객프

자바의 특성

플랫폼 독립성

플랫폼에 종속죄지 않는 독립적인 바이트 코드로 컴파일되며 자바 가상 기계만 있으면 하드웨어/운영체제를 막론하고 자바 프로그램이 실행이 가능하다.

객체 지향

자바는 객체 지향 언어로서 캡슐화, 상속, 다형성을 지원

클래스로 캡슐화

자바는 객체 지향 언어의 캡슐화 원칙을 철저히 지켜, 변수나 메소드는 반드시 클래스 내에 구현하도록 한다.

소스와 클래스파일

클래스 파일에는 반드시 하나의 자바 클래스만 들어있다. 그러므로 하나의 자바 소스 파일에 여러 개의 클래스를 작성한 경우, 컴파일하면 클래스마다 별도의 클래스 파일이 생성된다.

실행 코드 배포

자바 응용프로그램은 한 개의 클래스 파일 또는 다수의 클래스 파일로 구성된다. 다수의 클래스 파일을 jar 파일 형태로 압축하여 배포하거나 실행할 수 있다. 자바의 실행은 main()메소드에서 시작되며, 하나의 클래스 파일에 두 개 이상의 main()메소드가 있을 수 없다.

패키지

서로 관련 있는 클래스는 패키지로 묶어 관리한다. 패키지는 파일 시스템의 폴더와 같은 개념이다.

멀티스레드

하나의 자바 프로그램이 다수의 작업을 처리할 수 있도록 다수의 스레드가 동시에 실행할 수 있는 환경을 지원한다. 자바는 운영체제의 도움 없이 멀티스레드 프로그래밍이 가능하기 때문에, 멀티스레드를 지원하지 않는 운영체제에서도 자바를 이용하면 멀티스레드 프로그램을 개발할 수 있다.

가비지 컬렉션
자바 언어는 메모리를 할당받는 기능은 있지만, 메모리를 반환하는 기능은 없다. 프로그램 내에 사용되지 않는 메모리(가비지)는 자바 가상 기계의 가비지 컬렉션 기능에 의해 자동으로 회수된다.

실시간 응용 시스템에 부적합
자바 응용 프로그램은 실행 도중 예측할 수 없는 시점에 가비지 컬렉션이 진행되므로 프로그램 실행이 일시적으로 중단된다.

자바 프로그램은 안전하다.
자바 언어는 타입체크가 매우 엄격하며 포인터의 개념이 없기 때문에 잘못된 자바 프로그램으로 인해 컴퓨터 시스템이 중단되는 일은 없다.

프로그램 작성이 쉽다.
포인터의 개념이 없기 때문에 프로그램 작성에 부담이 적다.

실행 속도를 개선하기 위해 JIT컴파일러가 사용된다.
CPU의 기계어 코드로 컴파일하고 CPU가 바로 기계어를 실행하도록하는 JIT컴파일링 기법을 이용하므로 실행 성능이 개선되었다. -> 플랫폼 독립성 포기

식별자 이름 규칙

특수문자, 공백은 식별자로 사용할 수 없으나, '_','$'는 예외로 사용할 수 있다.
한글도 유니코드 문자 사용가능하기에 식별자로 가능하지만 전세계에서 읽어야하고 정상 동작을 안 할 수도 있으므로 지양
자바의 키워드는 식별자로 사용할 수 없다.
식별자의 첫 번째 문자로 숫자는 사용할 수 없다.
대소문자를 구별한다
길이 제한이 없다(가독성 높아야 함)

기본: 가독성 높은 이름
목적을 나타내는 이름 붙이기: s보다는 sum
충분히 긴 이름으로 붙이기 (변수는 소문자로 시작, 클래스는 대문자로 시작)

자바 언어의 이름 붙이는 관습: 헝가리언 이름 붙이기(int-> iNum)
클래스 이름
첫 번째 문자는 대문자로 시작, 소스파일이름과 같아야함

각 단어의 첫 번째 문자만 대문자(AutoVendingMachine)
변수, 메소드 이름
첫 단어 이후 각 단어 첫 번째 문자는 대문자로 시작(myAge)
상수이름
모든 문자를 대문자로 표시(final static double PI)
함수이름
동사로 시작

기본 타입:8개

Boolean(1바이트)
char 2
byte 1
short 2
int 4
long 8
float 4
double 8

레퍼런스 타입: 1개

레퍼런스 타입은 한 가지이지만 용도는 다음과 같이 3가지이다,

배열에 대한 레퍼런스
클래스에 대한 레퍼런스
인터페이스에 대한 레퍼런스

변수

프로그램 실행 중에 값을 임시 저장하기위한 공간
데이터 타입에서 정한 크기의 메모리 할당
변수 값은 프로그램 수행 중 변경될 수 있음

변수 선언

변수 타입 다음에 변수 이름을 적어 선언

리터럴(3, 15, 10, 'A', True, False, NULL…) != 상수

프로그램에서 직접 표현한 값
정수, 실수 , 문자, 논리, 문자열, 리터럴 있음

상수 선언

final 키워드 사용
선언 시 초기값 지정
실행 중 변경 불가능

타입 변환

자동변환

치환문이나 수식 내에서 타입이 일치하지 않을 때, 컴파일러는 오류 대신 작은 타입을 큰 타입으로 자동 변환한다.

long m =. 5 (리터럴 25는 int 타입, 25가 long타입으로 자동 변환)
double d = 3.14*10(10을 10.0으로)

강제 변환

큰 타입의 값을 작은 타입의 값으로 변환할 때

int n = 300;
byte b = (byte)n
하지만 강제 변환을 하면 컴파일 오류가 발생하지 않을 뿐 여전히 300에서 256을 초과한 만큼 즉 44가 변수 b에 저장되어 데이터 손실이 발생한다.
실수->정수 소수점 이하의 손실 발생

자바에서 사용자로부터 키 입력을 받는 방법

System.in (키보드 장치를 직접 제어하고 키 입력을 받는 표준 입력 스트림 객체)
키 값을 문자형이 아닌 바이트형으로 리턴한다.

Scanner클래스는 사용자가 입력하는 키 값을 공백 문자를 기준으로 분리하여 토큰 단위로 읽는다

next()와 nextLine()차이 = 공백이 낀 문자열을 입력 받기 위해서는 후자 사용 후자는 엔터키로 분리 하기 때문

조건식에 산술식 쓰기 말 것 if(number % 3 == 0) -> if(iMod3==0)

if else if else 문 else에 예외처리

문자열 비교 식별자.equals(“string”)로 비교

반복 횟수 알고 있을 때 : for문 사용/모를 때 while문 사용

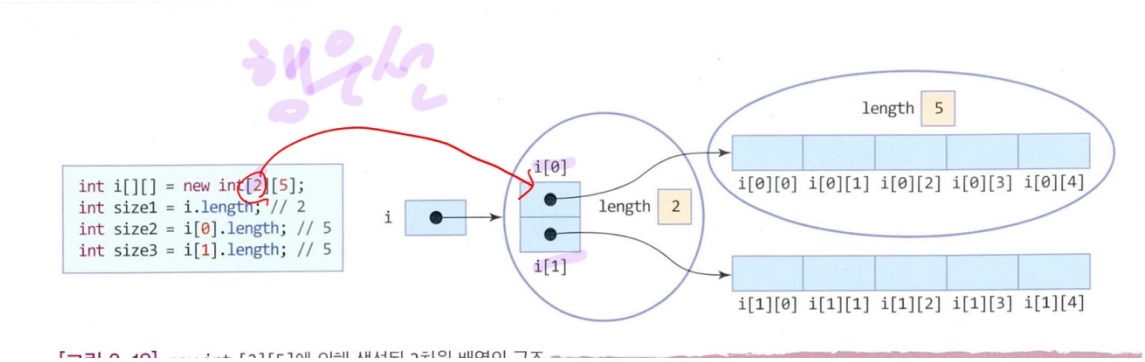
배열

여러 개의 데이터를 하나의 이름으로 저장
접근 속도 증가
같은 type
각각 element를 index로 구분 (index: 0부터 시작)

int I [] = new int[10] 10개의 정수 공간 배열 생성. 배열의 이름은 i
배열에 대한 레퍼런스 선언 int intArray []; 벼열 생성 intArray = new int [5]
레퍼런스 치환으로 두 레퍼런스가 하나의 배열을 공유하는 경우
레퍼런스 즉 배열에 대한 주소만 복사된다.

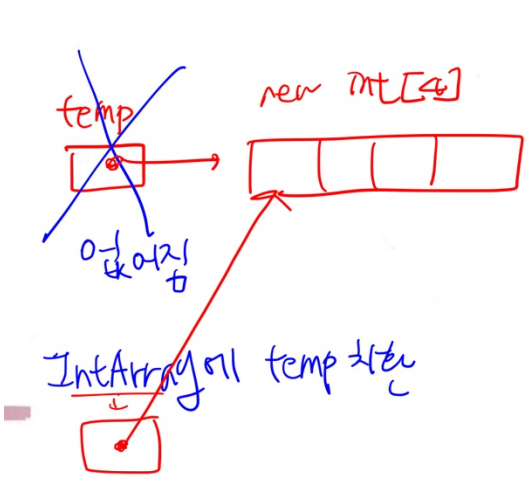
배열의 크기 length필드
배열과 for each문 for(int k : n)//n.length번 반복. k는 n[0],n[1],….로 번갈아 반복

다차원 배열
행우선 저장



i.length ->2
i[0].length->5

메소드에서 배열 리턴
배열에 대한 레퍼런스만 리턴된다.




```
int[] makeArray{
    int temp[] = new int[4];
    return temp;
}

int [] intArray = makeArray();
```

main 메소드

자바의 응용프로그램의 실행은 main()메소드부터 시작한다.

main()메소드는 public 속성이다
자바 가상 기계에 의해 호출되어야 하므로

main()메소드는 static 속성이다.
main()메소드는 자신을 포함하는 클래스의 객체가 생성되기 전에, 처음부터 자바 가상 기계에 의해 호출되므로 static속성으로 선언되어야 한다.

main()메소드의 return타입은 void이다.
아무 값도 return하지 않기 때문이다

main()메소드에는 문자열 배열 string[]이 매개변수로 전달된다.
명령행에 입력된 인자들을 문자열 배열로 만들어 main()메소드에 전달한다.

예외(컴파일 오류가 없는 자바 프로그램이 실행 중에 발생한 오류)

예외 발생-> 자바 플랫폼 인지 -> 응용프로그램 전달 -> 응용프로그램에서 예외처리가 되지 않음-> 응용프로그램 강제 종료

```
try {
    예외가 발생할 가능성이 있는 실행문(try 블록)
}
catch (처리할 예외 타입 선언) {
    예외 처리문(catch 블록)
}
finally {
    예외 발생 여부와 상관없이 무조건 실행되는 문장(finally 블록)
}
```

생략 가능

〈표 3-1〉 자주 발생하는 예외

예외 타입(예외 클래스)	예외 발생 경우	패키지
ArithmeticException	정수를 0으로 나눌 때 발생	java.lang
NullPointerException	null 레퍼런스를 참조할 때 발생	java.lang
ClassCastException	변환할 수 없는 타입으로 객체를 변환할 때 발생	java.lang
OutOfMemoryError	메모리가 부족한 경우 발생	java.lang
ArrayIndexOutOfBoundsException	배열의 범위를 벗어난 접근 시 발생	java.lang
IllegalArgumentException	잘못된 인자 전달 시 발생	java.lang
IOException	입출력 동작 실패 또는 인터럽트 시 발생	java.io
NumberFormatException	문자열이 나타내는 숫자와 일치하지 않는 타입의 숫자로 변환 시 발생	java.lang
InputMismatchException	Scanner 클래스의 nextInt()를 호출하여 정수로 입력받고자 하였지만, 사용자가 'a' 등과 같이 문자를 입력한 경우	java.util

String s;
char c = s.charAt(0) 문자열의 첫 번째 문자

Integer.ParseInt(String)//문자열 정수로 바꿈

함수 사용 이유

프로그램 가독성, 수정/유지보수 쉬움, 중복 제거, 재사용성증가, 구조화 증대

클래스

클래스 객체 모양을 선언한 틀(캡슐화)
메소드(멤버 함수)와 필드(멤버 변수)는 모두 클래스 내에서 정의, 공통으로 사용 가능(선언x, 함수에서 사용)

객체
클래스의 모양대로 생성된 실체(메모리 할당) 객체 내 데이터에 대한 보호, 외부 접근 제한

객체 지향 언어의 특성

캡슐화

캡슐화란 객체를 캡슐로 싸서 내부를 보호하고 볼 수 없게 하는 것으로 객체의 가장 본질적인 특징이다.

상속

객체는 상속이 없고 클래스에 있다. 자바는 클래스 다중상속이 안된다. 자바의 상속은 자식 클래스가 부모 클래스의 속성을 물려받고 기능을 추가하여 확장하는 개념이다. 자바에서 부모클래스를 슈퍼 클래스 라고 부르며 자식 클래스를 서브 클래스 라고 부른다. 상속은 슈퍼 클래스의 필드와 메소드를 물려받아 코드를 재사용함(+새로운 특성 추가)으로써, 코드 작성에 드는 시간과 비용을 줄일 수 있다.

다형성

다형성은 같은 이름의 메소드가 클래스 혹은 객체에 따라 다르게 동작하도록 구현되는 것을 말한다. 메소드 오버라이딩(슈퍼클래스의 메소드를 서브 클래스마다 다르게 구현, 부모함수 매개변수나 타입이 같지만 구현이 다름), 메소드 오버로딩(같은 이름이지만 다르게 작동하는 여러 메소드, 매개변수의 수나 타입이 다름)

객체 지향 언어의 목적

소프트 웨어 생산성 향상

객체, 캡슐화, 상속, 다형성 등 소프트웨어의 재사용을 위한 여러 기법들을 가진 객체 지향 언어 탄생, 재사용하기 쉽고 부분 수정을 통해 소프트웨어를 다시 만드는 부담을 대폭 줄일 수 있다. 소프트웨어의 생산성을 향상시킨다.

실세계에 대한 쉬운 모델링

수학 계산에는 절차 지향이 알맞지만 실세계의 일은 절차나 처리 과정보다는 일과 관련된 많은 물체(객체)들의 상호 작용으로 묘사하는 것이 더 적합하다.

생성자

클래스의 이름과 동일한 메소드를 특별히 생성자라고한다. 생성자는 객체가 생성될 때 자동으로 한 번 호출되는 특별한 메소드이다. 리턴 타입이 없다. 객체가 생성되는 순간에 자동으로 호출되는 메소드

생성자의 특징

생성자의 이름은 클래스의 이름과 동일하다.
생성자는 여러 개 작성(오버로딩)할 수 있다
-> 매개변수의 개수와 타입이 달라야함
생성자는 new를 통해 객체를 생성할 때 한 번만 호출된다
생성자에 리턴 타입을 지정할 수 없다.
생성자의 목적은 객체가 생성될 때, 필요한 초기 작업을 위함이다.

기본 생성자

매개변수와 실행코드가 없어 단순 리턴하는 생성자이다. 디폴트생성자라고도 부른다.
기본 생성자가 자동으로 생성되는 경우: 생성자가 하나도 없는 경우
하나라도 있으면 자동으로 생성하지 않으므로 작성자가 직접 작성해 놓는게 좋다.

Circle pizza; 객체에 대한 레퍼런스 선언 -> 이 선언문으로는 Circle 타입의 객체가 생성되지 않는다.
pizza = new Cirlce() 객체 메모리 할당 및 객체 생성

this 레퍼런스

객체 자신을 가리키는 레퍼런스
this는 컴파일러에 의해 자동 관리되므로 선언 없이 사용가능
static에서 사용 불가(객체 생성 전에 생성되기 때문)
필요성: 객체의 멤버 변수와 메소드 변수의 이름이 같은 경우 this.멤버변수로 객체의 멤버 변수를 가리킬 때 사용, 다른 메소드 호출 시 객체 자신의 레퍼런스를 전달할 때, 메소드가 객체 자신의 레퍼런스를 반환할 때

this()로 클래스 내에서 생성자가 다른 생성자 호출
생성자 내에서만 사용가능, 생성자 코드의 제일 처음에 수행해야함

객체 치환 시 주의할 점

가비지

객체 배열

1. 배열에 대한 레퍼런스 선언 Circle[] c;
2. 레퍼런스 배열 생성 c = new Circle[5];
3. 객체 생성. for(int i = 0 ; i <c.length ; i++){c[i] = new Circle(i);}

메소드 형식

메소드 : 클래스의 멤버 함수
인자 전달: 값에 의한 호출
기본 타입의 값이 전해지는 경우 값은 변하지 않지만
배열과 클래스처럼 레퍼런스 값이 전달될 때는 함수내에서 값(객체의 필드, 배열의 원소)을 변경할 수 있다.

메소드 오버로딩 한 클래스 내에 이름이 같지만 매개변수의 타입니다 개수가 서로 다른 여러 개의 메소드를 중복 작성할 수 있다. 메소드의 리턴 타입니다 접근 지정자는 메소드 오버로딩과 관계없다.

객체의 소멸

자바에는 객체를 생성하는 new는 있지만 객체를 소멸시키는 연산자는 없다.
가비지
참조하는 레퍼런스가 하나도 없는 객체나 배열을 가비지로 판단한다.
가비지 컬렉션
메모리가 일정 크기 이하로 줄어들면 자동으로 가비지를 회수하여 가용 메모리를 늘린다.
가비지 컬렉터가 실행될 때 사용자의 눈에는 프로그램이 중단된 것처럼 보인다. 이런 이유로 자바는 실시간 처리 응용에는 부적합하다.
가비지 컬렉션 강제 요청
System.gc() 요청은 할 수 있지만 가비지 컬렉션은 자바 플랫폼이 전적으로 판단하여 적절한 시점에 작동시킨다.

패키지

자바는 서로 관련 있는 클래스 파일들을 패키지에 저장하여 관리하도록 한다.
패키지는 디렉터리 혹은 폴더와 같은 개념
접근 지정이 패키지와 관련 있다.

멤버 접근 지정

이제 클래스 멤버에 대한 접근 지정을 알아보자. 멤버에 대한 접근 지정자는 〈표 4-1〉과 같으며, private → 디폴트 → protected → public 순으로 공개의 범위가 넓어진다.

〈표 4-1〉 접근 지정자에 따른 멤버 접근

멤버에 접근하는 클래스	멤버의 접근 지정자			
	private	디폴트 접근 지정	protected	public
같은 패키지의 클래스	×	○	○	○
다른 패키지의 클래스	×	×	×	○
접근 가능 영역	클래스 내	동일 패키지	동일 패키지와 자식 클래스	모든 클래스

이 표를 보는 방법을 디폴트 접근 지정 칸을 예로 설명해보자. ○로 표기된 것은 어떤 클래스의 멤버가 디폴트 접근 지정으로 선언된 경우 같은 패키지에 있는 모든 클래스가 이 멤버를 접근할 수 있음을 의미하며, × 의 경우는 다른 패키지의 클래스에서는 접근할 수 없음을 의미한다.

static멤버

static 멤버는 각 객체마다 하나씩 생긴다고 해서 인스턴스 멤버라고 부른다.

〈표 4-2〉 non-static 멤버와 static 멤버의 차이

	non-static 멤버	static 멤버

선언	class Sample {
	  int n;
	  void g() {…}
	}
		class Sample {
		  static int m;
		  static void f() {…}
		}

공간적 특성	멤버는 객체마다 별도 존재 • 인스턴스 멤버라고 부름	멤버는 클래스당 하나 생성 • 멤버는 객체 내부가 아닌 별도의 공간(클래스 코드가 적재되는 메모리)에 생성 • 클래스 멤버라고 부름
시간적 특성	객체 생성 시에 멤버 생성됨 • 객체가 생길 때 멤버도 생성 • 객체 생성 후 멤버 사용 가능 • 객체가 사라지면 멤버도 사라짐	클래스 로딩 시에 멤버 생성 • 객체가 생기기 전에 이미 생성 • 객체가 생기기 전에도 사용 가능 • 객체가 사라져도 멤버는 사라지지 않음 • 멤버는 프로그램이 종료될 때 사라짐
공유의 특성	공유되지 않음 • 멤버는 객체 내에 각각 공간 유지	동일한 클래스의 모든 객체들에 의해 공유됨

this 사용 불가 : 객체 관련(자신의 객체 레퍼런스)

static메소드 non-static멤버 접근 불가

반대는 가능

객체가 생성되지 않은 상황에서도 static메소드는 실행될 수 있기 때문에 non-static메소드 별도 사용 불가

static멤버의 생성 및 활용
1: 객체.static멤버(static멤버를 객체의 멤버로 접근하는 경우)
2: 클래스명.static멤버 (static멤버를 클래스 이름으로 접근)
3: 전역 변수와 전역 함수를 만들 때 활용
->자바에서는 어떤 변수나 함수도 클래스 바깥에 존재할 수 없으며 클래스의 멤버로 존재하여야 한다 전역 변수, 전역함수가 필요한 경우에 대한 해결책이다.

Final

final클래스
->클래스 이름 앞에 final이 사용되면 클래스를 상속받을 수 없음을 지정

final 메소드
->오버라이딩을할 수 없는 메소드임을 선언

final 필드
->필드는 상수가됨 (final int ROWS = 10;)

상속

상속 선언만 하면, 자식 클래스는 부모 클래스에 만들어진 필드와 메소드를 만들지 않고도 만든 것과 같은 효과를 얻는다.
상속은 코드 중복을 제거하여 클래스를 간결하게 구현할 수 있게 한다.
클래스의 간결화 – 멤버의 중복 작성 불필요
클래스 관리 용이 – 클래스들의 계층적 분류
소프트웨어의 생산성 향상 – 클래스 재사용과 확장 용이

자바의 상속선언 extends 사용

서브 클래스는 슈퍼 클래스의 private 멤버를 제외하고 모든 멤버를 접근할 수 있다.

자바 상속의 특징

자바에서는 클래스의 다중 상속을 지원하지 않는다.
자바에서는 상속의 횟수에 제한을 두지 않는다.
자바에서 계층 구조의 최상위에 java.lang.Object클래스가 있다. (상속받는다고 선언하지 않아도 마찬가지)

〈표 5-1〉
슈퍼 클래스 멤버에 대한 접근 지정

슈퍼 클래스 멤버에 접근하는 클래스 종류	슈퍼 클래스 멤버의 접근 지정자			
	private	디폴트	protected	public
같은 패키지에 있는 클래스	×	○	○	○
다른 패키지에 있는 클래스	×	×	×	○
같은 패키지에 있는 서브 클래스	×	○	○	○
다른 패키지에 있는 서브 클래스	×	×	○	○

(○는 접근 가능함을, ×는 접근이 불가함을 뜻함)

상속과 생성자

서브 클래스 객체가 생성될 때 서브 클래스의 생성자와 슈퍼 클래스의 생성자가 모두 실행된다.
실행 순서: 서브 클래스의 생성자와 슈퍼 클래스의 생성자 중 슈퍼클래스가 먼저 실행
호출 순서: 서브 클래스의 생성자가 먼저 호출, 서브 클래스의 생성자는 실행 전 슈퍼 클래스 생성자 호출

서브 클래스에서 슈퍼 클래스 생성자 선택

자동 선택되는 경우

->개발자의 명시적 지시가 없으면, 서브 클래스의 생성자가 기본 생성자이든 매개변수를 가진 것이든, 슈퍼 클래스에 만들어진 기본 생성자가 선택된다.
(꼭 기본 생성자 만들기)

상속 관계에서의 생성자

슈퍼 클래스와 서브 클래스 각각 여러 생성자 작성 가능(오버로딩)
서브 클래스 생성자 작성 원칙
-> 서브 클래스 생성자에서 슈퍼클래스 생성자 하나 선택(super()이용)
만약 선택하지 않으면
슈퍼 클래스의 기본 생성자 컴파일러가 자동으로 선택

super()를 이용하여 명시적으로 슈퍼 클래스의 생성자 선택
매개변수 타입, 개수가 기준
반드시 생성자의 첫 라인에 사용되어야함 즉, this()와 같이 쓰지 못함