|  |  |
| --- | --- |
| 1 | 다음 설명 중 틀린 것은?  (1)자바의 기본 자료형(숫자,문자,논리값) 들 중 같은 자료형 간에는 형변환이 가능하다.  (2)객체 지향 언어인 자바는 사실상 자료형이 4개이다.  (3)객체 자료형도 자료형이다.  (4) 자바의 기본 자료형은 같은 종류로 분류가 가능하지만(예 숫자형과 문자형), 객체 자료형은 개발자  가 정의한 자료형 이기에, 같은 종류로 분류할 수 있는 방법이 없어 객체간 형 변환은 불가능하다. |
| 2 | 다음 설명 중 틀린 것을 모두 고르면?  (1)상속을 이용하면 객체 자료형도 같은 종류로 분류될 수 있다.  (2)상속관계에 있는 객체의 자료형은 부모 클래스의 자료형을 따라간다. 예를 들어 자식 코뿔소는 코  뿔소 형이다.  (3)자바의 기본 자료형 간에는 형 변환이 가능 하듯, 상속관계에 있는 클래스들도 서로 같은 종류의  데이터 타입으로 간주되어 상,하위 객체간 형변환이 가능하다.  (4)자식 자료형에서 부모 자료형으로의 형 변환을 upcasting이라 한다.    (5)부모 자료형에서 자식 자료형으로의 형변환을 demotion이라 한다. |
| 3 | 다음 설명 중 맞는 것은?    class Bird{  String name="난 그냥새";  }    class Duck extends Bird{  String name="난 오리";  }    class Test{  public static void main(String[] args){  Bird b=new Bird(); (A)  Duck d=new Duck(); (B)  Bird bird=d; (C)  Duck d2=(Duck)bird; (D)  System.out.println(d.name);(E)  }  }    (1) (A)에서 메모리에 올라가는 인스턴스는 Duck 과 Bird 이다.  (2) (B)에서 Duck 클래스가 올라가기 전에 Bird 클래스가 먼저 메모리에 올라가게 되어 있다.  (3) (C)에서 dk 를 레퍼런스할 수 있는 자료형은 Duck 형임에도 불구하고 Bird 형으로 레퍼런스하고  있으므로 오류가 날 것이다.  (4) (D)는 부모자료형인 bird를 자식자료형인 Duck으로 강제 형변환을 시도한 것이며 이러한 형변환을  업케스팅이라 한다.  (5) (E)에서의 출력결과는 "난 그냥새"가 출력된다. |
| 4 | 다음 설명 중 틀린 것은 ?  (1)상속관계에 있는 클래스들간에는 같은 종류의 자료형으로 본다.    (2)자바의 기본 자료형간 형 변환에서는 데이터의 손실 발생 여부를 고려해야 하지만, 객체 자료형간의 형변  환에서는 데이터의 크기와 데이터의 손실 발생 여부가 고려대상이 레퍼런스가 가리킬 수 있는 범위의  문제이므로, 데이터의 기본 자료형 간의 형변환과는 차이가 있다.    (3)자식 클래스로 내려갈 수록 사용할 수 있는 변수와 메서드 량은 점점 풍부해진다. 따라서 자식 클래스를  부모 클래스보다 큰 자료형으로 볼 수 있다.    (4)클래스간 형변환에 있어 자료형의 크고 작음의 판단은 기본 자료형들과는 틀리므로, 해당 객체가 가리킬  수 있는 객체의 범위가 넓을 수록 큰 자료형으로 본다, 따라서 부모 클래스의 자료형이 자식보다 크다고  표현한다. |
| 5 | 다음 중 틀린 설명을 고르면? 2, 5, 7  public class Bird {  String name="난 새";  public void fly() {  System.out.println("새가 난다");  }  }  public class Duck extends Bird{  String name="난 오리";  public void fly() {  System.out.println("오리가 난다.");  }  public static void main(String[] args) {  Bird b1 =new Bird();//(A)  Duck d1 = (Duck)b1;//(B)    Duck d2=new Duck();//(C)  Bird b2=(Bird)d2;//(D)  System.out.println(b2);//(E)  System.out.println(d2);//F)    System.out.println(b2.name);//(G)  b2.fly();//(H)  }  }  (1)(A)에서 메모리에 올라가는 인스턴스는 1개이다.    (2)(B)에서 Bird형 객체를 Duck 형으로 다운 케스팅 하고 있으므로 아무런 문제가 없다  ->3번의 이유로 컴파일 에러 발생..  (3)형변환은 메모리에 올라온 인스턴스간에 행해지는데, (B)에서는 아직 Duck 의 인스턴스가 올라온 적이  없으므로 컴파일 에러가 발생할 것이다.    (4)(C)에서 메모리에 올라가는 인스턴스가 2개이다.  (5)(E)에서 출력되는 주소값은 Bird 인스턴스의 주소값이다.  ->Bird가 아니라 Duck의 주소값을 담고 있다  (6)(D)에서 Duck의 인스턴스가 Bird의 인스턴스로 변환되는게 아니라, 레퍼런스 변수의 자료형이 변경된 것이므로 , 객체간 형변환에 의해 Duck의 인스턴스가 영향을 받지는 않는다.  (7)(G)에서 출력되는 변수는 Duck의 것이 출력된다.  ->Duck이 아니라 Bird의 것이 출력  (8)(H)에서 호출되는 메서드는 Bird의 메서드이다  (9)생성되는 인스턴스가 자식을 포함할 경우 메모리에는 부모와 자식의 인스턴스가 동시에 생성되어, 자식의 인스턴스 영역에서 부모의 인스턴스 영역을 접근까지 할 수 있으므로 개발자는 확장된 인스턴스를 보유한 셈이 된다, 이때 개발자는 확장된 인스턴스내에서 어떤 데이터나 메서드를 호출할지 자료형을 명시하여 사용을 선택할 수 있다.(오버라이딩 된 것은 자식 것을 우선) |