|  |  |
| --- | --- |
| 1 | 다음은 무엇에 대한 설명인가?  과거에 정의해 놓은 클래스를 재 사용할 수 있는 OOP언어의 객체 정의 기법  답 : ( 상속 ) |
| 2 | 다음 설명 중 맞는 것은? 2, 4  (1)상속이란 부모 클래스의 모든 코드가 자식 클래스로 물리적으로 덮어 써지는 기능이다.(물리적 x)  (2)상속의 형식은 ‘자식클래스 extends 부모클래스’ 이다  (3)코드를 재사용하기 위함이 목적이므로 extends 의 대상이 되는 클래스를 여러개 이상 둘 수 있다.(한번만 가능)  (4)부모 클래스의 코드가 자식 클래스에 덮어쓰기가 되는게 아니라, 자식과 부모가 각각 인스턴스를  생성하되, 자식의 인스턴스가 부모의 인스턴스를 내 것 처럼 접근이 가능한 기법이 상속이다.  따라서 부모의 재산을 “물려받는다” 라는 표현보다는 부모의 재산에 “접근할 수 있다”는 표현이 더  적절하다 |
| 3 | 다음 설명 중 맞는 것은? 6    class Bird{  String name="난 그냥새";  public void fly(){  System.out.println(“하늘을 날아요~”);  };  }    class Duck extends Bird{  String name="난 오리";  public void eat(){  System.out.println(“먹이를 먹어요”);  }  }    class Test{  public static void main(String[] args){  Bird b=new Bird(); (A)  Duck d=new Duck(); (B)  d.fly(); (C)  b.eat(); (D)  System.out.println(d.name);(E)  }  }    (1)(A)에서 메모리에 올라가는 인스턴스는 Duck 과 Bird 이다.  (2)(B)에서 메모리에 올라가는 인스턴스는 Duck의 인스턴스 1개이다  (3)오리 클래스에는 fly() 메서드가 작성되어 있지 않으므로, (C)에서 오류가 발생한다.  (4)상속관계에 있는 객체는 서로 데이터를 공유하고 있으므로 (D)는 아무런 문제가 없다.  (5)부모클래스 Bird에 name이란 멤버변수가 존재하고 자식클래스인 Duck 에도 name 이란 멤버변수  가 존재하므로 Duck의 name 변수는 Bird의 name 변수로 대체되어 버린다.  (6)(E)에서의 출력결과는 "난 오리"가 출력된다. |
| 4 | 다음 설명 중 틀린 것은? 2, 4 ,5  (1)하나의 클래스 내에 메서드 명을 중복 정의 할 수 있는 기법을 오버로딩이라 한다.  (2)부모 클래스에 이미 정의된 메서드 명을 자식 클래스에서 중복 정의하면 에러가 발생한다.  (3)부모 클래스의 메서드를 오버라이딩 하려면 메서드 명 뿐만 아니라 인수의 갯수 자료형까지 똑같아  야 한다.  (4)오버라이딩은 코드의 안정성을 해하므로, 가급적 사용해서는 안된다  (5)부모 클래스의 기능을 자식에 직접 사용하기에 무리가 있거나 , 상황에 맞지 않을 경우 자식은  부모 클래스 메서드를 자신에 맞게 재정의할 수 있는데, 이러한 메서드 재정의 기법을 오버로딩이  라 한다. |
| 5 | 다음 설명 중 맞는 것은? 4, 5  //부모 객체 자료형이 자식 객체 자료형보다 크다고 생각한다….  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(); //(F)  }  }  (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 뿐이다.  (4)(D)에서 출력되는 결과는 "난 꽃" 이다.  (5)(F)의 메서드 호출에 출력되는 결과는 "꽃이 자랍니다." 이다.  상속 관계에서 자식이 오버라이드한 메서드가 있다면 부모 메서드를 호출하더라도 자식 메서드가 적용된다 |
| 6 | 다음 설명 중 맞는 것을 모두 고르면? 3, 4  pakcage aaa;  class Fish{  private String location;  int weight=30;  protected int length=100;  public Fish(String location){  this.location=location;  }    public void swim(){  System.out.println("물고기가 헤엄을 칩니다");  }  }  pakcage bbb;  import aaa.Fish;  class Shark extends Fish{  public void swim(){  System.out.println("무게가 “+length+” 인 상어가 헤엄을 칩니다");  }    }  package use;  import bbb.shark;  class UseShark{  public static void main(String[] args){  Shark s=new Shark(); //(A)  System.out.println(s.location); //(B)  s.swim(); //(C)  )  }  }  //상속 관계에서는 생성자를 명시할 때 오버라이딩을 확실히 하자  (1)(A)라인에는 아무런 문제가 없다.  (2)(B)라인에는 문법적으로 아무런 문제가 없다.  (3)(C)라인에는 문법적으로 아무런 문제가 없다.  (4)Fish는 개발자가 생성자를 정의했으므로, 컴파일러에 의한 디폴트 생성자는 존재하지 않는다 |
| 7 | 다음 설명 중 틀린 것은? 3  public class Dog {  public void bark() {  System.out.println("멍멍");  }  public static void main(String[] args) {  Dog d = new Dog();  d.bark(); //(A)  d.test();//(B)  System.out.println(d.toString()); //(C)  }  }  (1)(A)는 Dog 클래스에 존재하는 메서드 호출이므로 아무런 문제가 없다.  (2)(B)는 Dog 클래스에 존재하지 않는 메서드 호출이므로 컴파일 에러가 발생한다  (3)(C)는 Dog 클래스에 존재하지 않는 메서드 호출이므로 컴파일 에러가 발생한다  ->objectManager라는 최상위 객체가 이미 toString()을 지정했으므로 오류가 나지 않는다  (4)이 세상에 부모없는 객체는 존재할 수 없다.즉 기원이 존재하지 않는 객체는 존재할 수 없다.  ->개발자가 extends로 상속을 따로 하지 않아도 java 프로그램 내에서 objectManager라는 부모 객체로 상속을 해준다.. 그러므로 모든 객체는 부모 객체를 가진다 |
| 8 | 다음 설명 중 올바르지 않은 것은? 4  (1)자식(sub)클래스는 상속을 통하여 부모(super)클래스의 멤버 변수 및 메서드를 물려받을 수 있다.  (2)상속을 이용하면, 객체가 일반적으로 갖는 공통성 있는 특징,기능들을 부모클래스에 정의하고,자식 클래스  에서는 동일한 코드를 중복 정의 하지 않아도 되므로, 개발시 소스의 재사용성이 높아진다.  (3)현실에서도 다중 상속이 존재하지 않듯, OOP언어에서도 다중 상속은 허용되지 않는다.  (4)부모 클래스에서 멤버변수에 private으로 접근 제한을 둘 경우, 이 멤버변수 조차도 자식이 접근할 수 있다.  ->private는 자신 클래스 내에서만 접근할 수 있다  (5)부모의 생성자는 자식에게 상속되지 않는다.  ->생성자까지 상속되면 부모와 동일해지므로…  (6) (5)의 이유는 생성자의 목적이 해당 객체가 태어날 때 그 객체 만이 갖는 개성과 스타일 등을 특징 지을 수 있는 유일한 초기화 작업을 지원하기 위함이므로, 생성자를 자식 객체가 물려받는다 하더라도, 사용할 의미가 없다고 보아야 한다. |