유형별로 분류한 프로그래밍 – JAVA활용 9문제
정보처리기사 실기

[클래스 기본]

1. 다음 Java로 구현된 프로그램을 분석하여 그 실행 결과를 쓰시오. (단, 출력문의 출력 서식을 준수하시오.)

```
class Test {
    public static void main(String args[]) {
        cond obj = new cond(3);
        obj.a = 5;
        int b = obj.func( );
        System.out.print(obj.a + b);
    }
}

class cond {
    int a;
    public cond(int a) {
        this.a = a;
    }
    public int func( ) {
        int b = 1;
        for (int i = 1; i < a; i++)
            b += a * i;
        return a + b;
    }
}
```

답 : 61

[해설]

```
class Test {
    public static void main(String args[]) {
❶       cond obj = new cond(3);
❹       obj.a = 5;
❺❶     int b = obj.func( );
⓬      System.out.print(obj.a + b);
```

```
    }
}

class cond {                        클래스 cond를 정의한다.
    int a;                          정수형 변수 a를 선언한다.
❷   public cond(int a) {
❸       this.a = a;
    }
❻   public int func( ) {
❼       int b = 1;
❽       for (int i = 1; i < a; i++)
❾           b += a * i;
❿       return a + b;
    }
}
```

모든 Java 프로그램은 반드시 main( ) 메소드에서 시작한다.

❶ 3을 인수로 생성자를 호출하여 cond 클래스의 객체 변수 obj를 선언한다.

❷ cond 클래스 생성자의 시작점이다. ❶번에서 전달받은 3을 a가 받는다.

❸ cond 클래스의 a에 3을 저장한다. 생성자가 종료되면 호출했던 ❶번의 다음 줄인 ❹번으로 이동한다. → **obj.a = 3**

- **this** : 현재의 실행중인 메소드가 속한 클래스를 가리키는 예약어이다. 여기에서는 cond 클래스의 객체 변수 obj의 생성자로 호출되었으므로 'obj.a'와 같은 의미이다.

❹ obj.a에 5를 저장한다. → **obj.a = 5**

❺ 정수형 변수 b를 선언하고 obj.func( ) 메소드를 호출한 후 돌려받은 값으로 초기화한다.

❻ 정수를 반환하는 func( ) 메소드의 시작점이다.

❼ 정수형 변수 b를 선언하고 1로 초기화한다.

❽ 반복 변수 i가 1부터 1씩 증가하면서 a보다 작은 동안 ❾번을 반복 수행한다. func( ) 메소드에는 별도로 생성한 'a'라는 변수가 없으므로 cond 클래스의 a를 가져와 사용한다. 즉 ❾번은 5보다 작은 동안 반복 수행된다.

❾ 'b = b + (a * i);'와 동일하다. a에 i를 곱한 값을 b에 누적시킨다.
반복문 실행에 따른 변수들의 변화는 다음과 같다.

a	i	b
5		1
	1	6
	2	16
	3	31
	4	51
	5	

❿ 5와 51을 더한 값 56을 메소드를 호출했던 ⓫번으로 반환한다.

⓫ b에 56이 저장된다.

⓬ 5+56의 결과인 **61**을 출력한다.

결과 **61**

2. 다음 Java로 구현된 프로그램을 분석하여 그 실행 결과를 쓰시오. (단, 출력문의 출력 서식을 준수하시오.)

```
class A {
    int a;
    int b;
}

public class Test {
    static void func1(A m) {
        m.a *= 10;
    }
    static void func2(A m) {
        m.a += m.b;
    }
    public static void main(String args[]) {
        A m = new A();
        m.a = 100;
        func1(m);
        m.b = m.a;
        func2(m);
        System.out.printf("%d", m.a);
    }
}
```

답 : 2000

[해설]

```
class A {           클래스 A를 정의한다.
    int a;          클래스 A에는 정수형 변수 a와 b가 선언되어 있다.
    int b;
}

public class Test {
❹   static void func1(A m) {
❺       m.a *= 10;
    }
❽   static void func2(A m) {
❾       m.a += m.b;
    }
    public static void main(String args[]) {
❶       A m = new A();
❷       m.a = 100;
```

```
❸          func1(m);
❻          m.b = m.a;
❼          func2(m);
❿          System.out.printf("%d", m.a);
       }
   }
```

모든 Java 프로그램은 반드시 main( ) 메소드에서 시작한다.

❶ 클래스 A의 객체 변수 m을 선언한다.

	int a	int b
객체 변수 m		

❷ 객체 변수 m의 변수 a에 100을 저장한다.

	int a	int b
객체 변수 m	100	

❸ 객체 변수 m의 시작 주소를 인수로 하여 func1 메소드를 호출한다.

❹ 반환값이 없는 func1( ) 메소드의 시작점이다. ❸번에서 전달받은 주소는 m이 받는다.

※ 객체 변수나 배열의 이름은 객체 변수나 배열의 시작 주소를 가리키므로, 인수로 전달하는 경우 메소드에서 변경된 값이 main( )의 객체 변수나 배열에도 적용된다는 점을 염두에 두세요.

❺ 'm.a = m.a * 10;'과 동일하다. m.a에 10을 곱한 값을 m.a에 저장한다. 메소드가 종료되었으므로 메소드를 호출했던 ❸번의 다음 줄인 ❻번으로 이동한다.

	int a	int b
객체 변수 m	1000	

❻ m.b에 m.a의 값 1000을 저장한다.

	int a	int b
객체 변수 m	1000	1000

❼ 객체 변수 m의 시작 주소를 인수로 하여 func2 메소드를 호출한다.

❽ 반환값이 없는 func2( ) 메소드의 시작점이다. ❼번에서 전달받은 주소는 m이 받는다.

❾ 'm.a = m.a + m.b;'와 동일하다. m.a와 m.b를 합한 값을 m.a에 저장한다. 메소드가 종료되었으므로 메소드를 호출했던 ❼번의 다음 줄인 ❿번으로 이동한다.

	int a	int b
객체 변수 m	2000	1000

❿ m.a의 값 2000을 정수로 출력한다.

결과 2000

[상속과 재정의]

3. 다음 Java로 구현된 프로그램을 분석하여 그 실행 결과를 쓰시오. (단, 출력문의 출력 서식을 준수하시오.)

```
public class ovr1 {
	public static void main(String[] args) {
		ovr1 a1 = new ovr1();
		ovr2 a2 = new ovr2();
		System.out.println(a1.sun(3,2) + a2.sun(3,2));
	}
	int sun(int x, int y) {
		return x + y;
	}
}
class ovr2 extends ovr1 {
	int sun(int x, int y) {
		return x - y + super.sun(x, y);
	}
}
```

답 : 11

[해설]

```
public class ovr1 {
	public static void main(String[] args) {
❶		ovr1 a1 = new ovr1();
❷		ovr2 a2 = new ovr2();
❸❻⓬		System.out.println(a1.sun(3,2) + a2.sun(3,2));
	}
❹❾	int sun(int x, int y) {
❺❿		return x + y;
	}
}
class ovr2 extends ovr1 {
❼	int sun(int x, int y) {
❽⓫		return x - y + super.sun(x, y);
	}
}
```

클래스 ovr2를 정의하고 부모 클래스로 ovr1을 지정하면서 ovr1에 속한 변수와 메소드를 상속받는다.

모든 Java 프로그램은 반드시 main( ) 메소드에서 시작한다.

❶ 클래스 ovr1의 객체변수 a1을 선언한다.

❷ 클래스 ovr2의 객체변수 a2를 선언한다.

❸ 3과 2를 인수로 a1의 sun( ) 메소드를 호출한 결과와, 3과 2를 인수로 a2의 sun( ) 메소드를 호출한 결과를 합하여 출력한 후 커서를 다음 줄로 옮긴다. 먼저 a1의 sun( ) 메소드를 호출한다.
❹ 정수를 반환하는 a1의 sun( ) 메소드의 시작점이다. ❸번에서 전달받은 3과 2를 x와 y가 받는다.
❺ x와 y를 더한 값 5를 함수를 호출했던 ❸번으로 반환한다.
❻ ❺번으로부터 a1의 sun( ) 메소드를 호출한 결과로 5를 전달받았으므로, 이번에는 3과 2를 인수로 a2의 sun( ) 메소드를 호출한다.
❼ 정수를 반환하는 a2의 sun( ) 메소드의 시작점이다. ❻번에서 전달받은 3과 2를 x와 y가 받는다.
❽ x에서 y를 뺀 값에 3과 2를 인수로 부모 클래스인 ovr1의 sun( ) 메소드를 호출한 결과를 더하여 함수를 호출했던 ⓬번으로 반환한다.
❾~⓫ 3과 2를 인수로 ovr1의 sun( ) 메소드를 수행한 결과를 ❹~❺번에서 구했으므로 그 결과를 그대로 사용하면 된다. ⓾번에서 5를 돌려받아 계산한 값 6(3-2+5)을 함수를 호출했던 ⓬번으로 반환한다.
⓬ ❺번으로부터 돌려받은 5와 ⓫번으로부터 돌려받은 6을 더한 값 11을 출력하고 커서를 다음 줄로 옮긴다.

결과 11

4. 다음 Java로 구현된 프로그램을 분석하여 그 실행 결과를 쓰시오. (단, 출력문의 출력 서식을 준수하시오.)

```
class A {
    int a;
    public A(int a) { this.a = a; }
    void display() { System.out.println("a=" + a); }
}
class B extends A {
    public B(int a) {
        super(a);
        super.display();
    }
}
public class Test {
    public static void main(String[] args) {
        B obj = new B(10);
    }
}
```

답 : a=10

[해설]

```
   class A {                    클래스 A를 정의한다.
       int a;
❹      public A(int a) { ❺ this.a = a; }
❼      void display() { ❽ System.out.println("a=" + a); }
   }
   class B extends A {          클래스 B를 정의하고 부모 클래스로 A를 지정하면서 A에 속한 변수와
                                메소드를 상속받는다.
❷      public B(int a) {
❸          super(a);
❻          super.display();
       } ❾
   }
   public class Test {
       public static void main(String[] args) {
❶          B obj = new B(10);
       } ❿
   }
```

❶ B obj = new B(10);
클래스 B의 객체 변수 obj를 선언하고 생성자에 인수 10을 전달한다.

❷ B 클래스 생성자 B( )의 시작점이다. ❶번에서 전달받은 10을 정수형 변수 a가 받는다.

❸ 부모 클래스의 생성자를 호출하며 인수로 a의 값 10을 전달한다.
※ **super** : 상속한 부모 클래스를 가리키는 예약어
❹ 생성자 A( )의 시작점이다. ❸번에서 전달받은 10을 생성자 A( )의 변수 a가 받는다.
❺ 메소드가 속한 A 클래스의 a에 A( ) 생성자의 변수 a의 값 10을 저장한다. 생성자가 종료되면 호출했던 ❸번의 다음 줄인 ❻번으로 간다.
※ **this** : 현재의 실행중인 메소드가 속한 클래스를 가리키는 예약어, 즉 'A.a'와 같은 의미이다.
❻ 부모 클래스의 메소드 display( )를 호출한다.
❼ A 클래스의 메소드 display( )의 시작점이다.
❽ "a="를 출력한 후 a의 값을 출력해야 하지만, 메소드에서 별도로 생성한 'a'라는 변수가 없으므로 클래스의 변수 a의 값 10을 출력하고, 다음 줄의 처음으로 커서를 이동시킨다.
※ 생성자나 메소드 안에서 생성된 변수는 생성자나 메소드를 벗어나서 사용하지 못하기 때문에 여기서는 생성자 A( )에 속한 a가 아닌 클래스 A에 속한 a를 출력한다.

결과 `a=10`

메소드를 호출했던 ❻번의 다음 줄인 ❾번으로 이동하고 이어서 B 클래스를 호출했던 ❶번의 다음 줄인 ❿번으로 이동하여 프로그램을 종료한다.

[인터페이스 객체]

5. 다음 Java로 구현된 프로그램을 분석하여 괄호에 들어갈 알맞은 답을 쓰시오.

```
class Car implements Runnable {
    int a;
    public void run() {
        try {
            while(++a < 100) {
                System.out.println("miles traveled : " + a);
                Thread.sleep(100);
            }
        } catch(Exception E) { }
    }
}

public class Test {
    public static void main(String args[]) {
        Thread t1 = new Thread(new (Car )());
        t1.start();
    }
}
```

답 : Car

[해설]

```
Ⓐ class Car implements Runnable {
      int a;
Ⓑ     public void run() {
Ⓒ1        try {
  2           while(++a < 100) {
  3               System.out.println("miles traveled : " + a);
  4               Thread.sleep(100);
              }
Ⓓ         } catch(Exception E) { }
      }
   }

   public class Test {
      public static void main(String args[]) {
Ⓔ❶        Thread t1 = new Thread(new Car());
Ⓕ❷        t1.start();
      } ❸
```

```
}
```

Ⓐ **class Car implements Runnable**

Runnable 인터페이스를 상속받은 클래스 Car를 정의한다.

- **implements** : extends와 같이 상속에 사용하는 예약어로, 인터페이스를 상속받을 때 사용함
- **Runnable** : 스레드 클래스를 만들 때 사용하는 인터페이스

※ 인터페이스 개체는 클래스와 크게 다르지 않습니다. 그 역할이 인터페이스로 고정되어 있을 뿐 클래스와 마찬가지로 변수와 메소드를 갖는 개체입니다.

※ 스레드는 시스템의 여러 자원을 할당받아 실행하는 프로그램의 단위입니다. 대부분은 main( ) 메소드로 실행하는 하나의 스레드로만 작업을 수행하는데, 스레드 클래스는 main( ) 메소드로 실행하는 스레드 외에 추가적인 스레드를 가질 수 있도록 스레드를 생성하는 기능을 갖고 있습니다.

Ⓑ **public void run( )**

Runnable 인터페이스를 상속받았다면 스레드가 수행할 작업들을 정의하는 run( ) 메소드를 반드시 정의해야 한다.

Ⓒ **try { }**

- 실행 중에 예외가 발생할 가능성이 있는 실행 코드들을 하나의 블록으로 묶어 놓은 곳이다. try 블록 코드를 수행하다 예외가 발생하면 예외를 처리하는 Ⓓ의 catch 블록으로 이동하여 예외 처리 코드를 수행하므로 예외가 발생한 이후의 코드는 실행되지 않는다.
- 4번에서 수행되는 Thread.sleep( ) 메소드는 인터럽트로 인한 예외를 발생시킬 가능성이 큰 메소드이므로 반드시 try ~ catch 문을 통해 예외를 처리해줘야 한다.

Ⓓ **catch(Exception E) { }**

인터럽트로 인한 예외를 처리할 수 있는 예외 객체는 InterruptedException이지만, Exception을 사용하면 InterruptedException을 포함한 대부분의 예외를 한 번에 처리할 수 있다.

Ⓔ **Thread t1 = new Thread(new Car( ));**

스레드 클래스의 객체 변수 t1을 선언한다. 스레드 클래스는 생성자를 호출할 때 Runnable 인터페이스를 인수로 사용한다. 여기에서는 Runnable 인터페이스를 상속받은 Car 클래스를 생성자의 인수로 사용했다.

Ⓕ **t1.start( );**

t1의 start( ) 메소드를 호출한다. start( ) 메소드는 스레드 클래스에 포함된 메소드로, run( ) 메소드에서 정의한 코드들을 실행하는 메소드이다. 이때 run( ) 메소드에서 정의한 코드들은 main( ) 메소드와는 별개로 시스템으로부터 자원을 새로 할당받아 실행된다. 즉 main( ) 메소드와 별개로 실행되기 때문에 main( ) 메소드의 작업이 종료되어도 run( ) 메소드의 작업이 끝나지 않으면 계속 수행한다.

모든 Java 프로그램은 반드시 main( ) 메소드에서 시작한다.

❶ 스레드 클래스의 객체 변수 t1을 선언한다. 스레드에서 실행할 run( ) 메소드를 정의하고 있는 Car( ) 클래스를 생성자의 인수로 사용한다.

❷ t1의 start( ) 메소드를 호출한다. Car 클래스의 run( ) 메소드가 실행된다. 이후 main( ) 메소드는 ❸번으로 이동하여 프로그램을 종료한다.

main( ) 메소드와는 별개로 시스템으로부터 자원을 새로 할당받아 run( ) 메소드를 시작한다.

1 예외를 처리하기 위한 try ~ catch문의 시작점이다.

2 a가 100보다 작은 동안 3, 4번을 반복 수행한다. a는 전치증가 연산이므로 a에 1을 더한 후 조건

을 확인한다.

※ 클래스의 속성으로 선언된 변수 a는 자동으로 0으로 초기화됩니다.

3 **miles traveled** : 를 출력한 후 이어서 a의 값을 출력한다.

4 100을 인수로 Thread 클래스의 sleep( ) 메소드를 호출한다. 0.1초 동안 스레드를 일시 정지시킨다.

- **Thread.sleep(n)** : n/1000초 동안 스레드를 일시 정지시킨다.

2~4번을 수행한 결과로 다음과 같이 0.1초마다 한 줄씩 출력된다.

결과

```
miles traveled : 1
miles traveled : 2
miles traveled : 3
miles traveled : 4
       ⋮
miles traveled : 98
miles traveled : 99
```

[클래스 응용 - 싱글톤 패턴]

6. 다음 Java로 구현된 프로그램을 분석하여 그 실행 결과를 쓰시오. (단, 출력문의 출력 서식을 준수하시오.)

```
class Connection {
	private static Connection _inst = null;
	private int count = 0;
	static public Connection get() {
		if(_inst == null) {
			_inst = new Connection();
			return _inst;
		}
		return _inst;
	}
	public void count() { count++; }
	public int getCount() { return count; }
}

public class Test {
	public static void main(String[] args) {
		Connection conn1 = Connection.get();
		conn1.count();
		Connection conn2 = Connection.get();
		conn2.count();
		Connection conn3 = Connection.get();
		conn3.count();
		System.out.print(conn1.getCount());
	}
}
```

답 : 3

[해설]

이 문제는 객체 변수 _inst가 사용하는 메모리 공간을 객체 변수 conn1, conn2, conn3이 공유함으로써 메모리 낭비를 방지하는 싱글톤(Singleton) 개념을 Java로 구현한 문제입니다.

```
class Connection {          클래스 Connection을 정의한다.
Ⓐ     private static Connection _inst = null;
Ⓑ     private int count = 0;
❷⑩⑰  public static Connection get() {
❸⑪⑱       if(_inst == null) {
❹                 _inst = new Connection();
```

```
❺                       return _inst;
                    }
⓬⓳                  return _inst;
            }
❽⓯㉒        public void count() { count++; }
㉔           public int getCount() { return count; }
}

public class Test {
            public static void main(String[] args) {
❶❻                  Connection conn1 = Connection.get();
❼                   conn1.count();
❾⓭                  Connection conn2 = Connection.get();
⓮                   conn2.count();
⓰⓴                  Connection conn3 = Connection.get();
㉑                   conn3.count();
㉓㉕                 System.out.print(conn1.getCount());
            }
}
```

Ⓐ Connection 클래스의 객체 변수 _inst를 선언하고 null로 초기화한다.

※ 객체 변수를 생성한다는 것은 Connection _inst = new Connection( );과 같이 객체 생성 예약어인 new를 통해 heap 영역에 공간을 확보하여 Connection 클래스의 내용을 저장한 후 그 주소를 객체 변수에 저장하는 것인데, Ⓐ에서는 객체 생성 예약어인 new가 생략되었으므로 생성이 아닌 선언만 합니다. 객체 변수를 선언만 하게 되면 heap이 아닌 stack 영역에 내용 없이 저장되어 사용이 불가능합니다. 이후 ❹번과 같이 객체 생성 예약어인 new가 사용되어야만 heap 영역에 내용이 저장되고 그 주소도 객체 변수에 전달되면서 사용 가능한 객체 변수가 됩니다.

Ⓑ 정수형 변수 count를 선언하고, 0으로 초기화한다.

stack 영역	
변수	값
_inst	null
count	0

heap 영역	
주소	내용

모든 Java 프로그램은 반드시 main( ) 메소드에서 시작한다.

❶ Connection 클래스의 객체 변수 conn1을 선언하고, get( ) 메소드를 호출한 결과를 저장한다.

※ Ⓐ에서와 같이 객체 변수를 선언만 하였으므로 객체 변수 conn1은 stack 영역에 생성됩니다.

stack 영역	
변수	값
_inst	null
count	0
conn1	

heap 영역	
주소	내용

❷ Connection 형을 반환하는 get( ) 메소드의 시작점이다.

❸ _inst가 null이면 ❹, ❺번을 수행하고, 아니면 ⓬번으로 이동한다. _inst가 null이므로 ❹번으로 이동한다.

❹ Connection 클래스의 내용을 heap 영역에 저장하고 그 주소를 _inst에 저장한다.

※ Ⓐ에서 객체 변수 _inst는 이미 선언되었으므로, Connection _inst = new Connection( );과 같이 작성하지 않고 앞쪽의 클래스명을 생략하여 _inst = new Connection( );과 같이 작성합니다. 생성 예약어인 new를 통해 heap 영역에 공간을 확보하고 Connection 클래스의 내용을 저장한 후 그 주소를 객체 변수 _inst에 저장합니다. 이제 객체 변수 _inst는 Connection( ) 클래스의 내용이 저장된 heap 영역을 가리키게 됩니다.

stack 영역	
변수	값
_inst	100
count	0
conn1	

heap 영역	
주소	내용
0	
100	private static Connection _inst private int count = 0 static public Connection get() { ... } public void count() { ... } public int getCount() { ... }
200	
300	

❺ _inst에 저장된 값을 메소드를 호출했던 ❻번으로 반환한다.

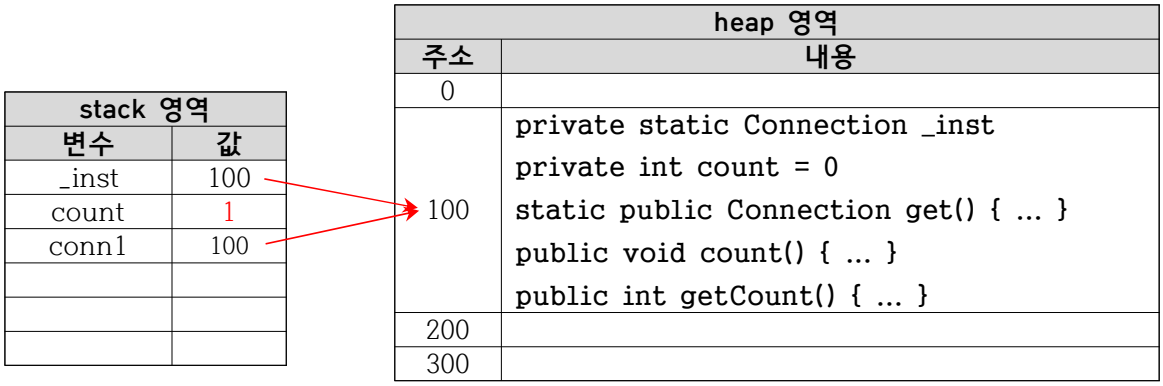
❻ ❺번에서 돌려받은 _inst의 값을 conn1에 저장한다. _inst에는 Connection( ) 클래스의 내용이 저장된 heap 영역의 주소가 저장되어 있으며, conn1에도 동일한 주소가 저장되므로 이후 _inst와 conn1은 같은 heap 영역의 주소를 가리키게 된다.

stack 영역	
변수	값
_inst	100
count	0
conn1	100

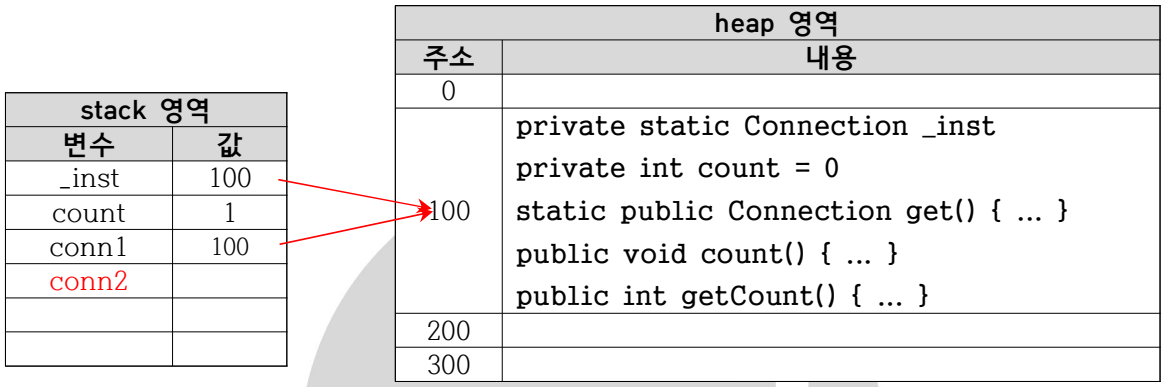
heap 영역	
주소	내용
0	
100	private static Connection _inst private int count = 0 static public Connection get() { ... } public void count() { ... } public int getCount() { ... }
200	
300	

❼ conn1의 count( ) 메소드를 호출한다. conn1은 Connection( ) 클래스의 객체 변수이므로 Connection 클래스의 count( ) 메소드를 호출한다는 의미이다.

❽ 반환값이 없는 count( ) 메소드의 시작점이다. count의 값에 1을 더한 후 count( ) 메소드를 호출했던 ❼번으로 돌아가 다음 문장인 ❾번을 수행한다.



❾ Connection 클래스의 객체 변수 conn2를 선언하고, get( ) 메소드를 호출한 결과를 저장한다.

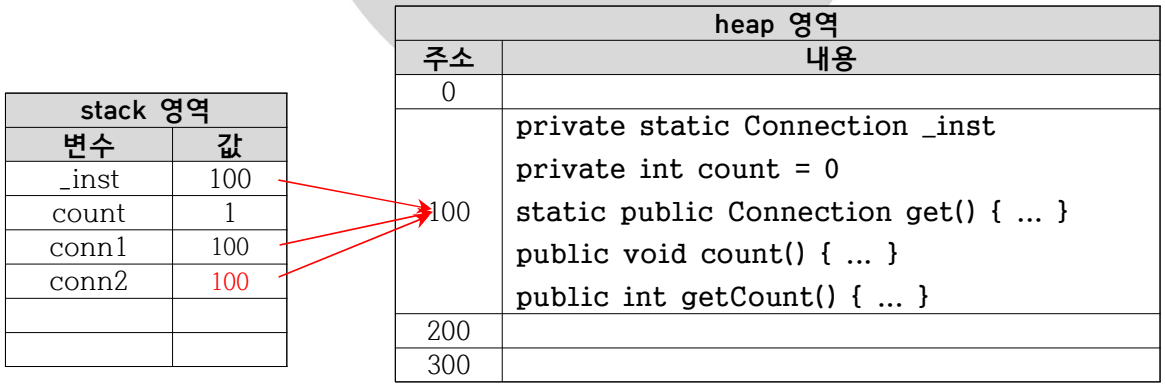


❿ Connection 형을 반환하는 get( ) 메소드의 시작점이다.

⓫ _inst가 null이면 ❹, ❺번을 수행하고, 아니면 ⓬번으로 이동한다. _inst에는 ❹번에서 저장한 heap 영역의 주소가 저장되어 있어 null이 아니므로 ⓬번으로 이동한다.

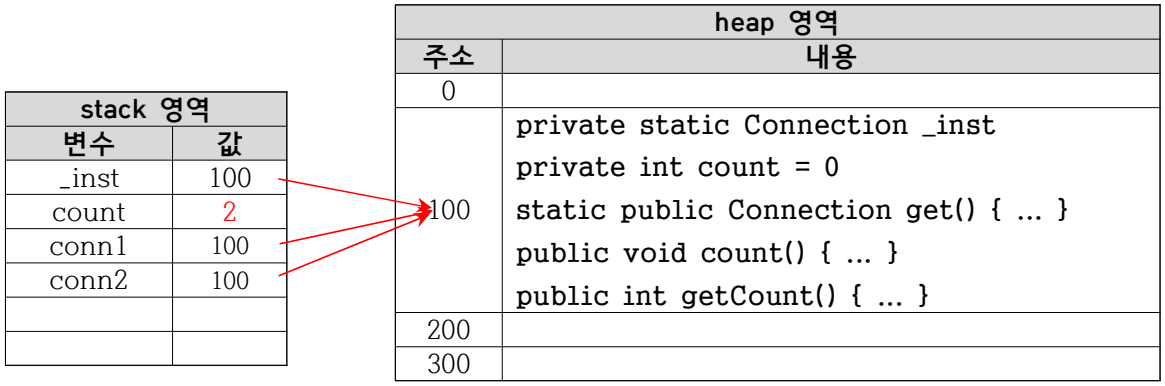
⓬ _inst에 저장된 값을 메소드를 호출했던 ⓭번으로 반환한다.

⓭ ⓬번에서 돌려받은 _inst의 값을 conn2에 저장한다.

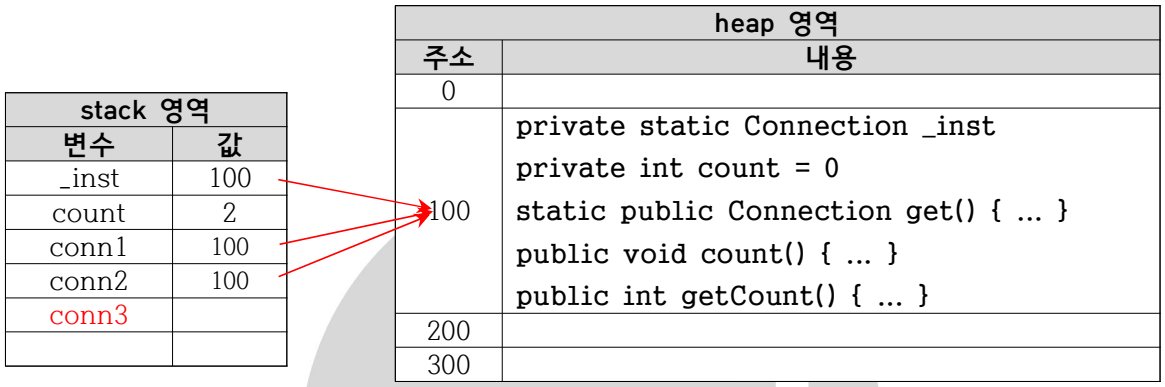


⓮ conn2의 count( ) 메소드를 호출한다.

⓯ 반환값이 없는 count( ) 메소드의 시작점이다. count의 값에 1을 더한 후 count( ) 메소드를 호출했던 ⓮번으로 돌아가 다음 문장인 ⓰번을 수행한다.



⑯ Connection 클래스의 객체 변수 conn3을 선언하고, get( ) 메소드를 호출한 결과를 저장한다.

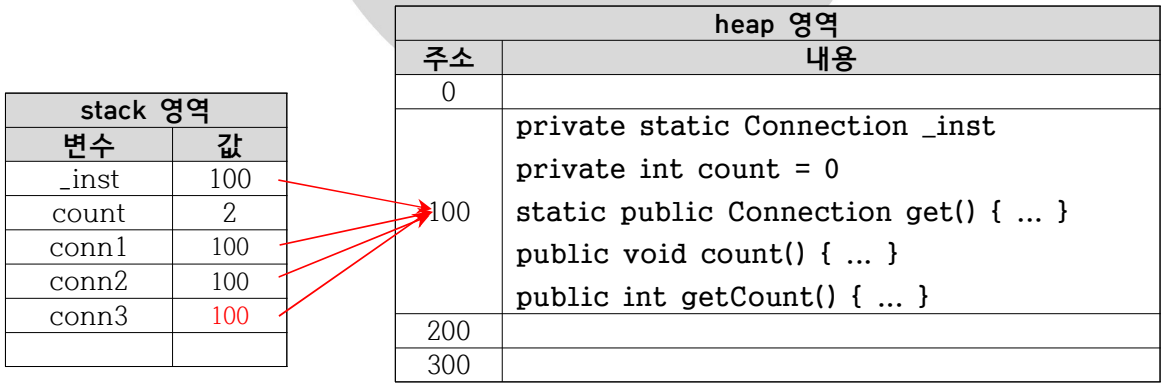


⑰ Connection 형을 반환하는 get( ) 메소드의 시작점이다.

⑱ _inst가 null이면 ❹, ❺번을 수행하고, 아니면 ⑲번으로 이동한다. _inst가 null이 아니므로 ⑲번으로 이동한다.

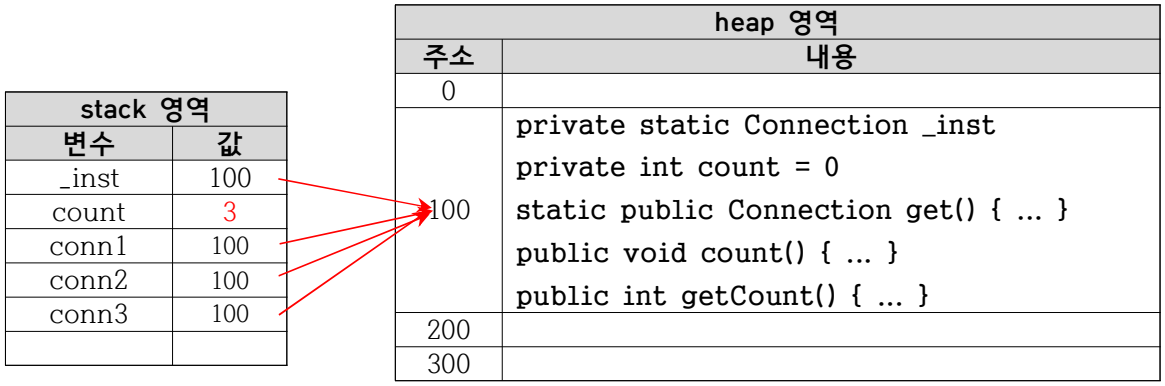
⑲ _inst에 저장된 값을 메소드를 호출했던 ⑳번으로 반환한다.

⑳ ⑲번에서 돌려받은 _inst의 값을 conn3에 저장한다.



㉑ conn3 객체 변수의 count() 메소드를 호출한다.

㉒ 반환값이 없는 count( ) 메소드의 시작점이다. count의 값에 1을 더한 후 count( ) 메소드를 호출했던 ㉑번으로 돌아가 다음 문장인 ㉓번을 수행한다.



stack 영역	
변수	값
_inst	100
count	3
conn1	100
conn2	100
conn3	100

heap 영역	
주소	내용
0	
100	private static Connection _inst private int count = 0 static public Connection get() { ... } public void count() { ... } public int getCount() { ... }
200	
300	

㉓ conn1의 getCount() 메소드를 호출하고 돌려받은 값을 출력한다.

㉔ 정수를 반환하는 getCount() 메소드의 시작점이다. count의 값 3을 메소드를 호출했던 ㉕번으로 반환한다.

※ 객체 변수 _inst, conn1, conn2, conn3은 모두 같은 heap 영역의 주소를 가리키고 있으므로 해당 heap 영역에 저장된 내용을 공유하게 됩니다.

㉕ ㉔번에서 돌려 받은 화면에 3을 출력한다.

결과 3

[객체의 형 변환]

7. 다음 Java로 구현된 프로그램을 분석하여 괄호에 들어갈 알맞은 답을 쓰시오.

```
class Parent {
    void show() { System.out.println("parent"); }
}
class Child extends Parent {
    void show() { System.out.println("child"); }
}
public class Test {
    public static void main(String[] args) {
        Parent pa = (new) Child();
        pa.show();
    }
}
```

답 : new

[해설]

```
class Parent {                          클래스 Parent를 정의한다.
    void show() { System.out.println("parent"); }
}
class Child extends Parent {            클래스 Child를 정의하고 부모 클래스로 Parent를 지정하면서 Parent에 속한 변수와 메소드를 상속받는다.
❸   void show() { ❹ System.out.println("child"); }
}
public class Test {
    public static void main(String[] args) {
❶       Parent pa = new Child();
❷       pa.show();
    } ❺
}
```

모든 Java 프로그램은 반드시 main( ) 메소드에서 시작한다.

❶ **Parent pa = new Child( );**

Child 클래스의 생성자를 이용하여 Parent 클래스의 객체 변수 pa를 선언한다.

- **[부모클래스명] [객체변수명] = new [자식클래스생성자( )]** : 부모 클래스의 객체 변수를 선언하면서 자식 클래스의 생성자를 사용하면 형 변환이 발생한다.
- 이렇게 형 변환이 발생했을 때 부모 클래스와 자식 클래스에 동일한 속성이나 메소드가 있으면 자식 클래스의 속성이나 메소드로 재정의된다.

❷ Pa의 show( ) 메소드를 호출한다. ❸번으로 이동한다.

pa.show( )는 pa 객체의 자료형이 Parent이므로 Parent.show( )라고 생각할 수 있지만 ❶번에서 클래스 형 변환이 발생하였고, show( ) 메소드가 자식 클래스에서 재정의되었으므로 Child 클래스

의 show( ) 메소드가 수행된다.

❸ Child 클래스의 show( ) 메소드의 시작점이다.

❹ 화면에 문자열 child를 출력하고, 다음 줄의 처음으로 커서를 이동시킨다. show( ) 메소드가 종료되었으므로 메소드를 호출했던 ❷번의 다음 줄인 ❺번으로 이동하여 프로그램을 종료한다.

결과 child

8. 다음 Java로 구현된 프로그램을 분석하여 그 실행 결과를 쓰시오. (단, 출력문의 출력 서식을 준수하시오.)

```
class Parent {
	int compute(int num) {
		if(num <= 1) return num;
		return compute(num - 1) + compute(num - 2);
	}
}

class Child extends Parent {
	int compute(int num) {
		if(num <= 1) return num;
		return compute(num - 1) + compute(num - 3);
	}
}

public class Test {
	public static void main(String[] args) {
		Parent obj = new Child();
		System.out.print(obj.compute(4));
	}
}
```

답 : 1

[해설]

```
class Parent {                                  클래스 Parent를 정의한다.
	int compute(int num) {
		if(num <= 1) return num;
		return compute(num - 1) + compute(num - 2);
	}
}

class Child extends Parent {            클래스 Child를 정의하고 부모 클래스로 Parent을 지정하면서 Parent에 속한 변수와 메소드를 상속받는다.
❸	int compute(int num) {
❹		if(num <= 1) return num;
❺		return compute(num - 1) + compute(num - 3);
	}
}

public class Test {
	public static void main(String[] args) {
❶		Parent obj = new Child();
```

```
❷              System.out.print(obj.compute(4));
        }
}
```

모든 Java 프로그램은 반드시 main() 메소드에서 시작한다.

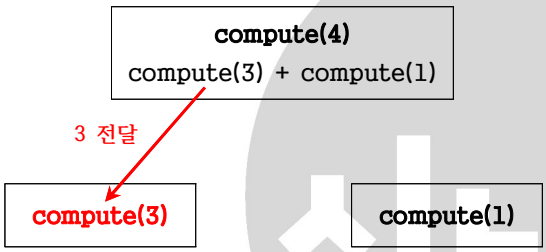
❶ `Parent obj = new Child( );`
클래스 Child로 형 변환이 수행된 클래스 Parent의 객체 변수 obj를 선언한다.

❷ `System.out.print(obj.compute(4));`
obj.compute( )는 obj 객체의 자료형이 Parent이므로 Parent.compute( )라고 생각할 수 있지만 ❶번에서 클래스 형 변환이 발생하였고, compute( ) 메소드가 자식 클래스에서 재정의되었으므로 자식 클래스인 Child의 compute( ) 메소드가 수행된다. 4를 인수로 하여 Child의 compute( )를 호출하고 돌려받은 값을 출력한다.

❸ compute( ) 메소드의 시작점이다. ❷번에서 전달한 값을 정수형 변수 num이 받는다.

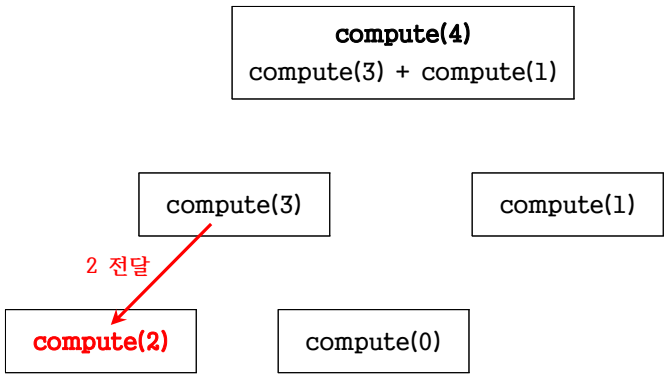
❹ num이 1보다 작거나 같으면 num의 값을 반환하고 메소드를 종료한다. num의 값이 4이므로 ❺번으로 이동한다.

❺ compute(3)을 호출하여 돌려받은 값과 compute(1)을 호출하여 돌려받은 값을 더한 후 반환해야 하므로 먼저 compute(3)을 호출한다.



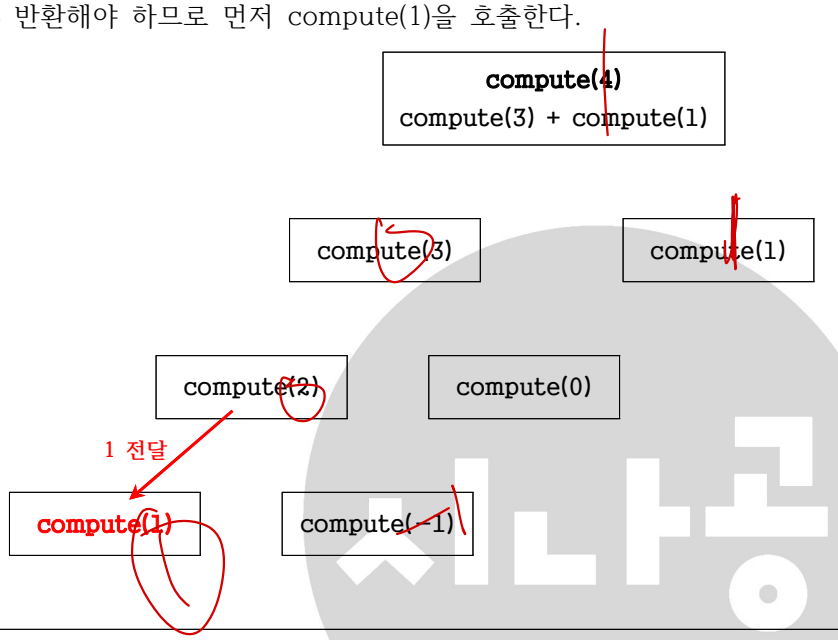

```
class Child extends Parent {
        int compute(int num) {
❻               if(num <= 1) return num;
❼               return compute(num - 1) + compute(num - 3);
        }
}
```

❺번에서 인수로 3이 전달되었으므로 num은 3이다. ❻번에서 num이 1보다 작거나 같지 않으므로 num을 반환하지 않고, ❼번을 수행한다. compute(2)와 compute(0)을 호출하여 돌려받은 값을 더한 후 반환해야 하므로 먼저 compute(2)를 호출한다.



```
class Child extends Parent {
        int compute(int num) {
❽               if(num <= 1) return num;
❾               return compute(num - 1) + compute(num - 3);
        }
}
```

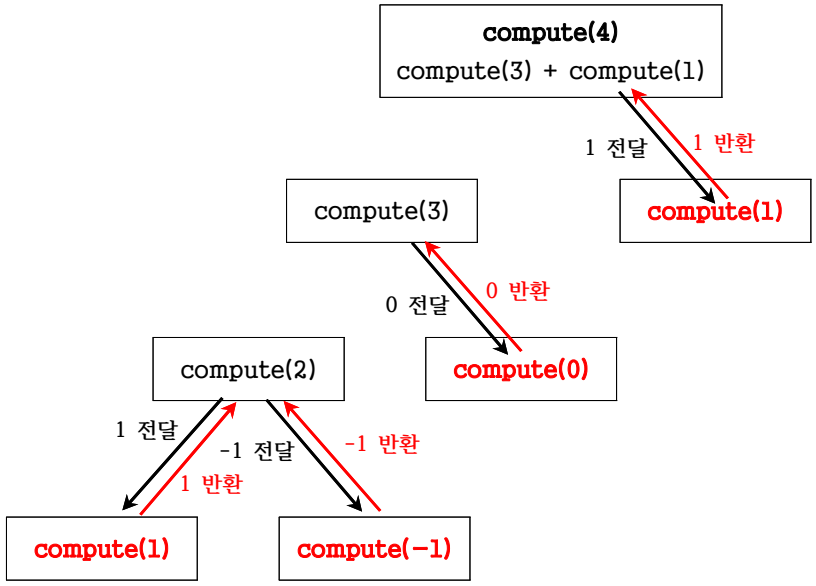
❼번에서 인수로 2가 전달되었으므로 num은 2이다. ❽번에서 num이 1보다 작거나 같지 않으므로 num을 반환하지 않고, ❾번을 수행한다. compute(1)와 compute(-1)을 호출하여 돌려받은 값을 더한 후 반환해야 하므로 먼저 compute(1)을 호출한다.




```
class Child extends Parent {
        int compute(int num) {
❿               if(num <= 1) return num;
                return compute(num - 1) + compute(num - 3);
        }
}
```

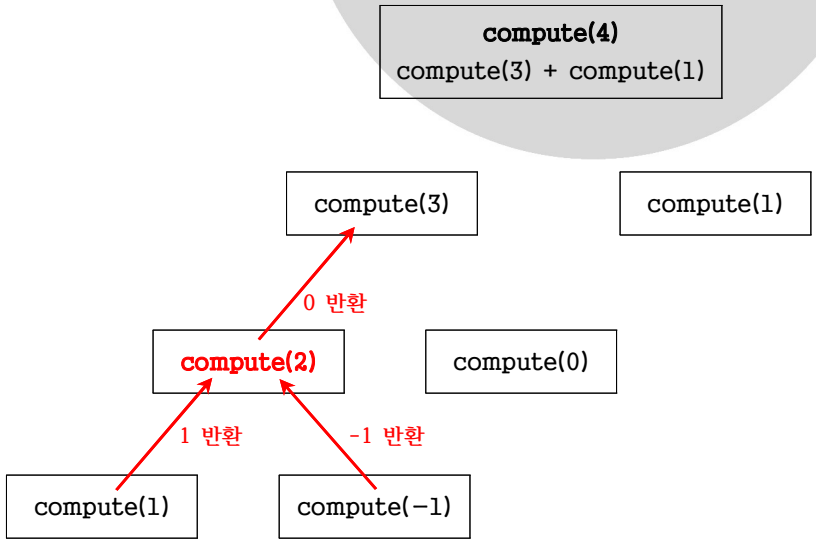
❾번에서 인수로 1이 전달되었으므로 num은 1이다. ❿번에서 num은 1보다 작거나 같으므로 num의 값을 compute(1)을 호출했던 곳으로 반환한다.

※ compute( ) 메소드를 호출할 때 전달되는 인수가 1보다 작거나 같으면 인수의 값(num)을 그대로 반환한다는 것을 알 수 있습니다. 그러면 1보다 작은 값을 인수로 받아 호출되는 모든 compute( ) 메소드의 반환값을 다음과 같이 유추할 수 있습니다.



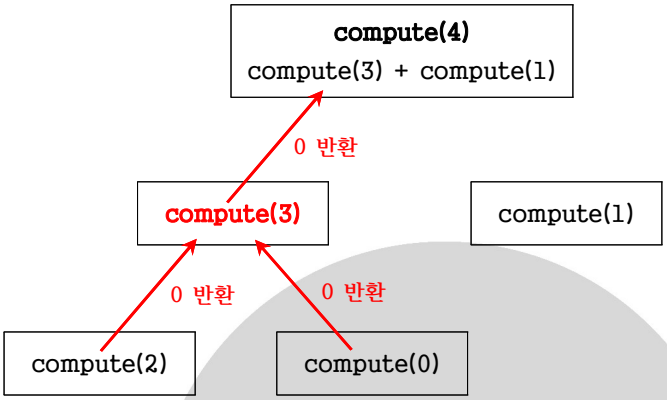

```
class Child extends Parent {
        int compute(int num) {
❽              if(num <= 1) return num;
❾⓫             return compute(num - 1) + compute(num - 3);
        }
}
```

⓫ compute(1)를 호출했던 곳으로 돌아와 compute(1)의 반환값 1과 compute(-1)의 반환값 -1을 더한 0을 compute(2)를 호출했던 곳으로 반환한다.



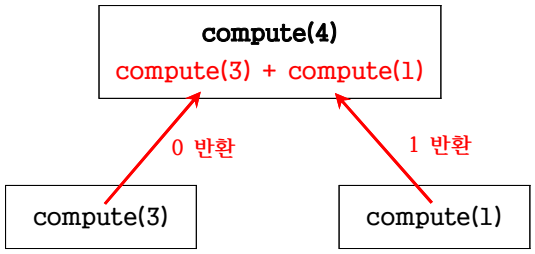
```
class Child extends Parent {
        int compute(int num) {
⑥              if(num <= 1) return num;
⑦⑫             return compute(num - 1) + compute(num - 3);
        }
}
```

⑫ compute(2)를 호출했던 곳으로 돌아와 compute(2)의 반환값 0과 compute(0)의 반환값 0을 더한 0을 compute(3)을 호출했던 곳으로 반환한다.




```
class Child extends Parent {
③       int compute(int num) {
④              if(num <= 1) return num;
⑤⑬             return compute(num - 1) + compute(num - 3);
        }
}
```

⑬ compute(3)을 호출했던 곳으로 돌아와 compute(3)의 반환값 0과 compute(1)의 반환값 1을 더한 1을 가지고 compute(4)를 처음 호출했던 main( ) 메소드로 돌아간다.




```
public class Test {
        public static void main(String[] args) {
①              Parent obj = new Child();
②⑭             System.out.print(obj.compute(4));
        }
}
```

⑭ compute(4)를 호출하고 반환받은 값인 1을 출력하고 프로그램을 종료한다.

결과 1

9. 다음 Java로 구현된 프로그램을 분석하여 그 실행 결과를 쓰시오. (단, 출력문의 출력 서식을 준수하시오.)

```
abstract class Vehicle {
	String name;
	abstract public String getName(String val);
	public String getName() {
		return "Vehicle name : " + name;
	}
}
class Car extends Vehicle {
	private String name;
	public Car(String val) {
		name = super.name = val;
	}
	public String getName(String val) {
		return "Car name : " + val;
	}
	public String getName(byte[] val) {
		return "Car name : " + val;
	}
}
public class Test {
	public static void main(String[] args) {
		Vehicle obj = new Car("Spark");
		System.out.print(obj.getName());
	}
}
```

답 : Vehicle name : Spark

[해설]

```
abstract class Vehicle {      추상 클래스 Vehicle을 정의한다.
	String name;
	abstract public String getName(String val);      추상 메소드 getName(String val)을 정의한다.
❺	public String getName() {
❻		return "Vehicle name : " + name;
	 }
}
class Car extends Vehicle {      클래스 Car를 정의하고 부모 클래스로 Vehicle을 지정하면서 Vehicle에 속한 변수와 메소드를 상속받는다.
	private String name;
```

```
❷       public Car(String val) {
❸               name = super.name = val;
        }
        public String getName(String val) {
                return "Car name : " + val;
        }
        public String getName(byte[] val) {
                return "Car name : " + val;
        }
}
public class Test {
        public static void main(String[] args) {
❶               Vehicle obj = new Car("Spark");
❹❼              System.out.print(obj.getName());
        }
}
```

모든 Java 프로그램은 반드시 main( ) 메소드에서 시작한다.

❶ **Vehicle obj = new Car("Spark");**

Car 클래스의 생성자를 이용하여 Vehicle 클래스의 객체 변수 obj를 선언하고, "Spark"를 인수로 Car 클래스의 생성자를 호출한다.

- **[부모클래스명] [객체변수명] = new [자식클래스생성자( )]** : 부모 클래스의 객체 변수를 선언하면서 자식 클래스의 생성자를 사용하면 형 변환이 발생한다.
- 이렇게 형 변환이 발생했을 때 부모 클래스와 자식 클래스에 동일한 속성이나 메소드가 있으면 자식 클래스의 속성이나 메소드로 재정의된다.

❷ 클래스 Car의 생성자 Car( )의 시작점이다. ❶번에서 전달받은 "Spark"를 val에 저장한다.

❸ **name = super.name = val;**

val의 값 "Spark"를 부모 클래스인 Vehicle 클래스의 변수 name과 Car 클래스의 변수 name에 저장한다. 이어서 Car( )를 호출했던 다음 줄인 ❹번으로 이동한다.

※ **super** : 상속 관계에 있는 부모 클래스를 가리키는 예약어로, 여기서는 Vehicle 클래스를 가리킨다.

❹ 객체 변수 obj의 getName( ) 메소드를 호출한다.

※ 형 변환으로 인해 호출되는 메소드가 Car 클래스의 getName( )이라고 생각할 수 있지만, 메소드의 이름이 동일해도 '인수의 자료형과 개수'가 다르면 서로 다른 메소드이다. 때문에 getName( ) 메소드는 Vehicle 클래스와 Car 클래스의 getName(String val)이나 Car 클래스의 getName(Byte[ ] val) 메소드가 아닌 Vehicle 클래스의 getName( ) 메소드이다.

❺ getName( ) 메소드의 시작점이다.

❻ 문자열 "Vehicle name : "에 변수 name에 저장된 값 "Spark"를 붙여 메소드를 호출했던 ❼번으로 반환한다.

❼ ❻번에서 반환받은 값을 출력하고 프로그램을 종료한다.

결과 Vehicle name : Spark