으로 관리, 운영함으로써 사용자에게 편리성을 제공하고자 하는 인간과 컴퓨터 사이의 인터페이스를 위한 시스템 소프트웨어이다.
② 컴퓨터 시스템의 모든 부분(소프트웨어, 하드웨어)을 제어하는 프로그램으로서 그 시스템에서 제공하는 기능을 사용할 수 있게 하는 소프트웨어이다.

(2) 운영체제의 구성
① 제어프로그램(Control Program)
- 컴퓨터 전체의 동작 상태를 감시, 제어하는 기능을 수행하는 프로그램을 말한다.
- 감시 프로그램, 데이터 관리 프로그램, 작업 관리 프로그램, 통신제어
② 처리프로그램(Processing Program)
- 제어프로그램의 감시 하에 특정 문제를 해결하기 위한 데이터 처리를 담당하는 프로그램을 말한다.
- 언어번역 프로그램, 서비스 프로그램, 사용자 프로그램

(3) 운영체제의 종류
① 윈도우(Windows) : 안정적이고 표준화된 GUI를 갖추고 있는 운영체제로 개인용 PC, 중소규모 서버, 태블릿 PC 등에서 사용된다.
② 유닉스(Unix) : AT&T에서 개발하여 멀티태스킹이 가능하고 다양한 사용자가 공유할 수 있다. 유닉스 운영체제의 경우는 컴퓨터 서버, 워크 스테이션, 휴대용 기기 등에서 사용된다.
③ 리눅스(linux) : Linus Torvals에 의해 1991년에 만들어 졌으며, 공개를 원칙으로 하기 때문에 무료 사용가능하다.
④ 모바일 운영체제 : Android, iOS

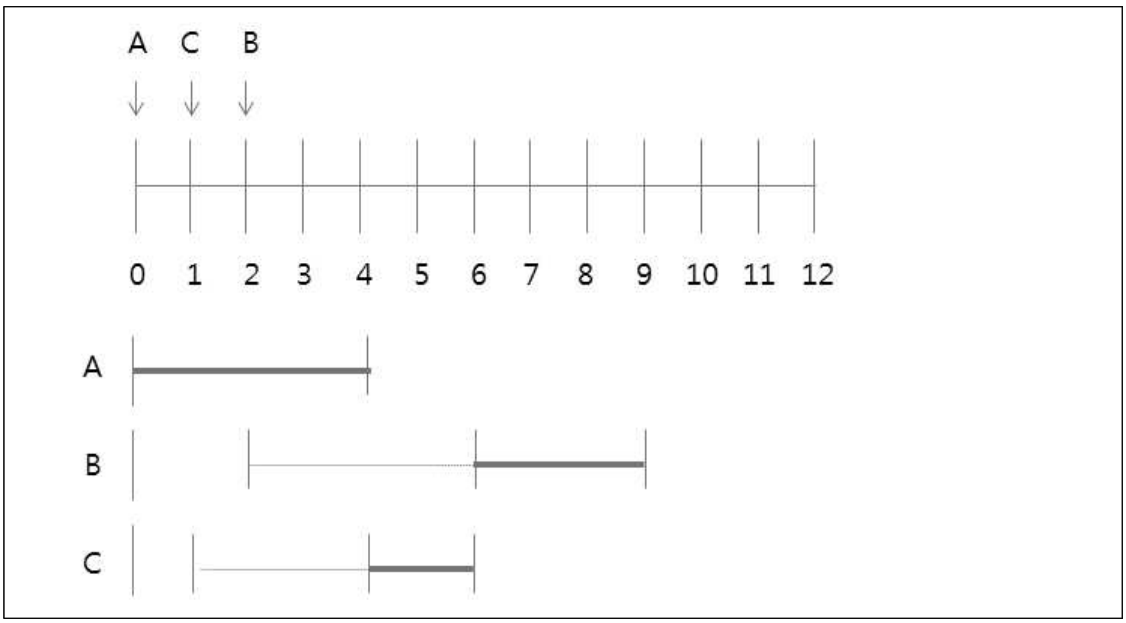
(4) 프로세스 스케줄링

1) 비선점형 기법
① 프로세스가 프로세서를 점유하면, 다른 프로세스는 현재 프로세스를 중단시킬 수 없는 기법이다.
② 프로세스가 프로세서를 할당받으면, 프로세스가 완료될 때까지 프로세서를 사용한다.
③ 모든 프로세스에 대한 공정한 처리가 가능하며, 일괄처리에 적합하다.
④ FCFS, SJF(Shortest Job First), HRN(Highest Response Next)

※ FCFS 기법
① 가장 대표적인 비선점형 스케줄링 기법이다.
② 대기리스트에 가장 먼저 도착한 프로세스 순서대로 CPU를 할당하므로, 알고리즘이 간단하고, 구현하기 쉽다.

[FCFS 평균반환시간]

프로세스	실행시간	도착시간
A	4	0
B	3	2
C	2	1



- 평균실행시간 = (4 + 3 + 2) / 3 = 3
- 평균대기시간 = (0 + 4 + 3) / 3 = 2.33…
- 평균반환시간 = 3 + 2.33… = 5.33…

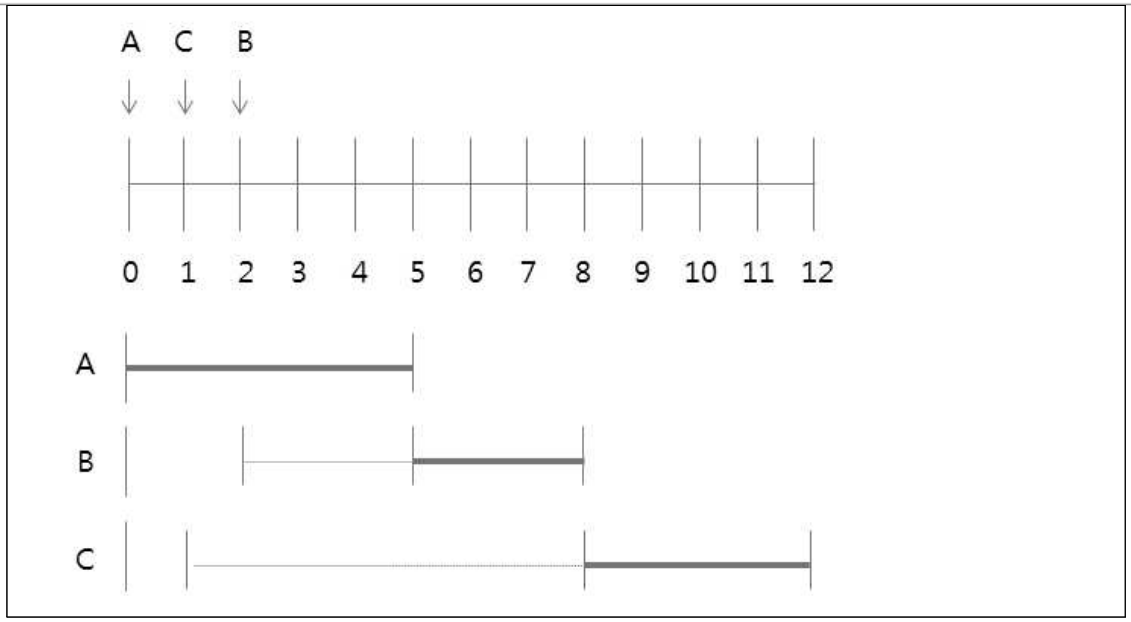
※ SJF(Shortest Job First)

① FCFS를 개선한 기법으로, 대기리스트의 프로세스들 중 작업이 끝나기까지의 실행시간 추정치가 가장 작은 프로세스에 CPU를 할당한다.

② FCFS 보다 평균 대기시간이 작지만, 실행시간이 긴 작업의 경우 FCFS보다 대기시간이 더 길어진다.

[SJF 평균반환시간]

프로세스	실행시간	도착시간
A	5	0
B	3	2
C	4	1



- 평균실행시간 = (5 + 3 + 4) / 3 = 4
- 평균대기시간 = (0 + 3 + 7) / 3 = 3.33…
- 평균반환시간 = 4 + 3.33… = 7.33…

※ HRN(Highest Response Next)

① SJF의 단점인 실행시간이 긴 프로세스와 짧은 프로세스의 지나친 불평등을 보완한 기법이다.

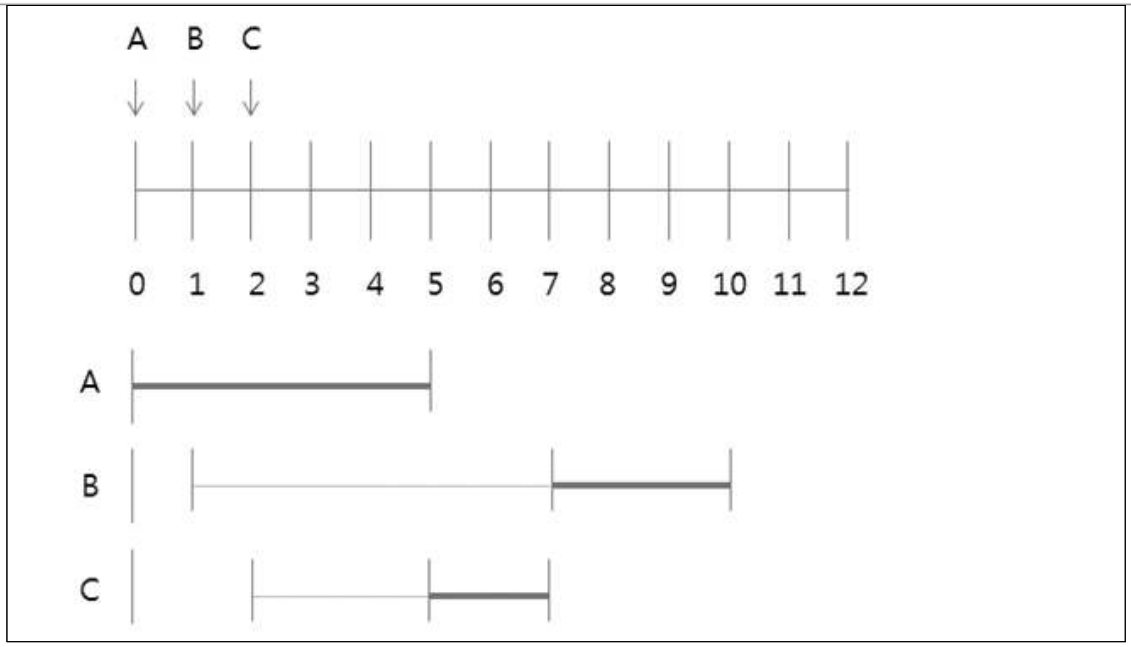
② 대기시간을 고려하여 실행시간이 짧은 프로세스와 대기시간이 긴 프로세스에게 우선순위를 높여준다

③ 우선순위 계산식에서 가장 큰 값을 가진 프로세스를 스케줄링 한다.

우선순위 = (대기사간 + 서비스 받을 시간) / 서비스 받을 시간

[HRN 평균반환시간]

프로세스	실행시간	도착시간
A	5	0
B	3	1
C	2	2



- 평균실행시간 = (5 + 3 + 2) / 3 = 3.33…
- 평균대기시간 = (0 + 6 + 3) / 3 = 3
- 평균반환시간 = 3.33… + 3 = 6.33…

2) 선점형 기법

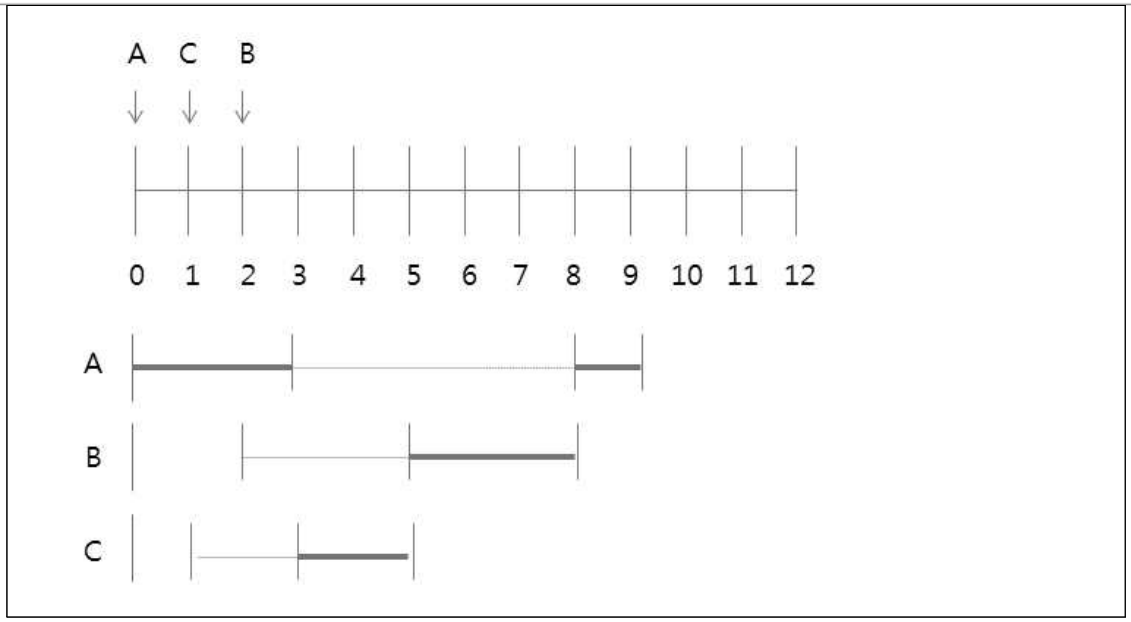
① 프로세스가 프로세서를 점유하면, 다른 프로세스는 현재 프로세스를 중단시킬 수 있는 기법이다.
② 우선순위가 높은 프로세스들을 처리할 때 유용하며, 대화식 시분할처리에 적합하다.
③ 선점으로 인해 많은 오버헤드를 발생시킨다.
④ RR(Round Robin, 라운드 로빈), SRT(Shortest Remaining Time), MLQ(Multi-level Queue, 다단계 큐), MFQ(Multi-level Feedback Queue, 다단계 피드백 큐)

※ RR(Round Robin, 라운드 로빈)

① FCFS를 선점형 스케줄링으로 변형한 기법이다.
② 대화형 시스템에서 사용되며, 빠른 응답시간을 보장한다.
③ RR은 각 프로세스가 CPU를 공평하게 사용할 수 있다는 장점이 있지만, 시간할당량의 크기는 시스템의 성능을 결정하므로 세심한 주의가 필요하다.

[RR 평균반환시간(시간할당량 : 3)]

프로세스	실행시간	도착시간
A	4	0
B	3	2
C	2	1



- 평균실행시간 = (4 + 3 + 2) / 3 = 3
- 평균대기시간 = (5 + 3 + 2) / 3 = 3.33…
- 평균반환시간 = 3 + 3.33… = 6.33…

2. 네트워크

(1) 프로토콜의 개념

- 네트워크상에 있는 디바이스 사이에서 정확한 데이터의 전송과 수신을 하기 위한 일련의 규칙들(set of rules)이다,
- 통신을 원하는 두 개체 간에 무엇을, 어떻게, 언제 통신할 것인가를 서로 약속하여 통신상의 오류를 피하도록 하기 위한 통신 규약이다.

(2) 프로토콜의 구성요소

① 구문(syntax) 요소: 데이터의 형식(format), 부호화 및 신호의 크기 등을 포함하여 무엇을 전송할 것인가에 관한 내용이 들어 있다.
② 의미(semantics) 요소: 데이터의 특정한 형태에 대한 해석을 어떻게 할 것인가와 그와 같은 해석에 따라 어떻게 동작을 취할 것인가 등 전송의 조정 및 오류 처리를 위한 제어정보 등을 포함한다.
③ 타이밍(timing) 요소: 언제 데이터를 전송할 것인가와 얼마나 빠른 속도로 전송할 것인가와 같은 내용을 포함한다.

(3) OSI 7계층 참조 모델(ISO Standard 7498)

1) 정의
① Open System Interconnection(개방형 시스템)의 약자로 개방형 시스템과 상호접속을 위한 참조 모델이다.
② ISO(International Organization for Standardization : 국제 표준화 기구)에서 1977년 통신기능을 일곱 개의 계층으로 분류하고 각 계층의 기능 정의에 적합한 표준화된 서비스 정의와 프로토콜을 규정한 사양이다.
③ 같은 종류의 시스템만이 통신을 하는 것이 아니라 서로 다른 기종이 시스템의 종류, 구현방법 등에 제약을 받지 않고 통신이 가능하도록 통신에서 요구되는 사항을 정리하여 표준 모델로 정립하였다.

2) 기본요소
① 개방형 시스템(open system) : OSI에서 규정하는 프로토콜에 따라 응용 프로세스(컴퓨터, 통신제어장치, 터미널 제어장치, 터미널)간의 통신을 수행할 수 있도록 통신기능을 담당하는 시스템
② 응용 실체/개체(application entity) : 응용 프로세스를 개방형 시스템상의 요소로 모델화한 것
③ 접속(connection) : 같은 계층의 개체 사이에 이용자의 정보를 교환하기 위한 논리적인 통신 회선
④ 물리매체(physical media) : 시스템 간에 정보를 교환할 수 있도록 해주는 전기적인 통신 매체 (통신회선, 채널)

3) 각 레이어의 의미와 역할

① Physical layer(물리 계층)
㉠ 물리계층은 네트워크 케이블과 신호에 관한 규칙을 다루고 있는 계층으로 상위계층에서 보내는 데이터를 케이블에 맞게 변환하여 전송하고, 수신된 정보에 대해서는 반대의 일을 수행한다. 다시 말해서 물리계층은 케이블의 종류와 그 케이블에 흐르는 신호의 규격 및 신호를 송수신하는 DTE/DCE 인터페이스 회로와 제어순서, 커넥터 형태 등의 규격을 정하고 있다. 이 계층은 정보의 최소 단위인 비트 정보를 전송매체를 통하여 효율적으로 전송하는 기능을 담당한다.
㉡ 전송매체는 송신자와 수신자간에 데이터 흐름의 물리적 경로를 의미하며, 트위스트 페어케이블, 동축케이블, 광섬유케이블, 마이크로파 등을 사용할 수 있다.

② Data Link layer(데이터 링크 계층)
㉠ 데이터 링크층은 통신 경로상의 지점간 (link-to-link)의 오류없는 데이터 전송에 관한 프로토콜이다. 전송되는 비트의 열을 일정 크기 단위의 프레임으로 잘라 전송하고, 전송 도중 잡음으로 인한 오류 여부를 검사하며, 수신측 버퍼의 용량 및 양측의 속도 차이로 인한 데이터 손실이 발생하지 않도록 하는 흐름제어 등을 한다.
㉡ 인접한 두 시스템을 연결하는 전송 링크 상에서 패킷을 안전하게 전송하는 것이다.

③ Network layer(네트워크 계층)
㉠ 네트워크층은 패킷이 송신측으로부터 수신측에 이르기까지의 경로를 설정해주는 기능을 수행한다.
㉡ 두 개의 통신 시스템 간에 신뢰할 수 있는 데이터를 전송할 수 있도록 경로선택과 중계기능을 수행하고, 이 계층에서 동작하는 경로배정(routing)프로토콜은 데이터 전송을 위한 최적의 경로를 결정한다.
㉢ IP 프로토콜이 동작하면서 IP헤더를 삽입하여 패킷을 생성하며 송신자와 수신자간에 연결을 수행하고 수신자까지 전달되기 위해서는 IP헤더 정보를 이용하여 라우터에서 라우팅이 된다.

④ Transport layer(전송 계층)
㉠ 전송층은 수신측에 전달되는 데이터에 오류가 없고 데이터의 순서가 수신측에 그대로 보존되도록 보장하는 연결 서비스의 역할을 하는 종단간(end-to-end) 서비스 계층이다.
㉡ 종단간의 데이터 전송에서 무결성을 제공하는 계층으로 응용 계층에서 생성된 긴 메시지가 여러 개의 패킷으로 나누어지고, 각 패킷은 오류없이 순서에 맞게 중복되거나 유실되는 일 없이 전송되도록 하는데 이러한 전송 계층에는 TCP, UDP 프로토콜 서비스가 있다.

⑤ Session layer (세션 계층)
㉠ Session Layer는 두 응용프로그램(Applications) 간의 연결설정, 이용 및 연결해제 등 대화를 유지하기 위한 구조를 제공한다. 또한 분실 데이터의 복원을 위한 동기화 지점(sync point) 을 두어 상위 계층의 오류로 인한 데이터 손실을 회복할 수 있도록 한다.
㉡ 시스템 간의 통신을 원활히 할 수 있도록 세션의 설정과 관리, 세션 해제 등의 서비스를 제공하고 필요시 세션을 재시작하고 복구하기도 한다.

⑥ Presentation layer (표현 계층)
㉠ Presentation Layer전송되는 정보의 구문 (syntax) 및 의미 (semantics)에 관여하는 계층으로, 부호화 (encoding), 데이터 압축 (compression), 암호화 (cryptography) 등 3가지 주요 동작을 수행한다.

⑦ Application layer(응용 계층)
㉠ Application Layer네트워크 이용자의 상위 레벨 영역으로, 화면배치, escape sequence 등을 정의하는 네트워크 가상 터미널 (network virtual terminal), 파일전송, 전자우편, 디렉토리 서비스 등 하나의 유용한 작업을 할 수 있도록 한다.

제12장 제품소프트웨어 패키징

제1절 제품소프트웨어 패키징

1. 애플리케이션 패키징

(1) 애플리케이션 패키징

1) 패키징

① 프로그램 제작자가 최종사용자가 사용 할 프로그램을 다양한 환경에서 쉽게 자동으로 설치(업데이트/삭제 가능)할 수 있게 패키지를 만들어 배포하는 과정을 말한다.(매뉴얼 포함)
② 개발이 완료된 제품 소프트웨어를 고객에게 전달하기 위한 형태로 패키징하고, 설치와 사용에 필요한 제반 절차 및 환경 등 전체 내용을 포함하는 매뉴얼을 작성하며, 제품 소프트웨어에 대한 패치 개발과 업그레이드를 위해 버전 관리를 수행할 수 있다.

2) 소프트웨어 패키징
① 소프트웨어 패키징은 각 모듈별로 생성한 실행 파일들을 묶어서 배포용 설치 파일을 만드는 것이다.
② 소프트웨어 패키징시에는 여러 가지 환경과 사용자의 요구사항을 반영할 수 있도록 변경 및 개선에 대한 관리를 고려해야 한다.

3) 릴리즈 노트
- 릴리즈 노트는 소프트웨어 제품과 함께 배포되는데 이 문서들에는 제품의 주요 변경 사항이 담겨 있다. 개발 과정에서 정리된 릴리즈 정보를 사용자(고객)과 공유하기 위한 문서이다.
- 릴리즈 노트를 통해 소프트웨어 버전 관리나 릴리즈 정보를 체계적으로 관리할 수 있다.

[릴리즈 노트 구성 항목]

구분	설명
헤더 (Header)	문서 이름(예: 릴리스 노트), 제품 이름, 릴리스 번호, 출시일, 노트 날짜, 노트 버전 등
개요	제품 및 변경에 대한 간략한 개요
목적	버그 픽스와 새로운 기능 목록
이슈 요약	버그 수정이나 개선사항에 대한 짧은 설명
재현 단계	버그 발생을 재현하기 위한 절차
해결책 (Solution)	버그 수정을 위한 수정/개선사항의 간단한 설명
최종 사용자 영향	버전 변경에 따른 최종 사용자 기준의 기능 및 응용 프로그램상의 영향도 기술
SW지원 영향도	버전 변경에 따른 SW 지원 프로세스 및 영향도 기술
참고	소프트웨어나 하드웨어의 설치, 업그레이드, 제품 문서화에 관한 참고사항 (문서화 업데이트 포함)
면책	회사 및 표준 제품 관련 메시지 프리웨어, 불법 복제 금지 등 참조에 대한 고지 사항
연락처	사용자 지원 및 문의 관련한 연락처 정보

※ 예외 케이스(베타 버전 출시, 긴급 버그 수정 출시, 업그레이드,,,)가 발생할 때에는 이에 해당하는 개선 항목들이 추가된다.

2. 애플리케이션 배포 도구

① 배포를 위한 패키징 시에 디지털 컨텐츠의 지적 재산권을 보호하고 관리하는 기능을 제공하며, 안전한 유통과 배포를 보장하는 도구이자 솔루션이다.

② 애플리케이션 배포 도구 활용 시에는 암호화/보안, 이 기종 간의 연동, 지속적 배포, 사용자 편의성을 고려한다.

③ 애플리케이션 배포도구 구성요소

- 암호화(Encryption) : 컨텐츠/라이센스 암호화, 전자 서명(Symmetric/Asymmetric Encryption, Digital Signature, PKI)
- 인증(Authentication) : 라이센스 발급 및 사용의 기준이 되는 사용자 인증 기술
- 키 관리(Key Management) : 컨텐츠를 암호화한 키에 대한 저장 및 배포기술(Centralized, Enveloping)
- 저작권 표현(Right Expression) : 라이센스의 내용 표현 기술(XrML/MPEG-21 REL, ODRL)
- 크래킹 방지(Tamper Resistance) : 크랙에 의한 컨텐츠 사용 방지 기술(Secure DB, Secure Time Management, Encryption)

3. DRM(Digital Rights Management)

(1) DRM의 개요

① 디지털 저작권 관리의 약자로, 디지털 콘텐츠 제공자의 권리와 이익을 안전하게 보호하며 불법복제를 막고 사용료 부과와 결제대행 등 콘텐츠의 생성에서 유통·관리까지를 일괄적으로 지원하는 기술이다.
② 디지털 콘텐츠의 생성과 이용까지 유통 전 과정에 걸쳐 디지털콘텐츠를 안전하게 관리 및 보호하고, 부여된 권한정보에 따라 디지털콘텐츠의 이용을 통제하는 기술이다.

(2) DRM의 개념적인 구성 요소

① 사용자(User) : 부여된 접근권한과 상태에 따라 콘텐츠를 이용할 주체
② 콘텐츠(Contents) : 지적 자산가치의 정보 단위(허가되지 않은 사용자로부터 보호되어야 할 대상)
③ 접근권한(Permission) : 콘텐츠의 이용권리
④ 상태(Condition) : 콘텐츠 이용 권리의 요구조건 및 제한요소

(3) DRM 시스템 구성 요소

구분	설명
콘텐트 제공자 (Contents Provider)	콘텐츠를 제공하는 저작권자
콘텐트 분배자 (Contents Distributor)	쇼핑몰 등으로써 암호화된 콘텐츠 제공
패키저(Packager)	콘텐츠를 메타데이터와 함께 배포 가능한 단위로 묶는 기능
보안 컨테이너	원본을 안전하게 유통하기 위한 전자적 보안장치
DRM 컨트롤러	배포된 콘텐츠의 이용 권한을 통제
클리어링 하우스 (Clearing House)	키관리 및 라이센스 발급 관리

(4) DRM의 핵심적 기술 요소

구분	설명	
암호화 (Encryption)	콘텐츠 및 라이센스를 암호화하고, 전자서명을 할 수 있는 기술	PKI, Encryption, Digital Sinature
키관리 (Key Management)	콘텐츠를 암호화한 키에 대한 저장 및 배포기술	Centralized, Enveloping
암호화 파일 생성(Packager)	콘텐츠를 암호화된 콘텐츠로 생성하기 위한 기술	Pre-packaging, On-the-fly Packaging
식별기술 (Identification)	콘텐츠에 대한 식별체계 표현 기술	DOI, URI
저작권표현 (Right Expression)	라이센스의 내용 표현 기술	ODRL, XrML/MPGE-21 REL
정책관리(Policy management)	라이센스 발급 및 사용에 대한 정책표현 및 관리 기술	XML, Contents Management System
크랙 방지 (Tamper Resistance)	크랙에 의한 콘텐츠 사용방지 기술	Secure DB, Secure Time Management, Encryption
인증 (Authentication)	라이센스 발급 및 사용의 기준이 되는 사용자 인증기술	SSO,ID/PW, 디지털인증, 이메일인증
인터페이스 (Interface)	상이한 DRM 플랫폼 간의 상호 호환성 인터페이스 및 인증 기술	IPMP
이벤트보고 (Event Reporting)	콘텐츠의 사용이 적절하게 이루어지고 있는지 모니터링 기술. 불법유통이 탐지되었을 때 이동경로를 추적에 활용	
사용권한 (Permission)	콘텐츠의 사용에 대한 권한을 관리하는 기술요소	렌더파미션, 트랜스포트 퍼미션, 데리버티브 퍼미션

제2절 제품소프트웨어 매뉴얼 작성

1. 제품소프트웨어 매뉴얼 작성

(1) 제품소프트웨어 매뉴얼의 개요

① 사용자에게 제품의 권장사항, 제품 설명, 설치법, 사용법 외의 부록을 첨부하여 제품 사용에 따른 이해와 만족도를 높인다.
② 시스템 문서 : 시스템 자체와 그 세부 구성을 설명하는 문서를 나타낸다. 요구사항 문서, 설계 결정, 아키텍처 설명, 프로그램 소스 코드 및 도움말 가이드 등을 포함한다.
③ 사용자 설명서 : 주로 소프트웨어 제품 최종 사용자와 시스템 관리자를 위해 작성된 매뉴얼이 수록되어 있다. 사용자 설명서에는 튜토리얼, 사용자 가이드, 문제 해결 매뉴얼, 설치 및 참조 매뉴얼이 포함된다.

(2) 설치 매뉴얼

① 설치 매뉴얼은 개발자 기준이 아닌 사용자 중심으로 작성한다.
② 최초 설치 진행부터 완료까지 설치 방법을 순차적으로 상세히 설명한다.
③ 각 단계별 메시지 및 해당 화면을 순서대로 전부 캡쳐하여 사용자가 이해하기 쉽도록 구성한다.
④ 설치 도중에 발생하는 오류나 이상 현상도 요약 정리하여 설명한다.

(3) 사용자 매뉴얼

1) 사용자 매뉴얼 작성 단계

① 작성 지침 정의: 매뉴얼 작성을위한 지침을 설정한다.
② 사용자 매뉴얼 구성 요소 정의: 제품 소프트웨어의기능, 구성 객체 목록, 객체별 메소드, 사용 예제, 환경 설정 방법
③ 구성 요소별 내용 작성: 구성 요소별 내용을 작성한다.
④ 사용자 매뉴얼 검토: 작성된 사용자 매뉴얼이 개발된 제품의 기능을 제대로 설명하는지, 누락여부 등을 검토

2) 사용자 매뉴얼 기본 사항

구분	설명
제품 소프트웨어 개요	제품 소프트웨어의 주요 기능 및 UI 설명 UI 및 화면 상의 버튼, 프레임 등을 도식화하여 설명
제품 소프트웨어 사용	제품 소프트웨어를 사용하기 위한 최소 환경 설명 PC 사양(CPU, Memory 등), OS 버전 명시 제품 소프트웨어 최초 동작을 위한 설명(실행 파일 or Website URL 등)
제품 소프트웨어 관리	제품 소프트웨어의 사용 종료 및 관리 등에 대한 내용 기재
모델, 버전별 특징	제품 구별을 위한 모델, 버전별 UI 및 기능의 차이 간단히 기술
기능, I/F 특징	제품의 기능 및 Interface 의 특징을 간단히 기술
제품 소프트웨어 구동 환경	개발 언어 및 호환 OS 설치 마법사(Setup Wizard) 이후 사용자가 구동하기까지의 과정 요약 (Windows, Linux)

2. 국제 표준 제품 품질 특성

- 상품과 서비스의 국제교류를 용이하게 하고 지식·과학·기술·경제 분야의 국제간 협력을 증진하기 위해 표준화와 이에 관련된 여러 가지 활동을 국제 규모로 발전, 촉진시키기 위한 목적으로 발족되었다

[소프트웨어 품질의 관점]

제3절 제품소프트웨어 버전관리

1. 소프트웨어 버전관리 도구

(1) 버전 관리 필요성
① 오류 복구
② 이전 버전으로의 복구
③ 개별 수정 부분에 대한 전체 동기화 과정의 자동화
④ 소스 코드의 변경 사항 추적
⑤ 안정적인 대규모 수정 작업
⑥ Branch를 통해 프로젝트에 미치는 영향을 최소화하는 동시에 새로운 부분 개발
⑦ 백업 기능

(2) 버전 관리 용어

구분	설명
Repository	관리 대상의 모든 파일, 관련 버전, 변경 이력 정보를 저장하는 공유 데이터베이스 - Local Repository : 현재 개발 환경에서의 파일 저장소 - Remote Repository : 서버상의 파일 저장소
Trunk	주류, 프로젝트의 중심
Branch	주류에서 파생된 프로젝트, 개발 라인 - Master Branch : 메인 개발 라인
Tag	특정 시점의 프로젝트 전체를 복사 및 보관
Revision	저장소에 저장된 파일의 버전 새롭게 저장소에 커밋할 경우, 해당 파일의 개정 번호 증가
Check Out	저장소에서 선택한 파일 또는 디렉토리를 현재 작업 환경으로 복사 디렉토리의 경우, 포함된 파일 및 하위 디렉토리 모두 체크 아웃 버전 관리를 위한 파일이 포함됨 (저장소와의 연결을 위함)
Check In, Commit	작업 파일 또는 디렉토리의 변경 사항을 저장소에 새 버전으로 저장
Conflict	동일한 파일에 대한 변경 사항 확인 충돌이 발견할 경우, 해결이 완료되어야 커밋 가능
Import	(버전 관리되고 있지 않은) 로컬 디렉토리의 파일을 처음으로 저장소에 저장
Export	체크 아웃과는 달리, 버전 관리 파일을 제외한 소스 파일만을 받아옴
Change log, History	수정 기록
Update, Sync	동기화, 저장소에 있는 최신 버전의 파일 또는 디렉토리 가져옴
Fork	하나의 소프트웨어 소스 코드를 통째로 복사하여 독립적인 새로운 소프트웨어 개발 허용되는 라이센스를 따라야 함

(3) 버전 프로그램 종류

① 버전 관리 방식별 구분

구분	설명
공유 폴더 방식 (RCS, SCCS)	- 매일 개발 완료 파일은 약속된 위치의 공유 폴더에 복사 - 담당자 한 명이 매일 공유 폴더의 파일을 자기 PC로 복사하고 컴파일하여 에러 확인과 정상 동작 여부 확인 - 정상 동작일 경우 다음날 각 개발자들이 동작 여부 확인
클라이언트/서버 방식 (CVS, SVN)	- 중앙에 버전 관리 시스템이 항상 동작 - 개발자들의 현재 작업 내용과 이전 작업내용 축적 용이 - 서로 다른 개발자가 같은 파일을 작업했을 때 경고 출력 - 트랙이나 cvs 뷰와 같은 GUI 툴을 이용 모니텅링 가능
분산 저장소 방식 (Git, Bitkeeper 등)	- 로컬 저장소와 원격저장소 구조 - 중앙의 저장소에서 로컬에 복사한 순간 개발자 자신만의 로컬 저장소에 생성 -개발 완료한 파일 수정 이후 로컬 저장소에 커밋한 이후 다시 원격 저장소에 반영하는 방식

② 대상에 따른 구분

구분	유형	종류
저장소 구분	로컬 버전 관리	RCS
	중앙 집중형 버전 관리	CVS, SVN, Clear case
	분산형 버전 관리	Git,Mercurial
소스 공개 유형	Open Source 툴	CVS, SVN
	상용 버전 관리 툴	PVCS, Clear Case

③ 버전관리 툴 특징

구분	설명
CVS (Concurrent Version System)	- 서버와 클라이언트로 구성되어 다수의 인원이 동시에 범용적인 운영체제로 접근 가능하여 버전 관리를 가능케 한다 - Client 가 이클립스에 내장되어 있다
SVN (Subverion)	- GNU 의 버전 관리 시스템으로 CVS 의 장점은 이어받고 단점은 개선하여 2000년에 발표되었다. 사실상 업계 표준으로 사용되고 있다
RCS (Revision Control System)	- CVS 와 달리 소스 파일의 수정을 한 사람만으로 제한하여 다수의 사람이 파일의 수정을 동시에 할 수 없도록 파일을 잠금하는 방식으로 버전 컨트롤을 수행한다.
Bitkeeper	- SVN 과 비슷한 중앙 통제 방식의 버전컨트롤 툴로서 대규모 프로젝트에서 빠른 속도를 내도록 개발되었다.
Git	- 기존 리눅스 커널의 버전 컨트롤을 하는 Bitkeeper를 대체 하기 위해서 나온 새로운 버전 컨트롤로 현재의 리눅스는 이것을 통해 버전 컨트롤이 되고 있다. Git는 속도에 중점을 둔 분산형 버전 관리 시스템(DVCS)이며, 대형 프로젝트에서 효과적이고 실제로 유용하다. - Git 는 SVN 과 다르게 Commit 은 로컬 저장소에서 이루어 지고 push 라는 동작으로 원격 저장소에 반영된다.(로컬 저장소에서 작업이 이루어져 매우 빠른 응답을 받을 수 있다.) - 받을 때도 pull 또는 Fetch로 서버에서 변경된 내역을 받아 올 수 있다.
Clear Case	- IBM에서 제작되었다. - 복수 서버, 복수 클라이언트 구조이며 서버가 부족할 때 필요한 서버를 하나씩 추가하여 확장성을 기할 수 있다.

2. 빌드 자동화 도구

(1) 빌드 자동화 도구의 개요
① 빌드단계에서는 compile, testing, inspection, deploy등의 과정등이 포함 될 수 있다.
② 빌드 자동화를 위한 툴은 언어마다 매우 다양한데, C나 C++ 등을 사용할때는 보통 Makefile, Java는 Ant, Maven, Gradle(구글이 안드로이드의 기본 빌드 시스템으로 Gradle을 선택하면서 사용자가 급증함) Scala는 sbt를 주로 사용한다.

(2) 빌드 도구의 기능
① 코드 컴파일 : 테스트를 포함한 소스코드 컴파일
② 컴포넌트 패키징 : 자바의 jar 파일이나 윈도우의 exe 파일 같은 배포할 수 있는 컴포넌트를 묶는 작업
③ 파일 조작 : 파일과 디렉토리를 만들고 복사하고 지우는 작업
④ 개발 테스트 실행 : 자동화된 테스트 진행
⑤ 버전관리 도구 통합 : 버전관리 시스템 지원
⑥ 문서 생성 : API문서(ex JAVA Doc)를 생성
⑦ 배포 기능 : 테스트 서버(alpha, beta) 배포 지원
⑧ 코드품질분석 : 자동화된 검사도구(findbug,checkstyle,pmd.)를 통한 코드 품질 분석

(3) 빌드 자동화 도구 툴

① ANT
- 안정성이 좋고, 문서화가 잘 되어 있다.
- target기능을 이용해서 세밀하게 빌드 할 수 있다.(자바 소스 파일 컴파일, jar, war, ear, zip 파일의 생성, javadoc 생성, 파일이나 폴더의 이동 및 복사, 삭제, 작업에 대한 의존성 설정, 외부 프로그램 실행등)
- 복잡한 빌드환경에서도 적절히 대처할 수 있는 많은 task를 제공한다.

② Maven
- 아주 적은 설정 만으로도 프로젝트를 빌드하고, 테스트를 실행하고, 품질 보고서를 생성할 수 있다.
- POM(Project Object Model)을 통해서 jar 파일의 의존성 관리, 빌드, 배포, 문서생성, Release 등을 관리 할 수 있다.
- 표준화된 디렉토리 레이아웃을 제공한다.
- 미리 제공된 plugin들을 활용해 거의 모든 작업을 수행 할 수 있다.

③ Gradle
- 기존의 Ant와 Maven을 보완
- 오픈소스기반의 build 자동화 시스템으로 Groovy 기반 DSL(Domain-Specific Language)로 작성
- Build-by-convention을 바탕으로함: 스크립트 규모가 작고 읽기 쉬움
- Multi 프로젝트의 빌드를 지원하기 위해 설계됨
- 설정 주입 방식 (Configuration Injection)

④ Jenkins

- 초창기 Hudson 이라는 이름을 가졌지만 오라클과 문제로 인해 이름을 바꾸게 되었다
- 프로젝트 표준 컴파일 환경에서의 컴파일 오류 검출
- 자동화 테스트 수행(CVS/SVN/Git과 같은 버전관리시스템과 연동하여 코드 변경을 감지)
- 코드 표준 준수여부 체크
- 프로파일링 툴을 이용한 소스 변경에 따른 성능 변화 감시
- 결합 테스트 환경에 대한 배포작업

