상속

|  |  |
| --- | --- |
| 1 | 다음 설명 중 틀린 것은?  (1) 자바에서는 객체형도 자료형으로 본다.  (2) ~~자바의 기본 자료형은, 비슷한 자료형 간 서로 형변환이 가능하지만, 객체 자료형은 서로 비슷한 자료형이라~~  ~~는 개념이 없으므로 형변환이 지원되지 않는다.~~  (3) 자바에서 상속 관계의 객체들 간은 같은 종류의 자료형으로 본다  (4) extends 로 상속을 명시할 경우 부모와 자식은 같은 종류의 자료형으로 인정되며, 이때의 extends는 is a 로  해석될 수 있다.  (5)객체지향 언어에서 상속이 갖는 의미는 크게 두가지인데, 하나는 자식의 메모리 영역이 부모까지 확장될 수 있다는 것이고 또 하나는 상속 관계의 객체들은 같은 종류의 자료형으로 보고 서로 형변환이 가능하다는 것이다 |
| 2 | 다음 설명 중 틀린 것은?  (1) ~~상속관계의 계보에서 부모 자료형으로 올라갈수록 부모는 자식보다 보유한 변수와 메서드 수가 더 많아지게~~  ~~된다~~  (2) ~~기본 자료형 간 형변환에서는 각 자료형이 수용할 수 있는 메모리 용량을 기준으로 서로 크고 작음을 판단하~~  ~~므로 객체 자료형의 경우도 각 객체 자료형 간 메모리 용량으로 서로 크고 작음을 판단한다.~~  (3) 객체 자료형에서의 크고 작음의 판단은 용량이 아닌, 객체가 가리킬 수 있는 자료형의 범위로 판단해야 한다  (4) Bird 형이 Duck 형보다 가리킬 수 있는 자료형의 범위가 더 넓다고 보아야 한다  (5)클래스 정의 시 개발자가 상속을 명시하지 않아도 자바의 최상위 클래스인 Object 클래스를 상속받게 되는데,  Object 클래스는 실질적으로 보유한 메서드 수가 얼마 되지 않지만, 대신 가리킬 수 있는 자료형의 범위는  가장 넓으므로 최상위 객체로 인정받는다  (6)객체가 보유한 속성이나 메서드 수로 객체 자료형의 크고 작음을 판단해서는 안되며, 가리킬 수 있는 범위가  클수록 큰 자료형으로 본다 |
| 3 | 상속 관계가 아래와 같을때, 이 두 객체가 어떤 관계에 있는지를 서술하세요   |  |  | | --- | --- | | 상속관계 | 관계 서술 | | Duck extends Bird | Duck ( is a ) Bird | | Orange extends Fruit | Orange ( is a ) Fruit | | Pizza extends Foot | Pizza ( is a ) Food | |
| 4 | 다음 설명 중 올바른 것을 모두 고르면?  **class** Duck **extends** Bird{    }  (1) extends를 is a 로 해석해도 문장에 무리가 없다.  (2) ~~클래스 선언을 Bird extends Duck 으로 순서를 바꾸어도 무리가 없다.~~  (3) 이 클래스가 올바로 컴파일 되려면 Bird 클래스가 먼저 컴파일 되어 있어야 할 것이다.  (4) Bird 를 부모 클래스라 하며 ~~자바에서는 Parent 클래스 라고 표현한다.~~  (5) Duck 을 자식 클래스라 하며 ~~자바에서는 Child 클래스 라고 표현한다.~~ |
| 5 | 주어진 자바코드에 의해 heap 영역에 생성되는 인스턴스의 개념을 우측영역에 그림으로 표현해보세요  ( 단 , Object 클래스도 존재한다는 전제하에 )   |  |  | | --- | --- | | 자바코드 | Heap 영역 개념도 | | **public** **class** Fish {  String name="난 물고기";  }  **public** **class** Shark **extends** Fish{  String name="난 상어";  }  **public** **class** UseAnimal {  **public** **static** **void** main(String[] args) {  Fish fish = **new** Fish(); //(가)    Fish s = **new** Shark(); //(나)  }  } |  | |
| 6 | 다음 설명 중 올바르지 않은 것은?  (1) 상속은 extends 키워드로 구현하며 extends 키워드를 is a 로 바꾸어 이해해도 된다.  (2) ~~상속관계에 있는 클래스 중 부모의 인스턴스를 생성하면 자식의 인스턴스도 함께 메모리에 생성된다.~~  (3) ~~상속관계에 있는 클래스 중 자식의 인스턴스를 생성할때 부모 클래스의 인스턴스도 함께 메모리에 생성된다.~~  (4) **상속관계에 있는 클래스 중 자식의 인스턴스를 생성하면 부모 클래스의 인스턴스가 생성되는 것이 아니라,**  **부모가 보유한 변수와 메서드만 자식의 인스턴스 영역으로 들어오게 된다.**  (5) ~~자식의 인스턴스 영역으로 들어온 부모의 변수와 메서드를 접근하려면, 레퍼런스 변수의 자료형을 부모형으~~  ~~으로 형변환하면 된다~~ (자식 것으로도 가능)  (6) 상속 관계에서 자식의 인스턴스 생성 시 부모의 인스턴스는 생성되지는 않지만, 부모가 보유한 멤버 변수와  메서드는 자식의 인스턴스 영역으로 들어오게 되는데, 이때 자식은 원하는 시점에 형변환을 통해 부모의 멤버  변수와 메서드를 접근할 수 **있다**. |
| 7 | 다음 설명 중 올바르지 않은 것은?  (1) 상속을 이용하여 객체가 일반적으로 갖는 보편적이고 공통성 있는 특징, 기능들을 부모 클래스에 정의하면  자식 클래스는 동일한 코드를 중복정의 하지 않아도 되므로 개발시 코드의 재사용성이 높아진다.  (2) **현실에서도 다중 상속이 존재하지 않듯, OOP언어에서도 부모를 동시에 둘 이상 두는 다중 상속은 허용되지**  **않는다.**  (3) ~~부모클래스가 멤버변수를 private 으로 접근 제한을 둘 경우, 자식의 인스턴스 생성시 이 부모의 멤버변수는~~  ~~자식의 인스턴스 영역으로 들어오지 않는다.~~ (접근제한은 있지만 메모리에 존재)  (4) 결국 상속관계에 있는 부모 자식 객체 중, 인스턴스가 생성되는 쪽은 자식 측이다.  (5) **자식이 부모가 보유한 메서드를 동일하게 재정의 할 경우 오라이딩이라 하며, 이때는 자식의 메서드 호출이**  **우선 순위를 갖는다** |
| 8 | 다음 중 틀린 것은?  (1) 자바는 같은 종류의 자료형 간 서로 형변환이 가능하다.  (2) **기본 자료형을 대상으로** 작은자료형에서 큰자료형으로의 형변환을 promotion 이라 하고, 그 반대의 경우를  demotion이라 한다  (3) 상속관계에 있는 클래스들간에는 같은 종류의 자료형으로 본다.  (4) 객체 자료형을 대상으로 상위 자료형이 하위 자료형으로 형변환 되어질때는 cast연산자로 강제 형변환 임을  명시 해야 한다.  (5) 하위 자료형에서 상위 자료형으로의 형변환을 UP Casting이라 하며, 그 반대의 경우를 Down Casting이라  한다. |
| 9 | 다음의 설명 중 맞는 것은?  **public** **class** Bird {  String name="난 그냥새";    **public** **void** fly() {  System.***out***.println("새가 날아요");  }  }  **public** **class** Duck **extends** Bird{  String name="난 도날드";    **public** **void** fly() {  System.***out***.println("오리가 날아요");  }  }  **public** **class** UseAnimal {  **public** **static** **void** main(String[] args) {  Duck d = **new** Duck(); //(가)  Bird b=d; //(나)  Duck c=b; //(다)    Bird k = **new** Bird(); //(라)  Duck m = (Duck)k;//(마)  }  }  (1) (가) 라인에서 Duck 의 인스턴스 생성 시, 자동으로 부모의 멤버변수와 메서드가 자식의 인스턴스에 포함된다  (2) ~~(나)에서 d는 Duck 형이므로 Bird형인 b 보다 데이터의 크기가 크기 때문에 컴파일 에러가 발생한다~~  (3) ~~(다)에서 자동 형변환이 발생하므로 아무런 문제가 없다~~  (4) ~~(라)에서 Bird 뿐 아니라 Duck의 인스턴스도 함께 메모리에 생성된다~~  **~~(5) (마)에서는 아무런 문제가 없이 형변환이 수행된다~~**  (6) (라)에 의해 메모리에 생성되는 인스턴스는 Bird형이며, Duck의 인스턴스는 생성되지 않는다  (7) **(마)에서 k가 가리키는 인스턴스는 Bird의 인스턴스 이므로, 즉 Duck인스턴스를 보유하지 않고 있으므로**  **형변환은 불가능하다** |
| 10 | 다음 설명 중 틀린 것은?  **public** **class** Car {  String name="난 그냥 차";    **public** **void** drive() {  System.**out**.print("차가 갑니다");  }  }  **public** **class** Bus **extends** Car{  String name="난 버스";    **public** **void** drive() {  System.**out**.print("버스가 갑니다");  }  }  **public** **class** UseBus {  **public** **static** **void** main(String[] args) {  Car car = **new** Bus(); //(가)  System.**out**.println(car.name);//(나)  car.drive(); //(다)  }  }  (1) ~~(가)에서 car 변수가 가리키고 있는 인스턴스는 Car 의 인스턴스이다~~    (2) (나)에서의 출력 결과는 “난 그냥 차” 이다  (3) ~~(다)에서 호출되는 메서드는 Car형의 메서드 이므로 “차가 갑니다” 가 출력된다~~  (4) **(다)에서 비록 Car 형 변수로 메서드를 호출하였다 하여도, Car의 drive() 메서드가 아닌 자식인 Bus의 driver()**  **를 호출하게 되어 있다**  **(5) (다)와 같은 현상을 가리켜 OOP에서는 다형성(polymorphism)이라 한다 (하나의 객체가 여러 모습을 띈다)** |
| 11 | 다음 설명 중 맞는 것은?  **class** Plant {  String name = "난 식물";  **public** **void** grow() {  System.***out***.println("식물이 자랍니다.");  }  }  **class** Flower **extends** Plant {  String name="난 꽃";  **public** **void** grow(){  System.***out***.println("꽃이 자랍니다.");  }  }  **class** UsePlant {  **public** **static** **void** main(String[] args){  Plant p1=**new** Plant(); (A)  System.out.println(p1.name); (B)    Plant p2=**new** Flower(); (C)  System.out.println(p2.name); (D)    p2.grow(); (E)  }  }  (1) ~~(A)라인에 의해 Plant의 인스턴스뿐 아니라 Flower의 인스턴스도 메모리에 올라간다.~~  (2) (~~B)에서 출력되는 결과는 "난 꽃" 이다.~~  (3) (C)라인에 의해 메모리에 올라가는 인스턴스는 Flower 뿐이며,Plant의 name과 grow() 메서드가 Flower  인스턴스에 포함된다  (4) (~~D)에서 출력되는 결과는 "난 꽃" 이다~~.  (5) (E)의 메서드 호출에 출력되는 결과는 "꽃이 자랍니다." 이다.. |
| 12 | 다음의 설명 중 맞는 것을 고르면?  **public** **class** Bird {  **public** **void** fly() {  System.***out***.println("새가 날아요");  }  }  **public** **class** Sparrow **extends** Bird{  **public** **void** fly() {  System.***out***.println("참새가 날아요");  }  }  **public** **class** Duck **extends** Bird{  **public** **void** fly() {  System.***out***.println("오리가 날아요");  }  }  **public** **class** UseBird {  **public** **static** **void** main(String[] args) {  Bird bird = **new** Bird();  bird.fly(); //(가)  bird=**new** Sparrow();  bird.fly(); //(나)  bird=**new** Duck();  bird.fly(); //(다)  Bird b=**new** Bird();  Duck d=(Duck)b; (라)  }  }  (1) ~~(가)에서 bird변수의 자료형이 Bird 형 이지만, 자식클래스들에 의해 Overring 되었으므로 실행 결과는 “오리가~~  ~~날아요” 이다~~  (2) ~~(나)에서 bird 변수의 자료형이 Bird 형 이므로 실행 결과는 “새가 날아요” 이다.~~  (3) (다)에서 bird 변수의 자료형이 Bird 형 이지만, 자식의 오버라이드 된 메서드 우선 호출의 원칙 상 “오리가 날아요” 가 출력된다.  (4) **(라)에서 형변환이 되려면, 관련 인스턴스들이 생성되어야 하지만 Duck형 인스턴스는 생성된 적이 없으므로**  **실행시 형변환 에러가 발생한다.**  (5) **bird 변수는 자료형이 Bird형 이지만 (가)에서는 그냥 새로 동작했고 (나)에서는 참새로 (다)에서는 오리로**  **동작했으므로, 하나의 자료형이 여러 모습으로 동작했다고 볼 수 있으므로 이러한 기법을 OOP언어에서는**  **Polymorphism (다형성)이라 한다.** |