**클래스 기초**

|  |  |
| --- | --- |
| 1 | 아래 클래스에 대한 설명 중 틀린 것을 모두 고르면?  **class** Cat{  String eyeColor="brown";  **int** age=3;  **char** gender='여';  }  (1) Cat 클래스는 개발자가 만들어낸 자료형이며 이를 사용자 정의 자료형이라 한다.  (2) 자바 언어는 기본자료형 이외에도 객체자료형을 지원하므로, 자바의 자료형은 결국 4가지인셈이다.  (3)이 클래스를 디자인한 개발자는 “브라운색의 눈을 가진 3살짜리 암컷 고양이”를 표현했다.  (4)이 클래스를 작성한 개발자는 현재 클래스에서 응용프로그램이 시작될 수 있도록 의도하지 않았다.  (5) ~~고양이가 보유한 3개의 상태 값들을 사용하려면 고양이의 탄생 없이도 사용이 가능하다.~~  ~~즉 고양이를 메모리에 올리지 않고도 eyeColor, age, gender 변수를 사용할 수 있다.~~  (6) 고양이가 존재해야 상태도 존재하는 것이므로, 고양이 인스턴스를 먼저 메모리에 올리고 나서야,  내부에 존재하는 3개의 상태값들도 사용이 가능할 것이다. |
| 2 | 다음 중 틀린 것은?  **class** Member { //A  **int** age=23;    **public** **void** talk(){  }  }  **public** **class** UseMember{  **public** **static** **void** main(String[] args){  age=7; //B  Member m = **new** Member();//C  System.*out*.println(m.age); //D  }  }  (1) A는 개발자가 현실의 "회원"이라는 개념을 Member라는 클래스로 정의한 것이고 이때 정의된 자료형  은 Member 형이다.  (2) (1)과 같이 개발자가 정의하는 자료형을 사용자 정의 자료형이라고 한다.    (3) Member 클래스에는 실행부가 없으므로 java.exe로 실행시킬 수 없으며 프로그램의 시작은 다른  클래스가 담당해야 할 것이다.  (4) ~~B에서 Member 클래스의 age 변수의 값이 23에서 7로 변경된다.~~  (5) C에서와 같이 m변수 앞에 Member를 명시해야 하는 이유는, 자바에서는 Member가 자료형으로  인정되기 때문이다  (6) C와 같이 객체 자료형을 메모리에 올리는 과정을 인스턴스화라 하고, 이 과정에 의해 메모리에 만들어  진 객체 1개를 인스턴스라 한다. |
| 3 | 다음 중 틀린 것은?  **public** **class** Book {  //클래스의 멤버변수는 개발자가 초기화하지 않으면  //컴파일러에 의해 디폴트값으로 초기화 됨  **int** price; //A 0  **int** page=300;  **boolean** onSale; //B false    **public** **static** **void** main(String[] args) {  //int x; //개발자가 해야함…  price=500; //C  Book b1 = **new** Book(); //D  Book b2 = **new** Book(); //E    b1.page=200; //  System.*out*.println(b2.page);//F  }  }  (1) 위 Book 클래스는 main 실행부가 있기 때문에 java.exe로 실행할 수 있는 클래스이다.  (2) ~~A의 price는 개발자가 초기값을 할당하지 않았기 때문에 아무런 값도 들어있지 않다.~~  (3) ~~B에서 컴파일러에 의해 초기화되는 디폴트값은 true이다~~.  (4) ~~C에 의해 Book 클래스의 price 값은 500으로 변경된다~~.  (5) D, E에 의해 메모리에 올라간 인스턴스는 총 2개이다.  (6) ~~F에서 출력되는 결과는 200이다.~~ |
| 4 | JVM(자바 가상머신)의 논리적 메모리구조에 대한 설명 중 틀린 것은?    (1) ~~.java 파일을 컴파일하는 시점에 .class 파일이 생성되어지면서 메모리에 로드된다.~~  (2) 프로그램 실행시 .class 파일을 해석한 코드가 method 영역에 올라가게 된다.  (3) Stack은 ~~FILO로 데이터가 관리되며~~ 자바의 모든 지역변수는 이 Stack에 생성된다.  (4) 모든 인스턴스는 Heap에 생성되어지며 , 소멸은 GC(Gabage Collector)가 담당한다. |
| 5 | 다음 설명 중 틀린 것은?  (1) 클래스로 부터 인스턴스를 생성하려면 new 연산자를 사용해야 한다.  (2) new 연산자는 일반 데이터가 아닌 덩치 큰 객체 데이터를 메모리에 올릴 때 사용하는 연산자이다.  (3) **일반변수는 변수 자체가 데이터를 포함할 수 있지만, 레퍼런스 변수는 객체를 직접 포함하지 않으**  **며 단지 객체가 메모리상에 위치한 주소 값만을 보유하고 있다.**  (4) ~~지역 변수는 힙(heap) 영역에 생성되며, 객체는 stack 영역에 생성된다.~~ |
| 6 | 다음 중 틀린 것은?  **public** **class** Tree {  **int** root=1; //A 인스턴스 변수  **static** **int** *height*=25; //B **클래스변수: 클래스 영역에 붙여넣어라**    **public** **static** **void** main(String[] args) {  Tree.height=8; // 정상적인 표현식, 변경됨  height=4; // 접근 가능, 변경됨  Tree t1 = **new** Tree(); //C  Tree t2 = **new** Tree(); //D    t1.*height*=30;//E 가능, 원본 변경됨    System.*out*.println(t2.*height*); //F 변경된 값 30  System.*out*.println(Tree.*height*); //G 접근가능, 30  }  }  (1) A에서 root는 인스턴스가 생성될 때 해당 인스턴스 내에 포함되므로, 인스턴스 변수라 불린다.  ~~(2)~~ **~~B에서 height를 사용하기 위해서는 Tree 클래스의 인스턴스 생성이 필요하다.~~**  (3) C,D에서 생성된 각각의 인스턴스마다 root와 **~~height 변수가 힙영역에 함께 생성되어 진다~~**  **(4) E에서 height 변수는 클래스 변수이므로 ~~인스턴스명으로 접근할 수 없다.~~**  **모든 인스턴스 변수는 생성된 영역 접근 가능**  **(5) ~~F의 출력결과는 25가 출력된다.~~ 30**  **(6) G처럼 클래스명으로 변수에 접근하는 것은 불가능하다.** |
|  | 다음 설명 중 틀린 것은?  (1) static으로 선언한 변수와 메서드는 인스턴스에 소속되지 아니하므로, 클래스를 통해서 접근이  가능하다. 따라서 static 으로 선언한 변수를 가리켜 클래스 변수라 한다.  (2) non-static 영역에서는 static 영역에 존재하는 변수 및 메서드를 클래스명만 알면 접근할 수 있지만,  **static 영역에서는 non-static 영역에 있는 변수 및 메서드 접근시 객체의 레퍼런스가 없다면 접근이**  **불가능하다**  (3) 멤버 메서드가 멤버 변수에 접근 못하는 경우도 있다.  (4) 자바의 모든 코드는 클래스안에 존재해야 하므로, 전역변수의 개념이 존재하지 않는다. 따라서  프로그램이 종료될때까지 heap에 올려진 객체들간 데이터를 공유하려면 모든 객체들이 접근할 수  있는 메모리영역  이 필요한데, 이 영역이 바로 static 영역이다.  **(5) 특정 클래스를 대상으로 new 연산자를 3번 적용 시켰다 해도, static으로 선언된 변수는 인스턴스에**  **소속된 변수가 아니라 클래스에 소속된 변수이므로, 메모리에 중복해서 올라가는 일은 없다.** |
| 8 | 빈 칸에 알맞은 단어를 채우세요.  (1) 클래스가 디자인 타임의 객체를 설계한 것이라면 ,실행 타임(=Runtime)시 메모리에 올라가는  객체 한 단위를 ( 인스턴스 )라고 한다.  (2) Car a = new Car() 에서 Car 는 자료형에 해당하고, a는 변수이지만 a와 같이 기본 자료형이 아닌  객체의 주소값을 가지는 변수를 ( **레퍼런스** 변수 ) 라고 한다.  (3) 객체 생성 후 프로그래머가 따로 메모리 관리를 해주지 않아도 자동으로 사용되지 않는 객체들을  소멸시키는 JVM의 메모리 청소기능을 ( 가비지 콜렉터 ) 이라한다.  (4) 메인 메서드는 개발자가 직접 호출 할 수 없으며 실행 시 ( Java.exe ) 에 의해 자동 호출된다.  (5) 레퍼런스 변수값을 출력해보면 @ 형태의 데이터가 출력되는데, 이것은 객체 자체의 값이 아니라 그  객체의 ( 주소 ) 값이다.  (6) 클래스로부터 생성되는 인스턴스가 메모리에 올라갈 때 ,클래스 내부에 정의한 변수, 메서드는 각각의 인스턴스마다 할당되어 지므로, 이 변수와 메서드를 각각 ( 인스턴스 ) 변수 , ( 인스턴스 ) 메서드라  고 한다.  (7) JVM도 프로그램이므로, 메모리를 OS로부터 할당받아 사용한다. 이때 JVM에서 사용되는 메모리  영역은 크게 ( 스태틱 ) , ( 스택 ), ( 힙 ) 영역으로 나눌 수 있다.  (8) 일반 데이터가 아닌 객체의 주소값을 담는 변수를 ( 레퍼런스 ) 라고 한다.  (9) 객체 생성시 프로그래머의 의도와는 상관없이 new 연산자 뒤에서 무조건 호출되는 메서드를  ( 클래스 변수 ) 라고 하며, 반드시 클래스명과 같아야 한다.  (10) 현실의 사물을 객체,(=오브젝트) 라고 하며, 이 객체를 자바 언어로 모델링 한 설계도가 ( 클래스 )이며 이 ( 클래스 )를 기준으로 메모리에 생성된 객체 한 단위를 ( 인스턴스 ) 라고 한다. |
| 9 | 다음 중 틀린 것은?  **public** **class** Phone {    **int** memory=300; //A  **int** price; //B    **public** **void** call(){  **int** n; //C  System.*out*.println("n의 값은 "+n); //D  }  }  (1) A에서 memory변수의 선언과 동시에 300이라는 정수로 초기화 한것이다.  (2) B는 개발자가 초기화하지 않았으므로 컴파일러에 의해 자바의 기본 자료형마다 정해진 디폴트  데이터로 초기화 되는데, 정수값이므로 0으로 초기화된다.  (3) C도 (2)번과 같이 0으로 초기화 될 것이다.  (4) D에서 초기화되지 않은 데이터를 사용하려 했으므로 컴파일 에러가 발생한다.  (5)자바에서 지역변수는 컴파일러가 자동으로 초기화해 주지 않는다. |
| 10 | 다음 중 맞는 것은?  **public** **class** Desk {  **int** x;  **static** **int** *y*;    { //A  **for**(**int** i=0;i<10;i++){  x++;  }  }      **static** { //B  **for**(**int** i=0;i<20;i++){  *y*++;  }  }    **public** **static** **void** main(String[] args){  System.*out*.println(x); //C  **int** a=**new** Desk().x; //D  System.*out*.println(a); //E  System.*out*.println(*y*); //F    }  }  (1) A의 브레이스는 y의 값을 초기화하기 위한 초기화 블럭이다.  (2) B의 브레이스는 클래스변수를 초기화하기 위한 초기화 블럭이다  (3) 변수값의 초기화시 복잡한 로직등을 수행해야 할 경우 블록수준에서도 초기화가 가능하며, 이러한  초기화 블럭은 static으로 선언된 클래스변수에 대해서만 가능하다.  (4) C에서 문법상 문제가 없다.  (5) E의 결과값은 10이다.  (6) F에서 y값을 Desk 클래스의 레퍼런스 없이 접근하려 했으므로 에러가 발생할 것이다. |
| 11 | 다음 중 맞는 것은?  **public** **class** Radio {  {  System.*out*.println("A");  }  **static**{  System.*out*.println("B");  }  **public** **static** **void** main(String[] args) {  System.*out*.println("C");  Radio ra1 = **new** Radio();  Radio ra2 = **new** Radio();  }  }  (1) 이 클래스에는 초기화 블럭이 3개다.    (2) static으로 선언된 초기화 블럭이 실행되는 시점은 컴파일시이다.  (3) 인스턴스 초기화 블럭은 인스턴스가 생성될 때마다 매번 수행된다.    (4) 이 클래스의 실행 순서는 C , B, A , A 이다.  (5) 이 클래스의 실행 순서는 B, C, A, A 이다. |
| 12 | 다음 중 틀린 것은?  **public** **class** Car {  **int** price=100;    **public** **static** **void** main(String[] args) {  **int** x=5; //A  Car car = **new** Car(); //B  {  **int** y=3; //C  }  System.*out*.println(y);//D  System.*out*.println(car);//E  }//F  }  (1) A에서 x는 main 메서드의 지역변수이므로 stack에 쌓여있다가 F를 만나면 stack에서 소멸된다.  (2) B에서 메모리에 올라간 car변수와 Car의 인스턴스는 F를 만나면 메모리에서 모두 소멸된다.  (3) C에서 y는 지역화된 영역안에서만 생명력을 가지므로 D에서 접근할 수 없다.  (4) E에서 car 를 출력하면 주소값이 출력되는데, 그 이유는 car변수가 객체를 직접 보유하지 않고  힙영역에 생성된 객체의 주소를 참조하기 때문이다. |
| 13 | 다음 중 틀린 것은?  **public** **class** Flower {  **int** leaf=10; //A  **static** **int** height=30; //B    **public** **void** grow(){;//C  **int** h=5  }    **public** **static** **void** main(String[] args){  height=20;//D    Flower f1 = **new** Flower();  Flower f2 = **new** Flower();    f2.height=100; //E  System.out.println(height); //F  }  }  (1) 개발자가 Flower라는 객체자료형을 정의한 것이다.  (2) A,B와 같이 클래스 영역에 선언되는 변수를 멤버변수라 하고, C를 멤버메서드라 한다.  (3) A의 leaf는 Flower클래스의 인스턴스가 메모리에 올라갈때 해당 인스턴스에 포함되므로 인스턴스  변수라 부른다.  (4) D에서 height 변수를 접근하려할때 컴파일 에러가 발생할 것이다. |
| 14 | 다음 중 틀린 것은?  **public** **class** Duck {  **int** age=3; //A  **boolean** fly; //B      **public** **static** **void** main(String[] args){  age=5;//C    {  **int** x=3;//D  Duck d = **new** Duck(); //E  }  System.*out*.println(d.age);  }  }  (1) 현실의 오리를 클래스로 표현한 것이다.  (2) A에서 age 변수가 메모리에 올라가는 시점은 클래스를 컴파일할 때이다.  (3) B에서 개발자가 변수 fly를 초기화하지 않았으므로 아무런 값도 들어있지 않다.  (4)오리가 메모리에 올라가는 시점은 프로그램을 실행할 때이다.  (5) C에 의해 오리가 갖는 A의 age 변수값은 5으로 변경된다.  (6) A,B 변수는 오리의 상태를 표현한 속성이므로 오리 인스턴스가 메모리에 올라가야 사용할 수 있다.  (7) D의 변수x는 닫는 브레이스 까지만 생명력을 갖는다.  (8) E에서 d변수는 닫는 브레이스 까지 생명력을 갖는다.  (9) E에서 메모리에 올라온 오리 인스턴스는 닫는 브레이스까지만 생명력을 가지므로 이 시점에  객체가 메모리에서 소멸된다 |
| 15 | 다음 중 틀린 것은?  (1) 자바는 문자,숫자,논리값이외에 제 4의 자료형인 객체자료형을 가진다.  (2) 일반 데이터를 메모리에 올리려면 자료형과 변수를 선언하듯이, 객체자료형도 자료형이므로  이 원칙을 벗어나지 않는다.  (3) OS입장에서는 JVM도여러 응용프로그램 중 하나에 불과하므로, 자신이 관리하는 메모리의  일부를 할당해준다  (4) JVM의 메모리구조는 논리적으로 크게 static, heap, stack 으로 구분할 수 있다.  (5) 자바의 모든 메서드의 지역변수는 stack 에 올라간다.  (6) stack 이란 FILO로 메모리를 관리한다.  (7) 자바의 모든 클래스는 실행에 앞서, 클래스가 메모리로 로드되어야 한다.  (8) 클래스 작성시 main() 메서드는 기본적으로 존재해야 한다.  (9) 실행부가 없다는 것은 단지 현재 클래스로부터 프로그램이 시작되지 않는다는 의미일 뿐이다. |
| 16 | 아래 작성된 2개 클래스에 대해 실행결과를 예상하세요.  **class** Phone{  **int** price=200;  String color="white";  **public** **void** setPrice(){  price = 500;  }  **public** **void** setColor(){  color="red";  }  }  **public** **class** UsePhone{  **public** **static** **void** main(String[] args){  **int** price=5;  Phone ph=**new** Phone();  price=3;  ph.setPrice();  System.*out*.println(price);  }  } |
| 17 | 아래 작성된 2개 클래스에 대해 실행 결과를 예상하세요.  **class** Book{  String title="해리포터";  **int** page=300;  **public** **int** getPage(){  **return** page;  }  **public** **void** setPage(**int** p){  page=p;  }  }  **class** UseBook{  **public** **static** **void** main(String[] args){  Book b=**new** Book();  b.setPage(150);  **int** page=b.getPage();  System.*out*.println("페이지 수는 "+page);  }  } |
| 18 | 아래 작성된 2개 클래스에 대해 실행 결과를 예상하세요.  **class** Computer{  String company="애플";  **int** price=130;  **public** **int** getPrice(){  **return** price;  }  **public** **void** setValue(String c, **int** p){  company=c;  price=p;  }  }  **class** UseComputer{  **public** **static** **void** main(String[] args){  Computer com=**new** Computer();  com.setValue("삼성",200);  **int** a=com.price;  **int** b=com.getPrice();  System.*out*.println("a+b 결과는 "+(a+b));  }  } |
| 19 | 클래스에 대한 올바른 설명은?  **class** TV{  String color="black";  String company="LG";  **int** weight=45;  **public** **void** setColor(String c){  color=c;  }  **public** **static** **void** main(String[] args){  weight=30; (가)  **new** TV(); (나)  String color="blue"; (다)  System.out.println(color); (라)  **int** w=**new** TV().weight; (마)  System.out.println("무게는 "+w); (바)  }  }  (1) (가)에 의해서 TV클래스의 weight 속성값은 30으로 변경될 것이다.  (2) 상태를 나타내는 변수가 총 3개, 동작을 표현하는 메서드가 총 2개 있는 TV 클래스이다.  (3) 실행부가 있는 클래스이므로, 이 클래스는 java.exe로 실행시킬 수 있는 클래스이다.  (4) (나)에서 TV를 메모리에 올리려는 시도는 문법이 잘못되었으므로, 컴파일시 에러가 발생할 것이다.  (5) (다)에서 TV클래스의 color 속성값이 blue로 변경된다.  (6) (라)에서는 "black"이 출력될 것이다.  (7) (마)는 문법상 오류이므로 컴파일이 불가능하다.  (8) (바)의 출력결과는 “무게는 45” 이다.  (9) 메인 실행부에서 메모리에 올려진 TV의 인스턴스는 총 1개이다. |