1. 바이트 코드와 자바 가상 기계

바이트코드

- 자바 가상 기계에서 실행 가능한 바이너리 코드
- 자바 가상 기계가 작동중인 플랫폼에서 실행
- 자바 가상 기계가 인터프리터 방식으로 바이트코드를 해석

자바 가상 기계(JVM : Java Virtual Machine)

- 동일한 자바 실행 환경 제공
    - 각기 다른 플랫폼에 설치
- 자바 가상 기계 = 플랫폼에 종속적
    - 자바 가상 기계 = 플랫폼마다 각각 작성됨 (ex Linux JVM -> Windows JVM 작동 X)
- 자바 응용프로그램 실행
    - JVM이 응용프로그램을 구성하는 클래스 파일(.class)의 바이트 코드 실행

C vs Java

C 언어

- 여러 소스 파일로 분할해 개발(.c)
- 링크를 통해 실행에 필요한 모든 코드를 하나의 실행파일(.exe)에 저장
- 실행 파일(exe)가 메모리 상에 모두 올려져야 실행 -> 메모리 공간이 한정적인 경우 문제 발생

JAVA

- 여러 소스(.java)로 나누어 개발
- 바이트 코드(.class)를 하나의 실행 파일(exe)로 만드는 링크 과정 X
- main() 메소드를 가진 클래스에서부터 실행
- JVM은 필요할 때, class파일 로드 -> 적은 메모리로도 실행이 가능

2. JAVA의 기본 구조

```java
/*
소스 파일 : Hello.java
*/

//static = 객체 생성 없이 바로 사용이 가능한 메소드
// 즉, 객체를 위해 만든 메소드 X -> Function 자체만을 사용하기 위한 메소드
// ex) math 관련 함수(계산 목적이 강한 메소드등)

public class Hello{

    public static int sum(int n, int m){
        return n+m;
    }

    public static void main(String[] args){
        int i = 20;
        int s;
        char a;

        s = sum(i,10); //sum 메소드 호출
        a = '?';
        System.out.println(a);
        System.out.println("Hello");
        System.out.println(s);
    }
}
```

3. 식별자

식별자

- 클래스, 변수, 상수, 메소드 등에 붙이는 이름

식별자의 원칙

- '@', '#', '!'등의 특수문자, 공백 또는 탭 = 식별자 사용 X, '_', '$'는 사용 가능
- 유니코드 문자 사용 O, **한글 사용 가능**
- 자바 언어의 키워드 = 식별자로 사용 **불가**
- '_' or '$'를 식별자의 첫 번째 문자로 사용이 가능은 하나, **일반적으로는 사용 X**
- 불린 리터럴(true,false), 널 리터럴(null) 식별자 사용 **불가**
- 길이제한 X

대소문자 구별

- Test, test = 별개

4. 자바 키워드

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	tarasient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

열거형

열거형 C언어에서의 Define과 비슷한 맥락

열거형 출력 방법

```
Day day1 = Day.FRIDAY;
System.out.println(day1);

String day2 = Day.FRIDAY.name();
System.out.println(day2);

Day day3 = Day.valueOf("FRIDAY");
System.out.println(day3);
```

주의점

- 대문자로 작성

5. 타입캐스팅

자동(묵시적)

- Promotion

강제(명시적)

- Casting

5_1 ex)

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

    int i = 20; //4Byte
    double j =20.1; //8Byte

    int sum1 = i+j;
    //Promotion 발생 -> 자동 형변환(큰쪽으로), [즉, 컴파일러가 자동으로 double 형변환]
    //그러나, sum 자체가 int형이기 때문에, 에러 발생 (보통 작은 사이즈 -> 큰사이즈 = 묵시적
변환)

    double sum2 = i+j;
    //Promotion 발생
    // sum 자체 double -> 문제없이 묵시적 변환 실행

    int sum3 = i + (int)j;
    //명시적 수행
    //j앞에 (int)를 붙여줌으로써 강제 타입 캐스팅
}
```

5_2 연산식에서의 자동 타입 변환

연산은 같은 타입의 피연산자(operand)간에만 수행

서로 다른 피연산자 = 같은 타입으로 변환

. 객체지향 vs 절차지향

- Class = 껍데기
- 객체 = 실체화된 데이터
- Data + Function -> 객체로 묶음 = 객체지향
- Data <-> Function 사이의 Dependency 존재 X = 절차지향