**생성자**

|  |  |
| --- | --- |
| 1 | 아래의 클래스에 대한 설명으로 틀린 것은?  **class** Rose{  **int** leaf=0;  String color="red";  **public** **void** fall(){  leaf=0;  }  }  **class** UseRose{  **public** **static** **void** main(){  Rose r1=**new** Rose(); // (가)  r1.fall(); //(나)  r1.bloom(); //(다)  }  }  (1) (다)에서의 bloom() 은 Rose 클래스에 존재하지 않는 메서드이므로 컴파일 에러가 발생한다.  (2) (나)에서의 fall()은 Rose 클래스의 멤버 메서드를 호출하고 있는 것이다.  (3) ~~(가)에서 장미 인스턴스를 생성하는 과정에서 new 연산자 뒤에 오는 Rose()는 Rose 클래내에 작성~~  ~~한 적이 없으므로 컴파일 에러가 발생 할 것이다.~~  (4) (가)에서 new 연산자 뒤에 오는 Rose() 메서드는 생성자 메서드이고, 개발자가 정의하지 않으면  컴파일러에 의해 디폴트 생성자가 존재하게 된다. |
| 2 | 생성자 메서드에 대한 설명 중 틀린 것은?  (1)객체가 new로 생성되어 질때 무조건 호출이 강제되는 메서드이다.  (2)자바에서는 생성자 메서드의 이름은 반드시 클래스명과 일치해야 한다.  (3)**생성자에는 반환형을 명시해서는 안된다.**  (4)인스턴스 생성시 new연산자 뒤에 생성자 호출은 무조건 강제되므로, 만일 개발자가 클래스내에  생성자를 정의하지 않을 경우, 존재하지 않는 메서드 호출이 되어 에러가 발생하게 된다. 따라서  이러한 문제가 발생되지 않으려면, 컴파일러에 의해 최소한의 기능을 가진 생성자가 자동으로 정의되어  지는데, 이러한 생성자를 디폴트 생성자라 한다.  (5)~~개발자가 생성자를 정의해도 디폴트 생성자는 컴파일러에 의해 계속 존재하게 된다.~~  (6) Sun사가 객체의 생성시, 생성자 메서드를 무조건 적으로 강제하고 있는 이유는 개발자로 하여금  객체 생성 타임에 초기화의 기회를 주기 위해서이다. |
| 3 | 생성자 메서드와 일반메서드의 차이점에 대한 설명 중 틀린 것은?  (1)**생성자가 아니더라도 객체에 개성을 부여하는 방법은 일반 메서드로도 가능하다.**  (2)생성자도 메서드이다.  (3) 실행 시기적 관점에서 본다면 생성자와 일반 메서드는 단지 작업의 착수 시점이 다를 뿐이다.  특히 생성자의 호출을 무조건적으로 강제한 이유는 개발자로 하여금 객체 생성 초기에 원하는  초기화 작업을 할 수 있도록 배려한 것이다.  (4) 위(3)번의 배려에도 불구하고 개발자가 생성자를 정의하지 않으면, 생성자가 존재하지 않게 되어  new 연산자에서 호출시 에러가 발생하게 되므로, sun에서는 개발자가 정의하지 않을 경우 최소  한의 관여만 하여 디폴트 생성자를 두게끔 처리했다.    (5) ~~생성자에 반환형을 둘 수 있다.~~  (6) 현실에 비유하자면 생성자의 사용은 제품의 출시 시점에 해야 할 작업에, 일반메서드 사용은 제품  출시 이후 변경 사항을 반영하는 것으로 비유할 수 있다 |
| 4 | 아래의 클래스를 생산 시점부터 빨간색인 하이힐을  만들수 있는 클래스로 완성하시오  **class** HighHeel{  String color="Red";  } |
| 5 | 아래의 클래스를 생산 시점부터 검정색으로 만들 수 있는 기능 및 생산한 이후에 언제든 색상을  빨간색으로 변경 할 수 있는 클래스로 완성하시오  **class** SportsCar{  String color="None Color";  public SportsCar(){  color = “black”;  }  public void setColor(String color){  This.color = color;  }  } |
| 6 | 아래의 클래스에 대한 설명 중 맞는 것을 고르면?  **class** Doll{  **static** String *name*="피카츄"; // 클래스변수  **int** price; // 인스턴스 변수  **public** Doll(**int** p){  price=p;  }  **public** **static** **void** main(String[] args){  Doll.*name*="테디베어"; //(가)  Doll d=**new** Doll(); //(나)  d.*name*="뽀로로"; //(다)  }  }  (1) 이 클래스의 멤버 변수는 2개이다.  (2) ~~개발자가 생성자를 정의하지 않았으므로 디폴트 생성자만 존재한다~~.  (3) ~~(가) 에서 아직 인형의 인스턴스가 생성되지 못한 상태에서 name변수를 접근하려 했으므로,~~  ~~컴파일 에러가 발생한다.~~  (4) ~~(나) 는 인형을 메모리에 올리는 코드이므로 잘못된 부분이 없다~~.  (5) ~~static 으로 선언된 name 변수는 (다) 와 같은 레퍼런스 변수명으로는 접근이 불가능하다.~~  **(6) 이 클래스는 개발자가 생성자를 정의하였으므로, (나)에서 인형의 인스턴스를 생성하려할 때 Doll()**  **은 존재하지 않는 메서드이므로, 컴파일 에러가 발생할 것이다.** // 매개변수 필요 |
| 7 | 아래의 클래스에 대한 설명 중 맞는 것은?  **class** Movie{  String title;  **int** price;  **public** Movie(**int** p){  price=p;  }  **public** Movie(String t){  title=t;  }  **public** **static** **void** show(){  System.out.println(“movie start!!”);  }  **public** **static** **void** main(String[] args){  Movie mv1=**new** Movie(); //(가)  Movie.*show*(); //(나)  }  }  (1) ~~이 클래스는 생성자가 2개나 정의되어 있으므로 컴파일 에러가 발생할 것이다.~~  (2) (~~가)에서 Movie 클래스의 생성자 호출에는 문제가 없다.~~  (3) ~~Movie 클래스에는 컴파일러에 의한 디폴트 생성자가 존재한다~~.  (4) 생성자 오버로딩이 적용된 클래스이며, 디폴트 생성자는 존재하지 않는다.  (5) ~~(나)의 show() 메서드는 인스턴스 메서드이므로 클래스명으로 접근할 수 없다.~~  (6) 생성자도 메서드이므로, 오버로딩이 적용될 수 있다. |