**- 배열-동일 타입 데이터 여러개 저장**

**데이터타입 배열변수명 [] = new 데이터타입[length];**

**배열변수명 [length-1]**

**배열변수명 [length] ---> 실행 error**

**데이터타입 배열변수명 [] = new 데이터타입[0];**

**데이터타입 배열변수명[] [] =**

**new 데이터타입[2차배열length][1차배열length];**

**배열변수명[0]===> 1차배열**

**배열변수명[0].length===> 1차배열**

**배열변수명[0][0]===> 1차배열 1번째요소**

**- ENUM- 정해진 갯수 정해진 값 변수 나열 열거**

**요일타입 – 월 ~ 일**

**enum Week{  
 MON, TUES, WEDNSE, THURS, FRI, SATUR, SUN**

**}**

**now = Calendar.....**

**now.get(Calendar.DAY\_OF\_WEEK)**

**===> 1 – 일**

**2 – 월**

**Week.SUN**

**Week.MON**

**- 클래스**

**객체 표현 틀**

**{적특성 = 멤버변수**

**동적특성 = 기능 = 메소드**

**생성자**

**}**

**- 멤버변수**

**- 메소드**

**- MODIFIER**

**접근권한**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **클래스** | **변수** | **메소드** | **생성자** |
| **public** | **o** | **o** | **o** | **o** |
| **protected** | **x** | **o** | **o** | **o** |
| **no modifier** | **o** | **o** | **o** | **o** |
| **private** | **x** | **o** | **o** | **o** |

**no modifier = (default) 상태 = (package) 상태**

**private – 현재 변수 선언 클래스 내부에서만 사용 가능**

**(default) - 현재 변수 선언 패키지 내부에서만 사용 가능**

**protected - 현재 변수 선언 패키지 내부에서만 사용 가능**

**+ 모든 패키지의 "상속받은 " 클래스 내부 사용 가능**

**public – 모든 클래스 사용 가능**

**사용방법**

**no modifier = (default)=(package) 상태**

**class Member{**

**private String id**

**pw**

**name**

**address**

**phone**

**public ? setId(){}**

**public ? getId(){}  
}**

**class Product{**

**}**

**class Employee{  
}**

**- 객체지향언어 조건**

**1>캡슐화 – encapsulation/정보은닉(information hiding)**

**class = 연관 데이터+기능 묶음 단위**

**private ~ public**

**2> 상속**

**3> 다형성**

**- 리턴타입-메소드 수행 출력값**

**- 매개변수-메소드 수행 입력값-외부전달값**

**int ma(int i){**

**s.o.p(i\*i);**

**return i\*i;**

**}**

**- 기본 매개변수 – 실제값 전달**

**- 참조 매개변수 – 주소값 복사 전달(동일 객체 참조)**

**- main 메소드**

**class A{ }--> A.java--> 컴파일 가능 --> 실행 불가능**

**public static void main(String args[]) 메소드 자동 실행**

**class A{**

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

**}**

**}==> 컴파일 가능, 실행 불가능**

**- 클래스 실행시 입력값 전달 저장 변수 사용**

**args = new String[0];**

**args = new String[1];**

**- 메소드 OVERLOADING**

**- constructor – 생성자**

**class A{.......}**

**class B{**

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

**A a1 = new A();**

**클래스명 객체참조변수명 = new 생성자;**

**1> new 키워드 뒤 호출**

**2> 생성자 이름은 클래스 이름과 같다**

**3> 객체 생성 시점에서 수행 필요 문장 생성자 내부 정의**

**4> 객체 생성(클래스 복사 )시 생성자 호출**

**5> 자바 모든 클래스 자동 기본 생성자 정의**

**클래스명( ){ }**

**6> 기본 생성자 대신에 사용자 정의 생성자 변경 가능**

**= 기본 생성자 자동 삭제**

**7> 생성자 형태**

**[modifier] 클래스이름([매개변수]){  
객체 생성시 수행 문장 정의**

**= heap 멤버변수값 "xxx" 변경 문장**

**}**

**8> 생성자 overloading**

**1개 클래스에 같은 이름 생성자 여러개 정의**

**매개변수 리스트 다르게 정의**

**리턴타입, modifier 제한없다**

**- this 키워드**

**this – 현재자신객체**

**1>멤버변수와 지역변수 이름 같을 때**

**멤버변수와 매개변수 이름 같을 때**

**구분 !!!!)의미 명확 전달**

**this.i**

**2> 같은 클래스 내에서**

**1개의 생성자에서 다른 생성자 호출**

**(생성자 overloading 구현)**

**this ([매개변수값 전달])**

**class A{**

**int i = 10;--> A 멤버변수**

**int ma(int b){ ---> 매개변수**

**int i = 30; --> ma 지역변수**

**s.o.p(i)**

**s.o.p(this.i)**

**for(){ int d = 10}**

**if(){...}**

**return i+b+c;**

**}**

**void mb(){  
 }**

**}**

**Employee e1 = new Employee();**

**Employee e2 = new Employee();**

**class Employee{**

**this...**

**}**

|  |  |
| --- | --- |
| **e2:200**  **e1:100** | **100:[..this......]**  **200:[..this.....]** |

**[modifier]**

**접근권한**

**private < (default) < protected < public**

**활용방법**

**- 멤버변수**

**- 생성자**

**- 메소드**

**class 멀티회사원{**

**String 회사명="멀티" ;===> 메모리 저장 - 1개 변수값 모든 객체 공유**

**String 급여;==> 메모리 저장 - 각 객체마다 개별적 저장**

**}**

**class 캠퍼스학생{**

**String 학교명="캠퍼스" ;**

**String 학생명;**

**}**

**1> 동일값 1개 공유 성격 – 1개 저장**

**2> 객체 생성 이전 메모리 저장**

**– 프로그램 종료시 유지(삭제x)**

**3> static-"정적"**

Person p1 = **new** Person();

p1.name="박민국";

p1.age = 20;

p1.nation="대한민국";

Person p2 = **new** Person();

p2.name="김대한";

p2.age = 30;

p2.nation="대한민국";

|  |  |
| --- | --- |
| **p2:200, 1000**  **p1:100**  **1000**  **p1 ~ p100 - 모든 국적 1개 동일 공유** | **100**  name="박민국";  age = 20;  **200**  name=김대한";  age = 30;  1000  nation="대한민국"; |

공유

공유

1개 생성 객체

공유

**- singleton 객체**

|  |  |
| --- | --- |
|  | **A**  **[static a1= new A(),**  **getInstance() { return a1} ]** |

**A.getIntance()**

**- 상속**

|  |  |
| --- | --- |
| **class 사람 {**  **이름 나이**  **밥먹다 잠자다**  **}** | **학생은 사람 상속받는다**  **사람은 학생의 상위, super , parent 클래스이다**  **학생은 사람의 하위, sub , child 클래스이다**  **학생은 사람이다**  **회사원은 사람이다** |
| **class 학생 extends 사람 {**  **학년 성적**  **공부하다**  **}** | **class 회사원 extends 사람 {**  **급여 직급 부서**  **일하다**  **}** |
| **class 교수{ m(){ x; }}**  **class 학생{ m() {y; } }**  **class 조교 extends 교수, 학생{}** | **클래스 다중상속 금지**  **- 인터페이스 해결** |

**class 자동차{**

**전진(){}**

**후진(){}**

**정지(){}**

**}**

**1. 객체생성=변수-has-a 관계**

**2.상속- IS – A 관계**

**class 운전자 {**

**운전하다(){**

**자동차 car = new자동차()**

**car.전진()**

**A a1 = new A();**

**a1.i**

**}**

**class C extends A{**

**s.o.p(i)  
}**

**A와B클래스 관계에서 A클래스에 포함된 모든 변수및 메소드를 쓸때만 상속을 쓰라는것인가요? A클래스에 많은 부분을 사용할때 쓰지않는 메소드가 하나라도 있다면 객체생성이 더 효율적인건가요?**

**- 클래스 멤버변수**

**1> 상위클래스 private 변수 사용불가**

**2> 상위, 하위클래스 동일명 변수 존재 – 하위클래스 변수 우선적 인식**

**3>**

**this - 자신객체**

**this. 멤버변수명--> 멤버변수와 지역변수, 멤버변수와 매개변수 이름 같을 때 구분**

**this(다른 생성자로 전달값 )**

**super - 상위클래스객체**

**super. 멤버변수명--> 상위 멤버변수와 하위 멤버변수 이름 같을 때 구분**

**super. 메소드명()--> 상위 메소드와 하위 메소드 이름 같을 때 구분**

**//this(다른 생성자로 전달값 )**

**- 메소드 overloading**

**1개의 클래스에 여러개 메소드 정의 가능**

**매개변수 리스트 다르게 정의.**

**리턴타입, modifier 상관없다**

**(기능 확장)**

**- 메소드 overriding 우선시하다**

**상속관계 2개의 클래스에 같은 메소드 정의 가능**

**매개변수 리스트 같아야 한다**

**리턴타입 같아야 한다.**

**modifier 같거나 더 넓어야 한다**

**private < (default) < protected < public**

**===> 상위, 하위클래스 동일명 메소드 존재 – 하위클래스 메소드 우선적 인식**

**(재사용)**

**1>캡슐화와 정보은닉**

**2>상속**

**3>다형성(polymorphism)**

**메소드 overloading**

**메소드 overriding**

**final-수정불가능**

**final 변수- 변수값 수정 불가능= 상수**

**final 메소드- 메소드 내용 수정 불가능**

**메소드 overriding 금지**

**final 클래스 – 상속금지 ,. 모든 메소드 overriding 금지**

**객체생성만 가능**

**abstract – "추상적"**

**abstract 메소드– 선언부만 있고 구현부 없는 메소드앞**

**overriding의무화.**

**abstract 클래스 – abstract 메소드 1개 이상 포함 클래스앞**

**객체생성금지. 상속의무화.**

**abstract class 도형{  
 abstract void 면적();**

**abstract void 둘레();**

**}**

**class 사각형 extends 도형{**

**void 면적(){가로\*세로;}**

**void 둘레(){2\*(가로+세로);}**

**}**

**class 원 extends 도형{**

**void 면적(){반지름 \* 반지름 \* 3.14;}**

**void 둘레(){반지름 \* 2 \* 3.14;}**

**}**

**class 삼각형**

**도형 shape = new 도형();**

**==> 객체생성=클래스 멤버변수와 메소드구현내용 복사 메모리 저장**

**팀장**

**class 사각형 {**

**void 면적(){가로\*세로;}**

**}//A**

**class 원 {**

**void 둘레(){반지름 \* 2 \* 3.14;}**

**}//B**

**class 삼각형{**

**}//C**

**서로 다른 클래스 공통 약속 메소드 스펙**

**abstract class 도형{  
 abstract void 면적();**

**abstract void 둘레();**

**}// 팀장**

**class 사각형 extends 도형{**

**void 면적(){가로\*세로;}**

**void 둘레(){2\*(가로+세로);}**

**}**

**class 원 extends 도형{**

**void 면적(){반지름 \* 반지름 \* 3.14;}**

**void 둘레(){반지름 \* 2 \* 3.14;}**

**}**

**생성자 – 객체 생성 호출**

**new 생성자()**

**객체 최초 생성시 멤버변수값 초기화 문장**

|  |  |  |
| --- | --- | --- |
| **class** A **extends Object 자동정의** {  **int** i;  A(){  **super();자동정의**  i = 10;  System.***out***.println("A 생성자 호출");  }  } | **class** B **extends** A{  **int** j;  B(){  **super();자동정의**  j = 100;  System.***out***.println("B 생성자 호출");  }  } | **public** **class** SuperTest {  **public** **static** **void** main(String[] args) {  B b1 = **new** B();  System.***out***.println  (b1.i);  System.***out***.println  (b1.j);  }  } |
| **b1:300** | **100: Object[ 멤버변수 + 메소드, ]**  **200:A [i = 10]**  **300:B[ j = 100 ]** | |

**B b1 = new B();**

**==> Object() + A() + B()**

**=B객체생성=B 멤버변수와 메소드 메모리 복사**

**+(자동)**

**=A객체생성=A 멤버변수와 메소드 메모리 복사**

**=Object객체생성=.....**

**super – 상위클래스객체 의미 키워드**

**super. 멤버변수명--> 상위 멤버변수와 하위 멤버변수 이름 같을 때 구분**

**super. 메소드명()--> 상위 메소드와 하위 메소드 이름 같을 때 구분**

**super(); 상위 기본 생성자 호출.**

**생성자 내부 첫문장 자동 정의**

**super(매개변수 ); 상위 매개변수 정의 생성자 호출**

**생성자 내부 첫문장 명시적 정의**

**상속 – 멤버변수 / 메소드 / 생성자**

**- 기본형변수 형변환(String ->int 불가능)**

**1> boolean제외**

**2>자동형변환-형변환연산자 생략 가능**

**byte->short->int->long-.float->double**

**char-> int->long-.float->double**

**3>명시적형변환-형변환연산자 명시**

**자동형변환 반대**

**- 클래스간 형변환**

**1> 상속관계**

**2>자동형변환-형변환연산자 생략 가능**

**하위클래스 객체를 상위클래스 변환.**

**상위클래스명 변수명1 = new 하위생성자();  
class A{}**

**class B extends A{}**

**class C extends A{}**

**class D extends C{}**

**class E {}**

**A a1 = new A();->형변환필요x**

**A a2 = new B(); ->자동형변환**

**A a3 = new C(); ->자동형변환**

**A a4 = new D(); ->자동형변환**

**C c1 = new D();->자동형변환**

**A a5 = new E(); -> 자동형변환오류**

**B b1= new A();-> 자동형변환오류**

**3> 자동형변환 이후에 사용 가능한 변수와 메소드는**

**상위클래스 속한 것 사용**

**(단 오버라이딩메소드 하위객체 포함 )**

**class A{ i ; ma() }**

**class B extends A{ j ; mb() }**

**A a2 = new B(); ->자동형변환**

**a2.i ->o**

**a2.j -> x**

**a2.ma()->o**

**a2.mb(); ->x**

**4>명시적형변환-형변환연산자 명시**

**자동형변환되었다가 다시 원래 하위타입 복원**

**==> 하위 변수와 메소드 사용 필요**

**class A{ i ; ma() , mc(){ x; } }**

**class B extends A{ j ; mb() mc(){ x+y;} }**

**A a2 = new B(); ->자동형변환**

**a2.mc();===> B (A 포함, B overriding메소드)**

**a2.i -> A**

**a2.ma()->A 포함, non-overriding메소드**

**B b2 = (B)a2; --> 명시적형변환**

**a2.i ->o**

**b2.j -> o**

**a2.ma()->o**

**b2.mb(); ->o**

**A(부모)클래스형을 B(자식)클래스생성자로 생성할때**

**B클래스에서 따로 선언된 메소드는 사용을 못한다는건**

**이해했습니다.**

**근데 오버라이딩된 메소드는 B클래스 메소드를 따르는데**

**굳이 형변환을 해서 생성하는 이유가 따로있나요?**

**메모리 사용이 더 효율적인가요?**

**- 객체들 타입 "통일" 경우**

**1> 서로 다른 객체들을 1개 배열 저장**

**배열 – A, B, C 객체들=> 객체들 타입 "통일" 경우**

**A ar[] = new A[5];**

**ar[0] = new A();**

**ar[1] = new B()**

**ar[2] = new C()**

**2>**

**class Sample{**

**void test( A a1 ){///.}**

**}**

**Sample s = new Sample();**

**s.test(new A())**

**s.test(new B())**

**s.test(new C())**

**3> 멤버변수 여러 객체 타입 통일**

**class 자동차{**

**타이어 mytire = new 금호타이어();**

**}**

**class 타이어{}**

**class 한국타이어 extends 타이어 {}**

**class 금호타이어 extends 타이어 {}**

Parent p1 = **new** Parent(); // 형변환필요x

Child c1 = **new** Child(); // 형변환필요x

Parent p2 = **new** Child(); // 자동형변환가능

Child c2 = (Child)p2; // 명시적형변환 가능

//Child c3 = new Parent();// 자동형변환불가능. 컴파일오류

//Child c3 = (Child)new Parent(); //명시적형변환불가능. 컴파일ok. 실행오류

|  |  |
| --- | --- |
| **p1:100** | **100:Object[....]**  **200:Parent[....]** |
| **c1:300** | **100:Object[....]**  **200:Parent[....]**  **300:Child[.....]** |
| **p2:500** | **400:Object[....]**  **500:Parent[....]**  **600:Child[su2 mc().]** |
| **c2:600** |  |
| **Child c3:error** | **700:Object[....]**  **800:Parent[....]** |
|  |  |