1. 자바 시작하기

자바란? 초기의 자바는 가전 제품에 탑재할 프로그래밍 언어로 개발

* 자바의 특징

이식성이 높다

: 서로 다른 실행 환경을 가진 시스템 간에 프로그램을 옮겨 실행할 수 있는 것

객체 지향적이다

: 객체지향언어? 부품에 해당하는 객체들을 먼저 만들고, 이것들읗 하나씩 조립 및 연결해서 전체 프로그램을 완성하는 기법(OOP : Object Oriented Programming)

메모리를 자동으로 관리

: 객체 생성시 자동적으로 메모리 영역을 찾아서 할당하고, 사용이 완료되면 쓰레기 수집기를 실행시켜 자동적으로 사용하지 않는 객체를 제거

멀티스레드를 쉽게 구현

동적 로딩을 지원

1. 변수와 타입

변수 : 하나의 값을 저장할 수 있는 메모리 공간

변수 작성 규칙 : 첫번째 글자는 문자이거나 ‘$’ , ‘\_’ 이어야 하고 숫자로 시작할 수 없다

데이터 타입

|  |  |  |
| --- | --- | --- |
| 값의 종류 | 기본타입 | 메모리크기 |
| 정수 | byte | 1 byte |
| char | 2 byte |
| short | 2 byte |
| Int | 4 byte |
| long | 8 byte |
| 실수 | float | 4 byte |
| double | 8 byte |
| 논리 | boolean | 1byte |

Char 타입

: 자바는 모든 문자를 유니코드로 처리한다. 유니코드는 0~65535범위의 2byte 크기를 가진 정수값이다. 자바는 하나의 유니코드를 저장하기 위해 2byte 크기인 char 타입을 제공한다. 유니코드는 음수가 없기 때문에 char 타입의 변수에는 음수앖을 저장할 수 없다. char타입 변수에 작은 따옴표(‘)로 감싼 문자를 대입하면 해당 문자의 유니코드가 저장된다. Ex) char var1 = ‘A’;

Char변수에 작은 따옴표(‘)로 감싼 문자가 아니라 직접 유니코드 정수값을 저장할 수도 있다. Ex) char c=’A’; int uniCode = c;

Long 타입

: int 타입의 저장범위를 넘어서는 큰 정수는 반드시 정수값 뒤에 l이나 L을 붙여야 한다.

Float 타입

: 자바의 실수 리터럴의 기본타입은 double이기 때문에 float 타입 변수에 저장하려면 리터럴 뒤에 소문자 ‘f’ 나 대문자 ‘F’를 붙여야 한다.

타입 변환

* 자동 타입 변환 : 프로그램 실행 도중에 자동적으로 타입 변환이 일어나는 것

자동타입변환

큰 크기 타입 = 작은 크기 타입

Byte b = 10;

Int i = b; // 자동 타입 변환이 일어난다.

예외) 음수가 저장될 수 있는 byte 타입은 char 타입으로 자동변환시킬 수 없다.

* 강제 타입 변환 : 강제적으로 큰 데이터 타입을 작은 데이터 타입으로 쪼개어서 저장하는 것 ( 캐스팅 : Casting )

강제 타입 변환

작은 크기 타입 = (작은 크기 타입)큰 크기 타입

Int I = 103029770;

Byte b = (byte) I; // 강제 타입 변환 ( 캐스팅 )

* Int 타입은 char 타입으로 자동 변환되지 않기 때문에 강제 타입 변환을 사용해야 한다.

Int I = ‘A’;

Char c = (char) I;

System.out.println(c);

* 실수타입은 정수타입으로 자동 변환되지 않기 때문에 강제 타입 변환을 사용해야 한다. 이 경우 소수점 이하부분은 삭제되고 정수 부분만 저장된다.

Double d = 3.14;

Int I = (int) d; // I 에 3만 저장

1. 연산자

* 연산자 우선순위

1. 최우선 연산자 : () , [] , .
2. 단항 연산자 : !, ~, ++ , -- , (cast)
3. 산술 연산자 : + , - , / , %
4. 쉬프트 연산자 : << , >> , >>>
5. 비교 연산자 : < , > , == , <= , >=
6. 논리 연산자 : & , ^ , && , ||
7. 삼항 연산자 : 조건문 ? 참 : 거짓
8. 대입 연산자 : = , \*= , &=
9. 후위 연산자 : ++ , --
10. 컴마 연산자 : .
11. 조건문과 반복문

(흐름)제어문 = 조건문(if문, switch문) , 반복문(for문, while문, do~while문)

조건문 : 제어문 블록이 실행 완료되었을 때 제어문 블록을 빠져나온다

반복문 : 제어문을 되돌아가 반복실행 한다

Switch문 : 조건에 만족해서 불록 밖으로 빠져나가지 않음. (break 사용)

Switch(변수) { } : 변수 자리에 실수는 사용할 수 없음

Do-while문 : 조건이 만족하지 않아도 최소 한번 실행

Break문 : 대게 if문과 같이 사용되어 if문의 조건식에 따라 for문과 while문을 종료할 때 사용

Public static void main(String[] args) {

While(true) {

Int num = ++1;

If(num==6) {

Break;

}

}

System.out.println(“프로그램종료”);

}

: if문이 참이면 break가 실행되어 while문을 빠져나옴

Continue 문

: 반복문인 for문, while문, do-while문에서만 사용되는데, 블록 내부에서 continue문이 실행되면 증감식, 또는 조건식으로 이동한다.

: continue 문도 대개 if문과 같이 사용 되는데, 특정 조건을 만족하는 경우에 continue문을 실행해서 그 이후의 문장을 실행하지 않고 다음 반복으로 넘어간다.

1. 참조타입

자바의 데이터 타입 : 기본타입 , 참조타입

기본타입 : 정수, 실수, 문자, 논리 리터럴을 저장하는 타입

참조타입 : 객체의 번지를 참조하는 타입 ( 배열, 열거, 클래스, 인터페이스 타입 )

변수 : 스택영역에 저장

객체 : 힙 영역에 저장

메모리 사용 영역

메소드(method) 영역 : 클래스

: JVM이 시작할때 생성되고 모든 스레드가 공유하는 영역이다

힙(heap)영역 : 힙 영역은 객체와 배열이 생성되는 영역

JVM 스택(stack) 영역 : 스택 영역은 각 스레드마다 하나씩 존재하며 스레드가 시작될 때 할당된다.

String 타입

String name1 = ‘윤다솜’;

String name2 = ‘윤다솜’;

: 자바는 문자열 리터럴이 동일하다면 String 객체를 공유하도록 되어 있다. 다음과 같이 name1, name2 변수가 동일한 문자열 리터럴인 “윤다솜”을 참조할 경우 name1과 name2는 동일한 String 객체를 참조하게 된다.

- 배열

: 배열은 같은 타입의 데이터만 저장할 수 있다. 한번 생성된 배열은 길이를 늘리거나 줄일 수 없다.

배열 선언

Int[] arr; int arr[];

: 배열도 객체이므로 힙영역에 생성된다.

Int [] arr = { 0, 1, 2 };

배열 값 수정

Arr[0] = 1;

New 연산자 사용

Int[] arr = new int[] { 1, 2, 3 };

String[] arr = null;

Arr = new String[] {“윤다솜” , “김자바” };

New 연산자로 배열 생성

Int[] arr = new int[5];

배열 길이 얻기

Arr.length;

다차원 배열

Int[][] scores = new int[2][3];

0 1 2

0 (0,0) (0,1) (0,2)

1 (1,0) (1,1) (1,2)

Int[][] arr = { {1,2} , {3,4} };

Int arr = arr[0][0];

1. 클래스

객체(Object) : 물리적으로 존재하거나 추상적으로 생각할 수 있는 것 중에서 자신의 속성을 가지고 다른 것과 식별 가능한 것을 말한다. 객체는 속성과 동작으로 구성되어 있다.

예를들어 사람은 이름, 나이의 속성과 웃다, 걷다등의 동작이 있다. 자바는 이 속성과 동작들을 각각 필드(field)와 메소드(method)라고 부른다.

객체 모델링(Object Modeling) : 현실 세계의 객체를 소프트웨어 객체의 필드와 메소드로 설계하는 것

객체지향 프로그래밍의 특징

1. 캡슐화(Encapsulation) : 객체의 필드, 메소드를 하나로 묶고, 실제 구현 내용을 감추는 것

노출시킬 것인지, 숨길 것인지를 결정하기 위해 접근 제한자를 사용한다.

1. 상속(Inheritance) : 부모 역할의 상위 객체와 자식 역할의 하위 객체가 있는데 상위의 객체는 자기가 가지고 있는 필드와 메소드를 하위 객체에게 물려주어 하위 객체가 사용할 수 있도록 해준다. – 반복된 코드의 중복을 줄여준다.
2. 다형성(Polymorphism) : 다형성은 같은 타입이지만 실행 결과가 다양한 객체를 이용할 수 있는 성질을 말한다. 자바는 다형성을 위해 부모 클래스 또는 인터페이스의 타입 변환을 허용한다. 다형성의 효과로 객체는 부품화가 가능하다.

클래스 : 설계도

인스턴스 : 클래스로부터 만들어진 객체

클래스명 작성 규칙

하나 이상의 문자로 이루어져야 한다, 첫번째 글자는 숫자가 올 수 없다, ‘$’ , ‘\_’외의 특수문자는 사용할 수 없다, 자바 키워드는 사용할 수 없다

클래스의 구성 맴버

필드

: 객체의 데이터가 저장되는 곳

: 필드를 변수라고 부르지 않는다. 변수는 생성자와 메소드 내에서만 사용되고 생성자와 메소드가 실행 종료되면 자동 소멸된다. 하지만 필드는 생성자와 메소드 전체에서 사용되며 객체가 소멸되지 않는 한 객체와 함께 존재한다.

: 초기값이 지정되지 않은 필드들은 객체 생성 시 자동으로 기본 초기값으로 설정된다.

생성자 : 객체 생성시 초기화 역할 담당

생성자를 오버로딩한 경우, 생성자에서 다른 생성자를 호출할 때 this()를 사용한다.

Car(String str1, String str2) {

This.str1 = str1;

This.str2 = str2;

}

Car() {

This(str1 , str2); // Car(String str1, String str2) 생성자 호출

}

메소드 : 객체의 동작에 해당하는 실행 블록 , 메소드는 객체간의 데이터 전달의 수단으로 사용된다. ( boolean의 초기값 : false , 배열, 클래스(String포함), 인터페이스의 초기값 : null)

Return; : 메소드 실행을 강제 종료

메소드 오버로딩 : 클래스 내에 같은 이름의 메소드를 여러 개 선언하는 것

: 매개변수의 타입, 개수, 순서가 달라야 함

인스턴스 맴버 : 객체(인스턴스)를 생성한 후 사용할 수 있는 필드와 메소드 ( 인스턴스 필드, 인스턴스 메소드 )

-정적맴버 , Static

정적맴버

: 객체를 생성하지 않고 사용할 수 있는 필드와 메소드

: 객체마다 가지고 있어야 할 데이터라면 인스턴스 필드로 선언하고, 객체마다 가지고 있을 필요성이 없는 공용적인 데이터라면 정적 필드로 선언하는 것이 좋다 ( ex. PI)

: 인스턴스 필드를 이용해서 실행해야 한다면 인스턴스 메소드로 선언하고, 인스턴스 필드를 사용하지 않는다면 정적 메소드로 선언한다

정적맴버 사용 : 클래스.필드; 클래스.메소드(매개값…); // 클래스 이름으로 접근한다

정적필드는 필드 선언과 동시에 초기값을 주는 것이 보통이다.

EX. Static double pi = 3.14159;

정적 메소드와 블록 선언시 주의점

: 내부에 인스턴스 필드나 인스턴스 메소드를 사용할 수 없다. 또한 this 키워드도 사용이 불가능하다.

싱글톤(Singleton) : 전체 프로그램에서 단 하나의 객체만 만들도록 보장해야 하는 경우 사용

1. 싱글톤을 만들려면 클래스 외부에서 new 연산자로 생성자를 호출 할 수 없도록 생성자에 private 접근 제한자 사용해 막아야 한다.

2. 정적필드를 하나 선언하고 자신의 객체를 생성해 초기화 한다. 정적 필드도 private 접근 제한자를 붙여 외부에서 필드값을 변경하지 못하도록 막는다. 대신 외부에서 호출 할 수 있는 정적 메소드인 getInstance() 를 선언하고 정적필드에서 참조하고 있는 자신의 객체를 리턴해준다.

Public class A {

Private static A a = new A(); // 정적 필드

Private A() { } // 생성자

Static A getInstance() { // 정적 메소드

Return a;

}

}

: 외부에서 객체를 얻는 유일한 방법은 getInstance() 메소드를 호출하는 방법이다.

GetInstance() 메소드는 단 하나의 객체만 리턴하기 때문에 a1 과 a2는 동일한 객체를 참조한다.

A a1 = A.getInstance();

A a2 = A.getInstance();

-Final 필드와 상수

Final 필드 : 초기값이 저장되면 이것이 최종적인 값이 되어서 프로그램 실행 도중에 수정할 수 없는 필드

Final 필드의 초기값을 줄수 있는 방법

1. 필드 선언시 주는 방법
2. 생성자에서 주는 방법

Public class A {

Final String s1 = “korea”;

Final String s2;

String name ;

Public A(String s2, String name) {

This.s2 = s2;

This.name = name;

}

}

-상수(static final)

: 상수란 불변의 값을 뜻한다.

: static final 필드는 객체마다 저장되지 않고, 클래스에만 포함된다. 그리고 한번 초기값이 저장되면 변경 할 수 없다.

: 상수 이름은 모두 대문자로 작성하는 것이 관례이다.

패키지

: 패키지는 클래스를 유일하게 만들어주는 식별자 역할을 한다. 클래스 이름이 동일하더라도 패키지가 다르면 다른 클래스로 인식한다.

Import문

다른 패키지에 속하는 클래스를 사용하는법

1. 패키지와 클래스를 모두 기술하는 것

2. import문을 사용

접근제한자

|  |  |  |
| --- | --- | --- |
| 접근제한 | 적용대상 | 접근할 수 없는 클래스 |
| Public | 클래스, 필드, 생성자, 메소드 | 없음 |
| Protected | 필드, 생성자, 메소드 | 자식 클래스가 아닌 다른 패키지에 소속된 클래스 |
| default | 클래스, 필드, 생성자, 메소드 | 다른 패키지에 소속된 클래스 |
| private | 필드, 생성자, 메소드 | 모든 외부 클래스 |

클래스에 적용 할 수 있는 접근제한자 : public, default

생성자에 적용 할 수 있는 접근제한자 : public, protected, default, private

: (디폴트)기본 생성자의 접근제한자는 클래스의 접근 제한과 동일하다

필드와 메소드에 적용 할 수 있는 접근제한자 : public, protected, default, private

Getter와 Setter 메소드

: 일반적으로 객체 지향 프로그래밍에서 객체의 데이터는 객체 외부에서 직접적으로 접근하는 것을 막는다. 그 이유는 객체의 데이터를 외부에서 마음대로 읽고 변경할 경우 객체의 무결성(결점이 없는 성질)이 깨어질 수 있기 떄문이다.

이러한 문제점을 해결하기 위해 객체지향 프로그래밍에서는 메소드를 통해서 데이터를 변경하는 방법을 선호한다. 외부에서 객체의 데이터를 읽을 때도 메소드를 사용하는 것이 좋다.

7. 상속

상속 개념

: 객체 지향 프로그램에서 부모 클래스의 맴버를 자식클래스에게 물려줄 수 있다.

: 부모클래스 = 상위클래스

: 자식클래스 = 하위클래스 = 파생클래스

: 코드의 중복을 줄여준다

: 부모 클래스에서 private 접근 제한을 갖즌 필드와 메소드는 상속 대상에서 제외된다. 그리고 부모 클래스가 다른 패키지에 존재한다면 default 접근 제한을 갖는 필드와 메소드도 상속 대상에서 제외된다.

- 상속 하는 방법

Class 자식클래스 extends 부모클래스 { }

- 자바는 다중상속을 허용하지 않는다.

- 자식 객체를 생성하면, 부모 객체가 먼저 생성되고 자식 객체가 그 다음에 생성된다.

- 부모 생성자는 자식 생성자의 맨 첫줄에서 호출된다. 예를들어 자식생성자가 명시적으로 선언되지 않았다면 컴파일러는 다음과 같은 기본 생성자를 생성해낸다.

Pulbic AA() {

Super(); // super()는 부모의 기본 생성자를 호출한다

}

만약 직접 자식생성자를 선언하고 명시적으로 부모 생성자를 호출하고 싶다면 다음과 같이 작성하면 된다.

자식클래스( 매개변수 선언, ..) {

Super(매개값 , …);

…

}

: super(매개값, … )는 매개값의 타입과 일치하는 부모 생성자를 호출한다.

- 메소드 재정의

: 메소드 오버라이딩은 상속된 메소드의 내용이 자식 클래스에 맞지 않을 경우, 자식 클래스에서 동일한 메소드를 재정의 하는 것을 말한다.

오버라이딩 규칙

1. 부모의 메소드와 동일한 시그너처( 리턴 타입, 메소드 이름, 매개변수 리스트)를 가져야 한다.

2. 접근 제한을 더 강하게 오버라이딩 할 수 없다.

3. 새로운 예외를 throws 할 수 없다.

- 자식 클래스 내부에서 오버라이딩된 부모 클래스의 메소드를 호출해야 하는 상황이 발생한다면 명시적으로 super 키워드를 붙여서 부모 메소드를 호출할 수 있다.

- final 클래스와 final 메소드

: 클래스와 메소드 선언시에 final 키워드가 지정되면 상속과 관련이 있다.

- 상속할 수 없는 final 클래스

: 클래스를 선언 할 때 final 키워드를 class 앞에 붙이게 되면 이 클래스는 최종적인 클래스이므로 상속할 수 없는 클래스가 된다. 즉 final 클래스는 부모 클래스가 될 수 없어 자식 클래스를 만들 수 없다는 것이다.

- 오버라이딩할 수 없는 final 메소드

: 메소드를 선언할 때 final 키워드를 붙이게 되면 이 메소드는 최종적인 메소드이므로 오버라이딩할 수 없는 메소드가 된다. 즉 부모 클래스를 상속해서 자식 클래스를 선언할 때 부모 클래스에 선언된 final 메소드는 자식 클래스에서 재정의할 수 없다는 것이다.

자동 타입 변환(Promotion)

: 프로그램 실행 도중에 자동적으로 타입 변환이 일어나는 것을 말한다.

부모 클래스 변수 = 자식 클래스 타입;

자동타입변환

: 자동 타입 변환의 개념은 자식은 부모의 특징과 기능을 상속받았기 때문에 부모와 동일하게 취급 될 수 있다는 것이다. ( 고양이는 동물의 특징과 기능을 상속받았다. 그래서 고양이는 동물이다가 성립한다 )

Cat cat = new Cat();

Animal animal = cat;

: 부모 타입으로 자동 타입 변환된 이후에는 부모 클래스에 선언된 필드와 메소드만 접근이 가능하다.