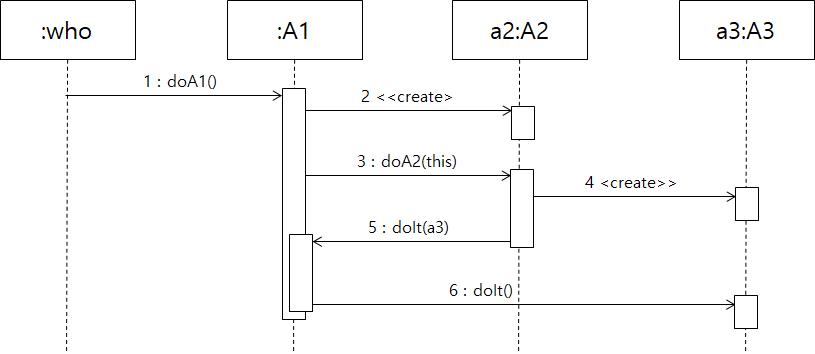
**디자인패턴 실습 5**

|  |  |  |  |
| --- | --- | --- | --- |
| **학번** | B889047 | **이름** | 윤준호 |

1. **다음의 시퀀스 다이어그램을 소스코드와 클래스 다이어그램으로 표현하시오**

**Who -> A1 에서 doA1() : 객체 생성부분이 생략 -> main함수에서 선언하는 등으로 표현하면 됨.**



1. 소스코드

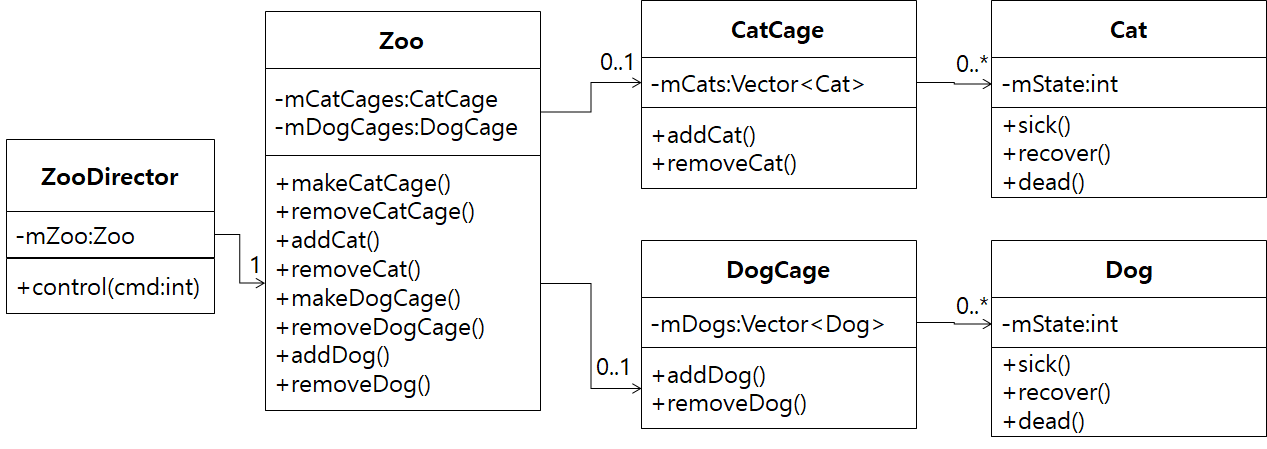
|  |
| --- |
| class who{  public static void main(String[] args){  A1 a1 = new A1();  a1.doA1();  } }  class A1{  public void doA1(){  A2 a2 = new A2();  a2.doA2(this);  }  public void doit(A3 a3){  a3.doit();  } }  class A2{  public void doA2(A1 a1){  A3 a3 = new A3();  a1.doit(a3);  } }  class A3{  public void doit(){  } } |

1. 클래스 다이어그램

|  |
| --- |
|  |

1. **다음의 클래스 다이어그램은 동물원을 추상화 한 구조이다. ZooDirector는 동물원에 동물 우리와 동물들을 추가/제거 할 수 있다. 동물을 추가하려면, 우선 동물 케이지가 존재해야 한다.**

**이 때, ZooDirector가 Zoo를 이용하여 CatCage, DogCage를 하나씩 추가하고, 각각의 Cage에 Cat과 Dog를 한 마리씩 추가하는 과정을 시퀀스 다이어그램으로 표현하시오.**



1. 시퀀스 다이어그램

|  |
| --- |
|  |

소스코드 변환

import java.util.Vector;  
  
class ZooDirector{  
 private Zoo mZoo = new Zoo();  
 public void control(int cmd){  
 switch (cmd){  
 case 1:  
 System.*out*.println("Make Cat Cage");  
 mZoo.makeCatCage();  
 break;  
 case 2:  
 System.*out*.println("Remove Cat Cage");  
 mZoo.removeCatCage();  
 break;  
 case 3:  
 System.*out*.println("Add cat");  
 mZoo.addCat();  
 break;  
 case 4:  
 System.*out*.println("Remove cat");  
 mZoo.removeCat();  
 break;  
 case 5:  
 System.*out*.println("Make Dog Cage");  
 mZoo.makeDogCage();  
 break;  
 case 6:  
 System.*out*.println("Remove Dog Cage");  
 mZoo.removeDogCage();  
 break;  
 case 7:  
 System.*out*.println("Add dog");  
 mZoo.addDog();  
 break;  
 case 8:  
 System.*out*.println("Remove Dog");  
 mZoo.removeDog();  
 break;  
 default:  
 System.*out*.println((new StringBuilder().append("Wrong types of command : ").append(cmd)).toString());  
 }  
 }  
}  
  
class Zoo{  
 private CatCage mCatCages;  
 private DogCage mDogCages;  
 public void makeCatCage(){  
 // 시퀸스 다이어그램 상에서 표현이 안되어있기 때문에, 다이어그램 만으로 구현해야하는 경우에는 넣어서는 안됨(alt문이 있는 경우라면 가능)  
 //  
 /\*  
 if(mCatCages == null){  
 this.mCatCages = new CatCage();  
 }\*/  
 this.mCatCages = new CatCage();  
 }  
 public void removeCatCage(){  
 this.mDogCages = null;  
 }  
 public void addCat(){  
 this.mCatCages.addCat();  
 }  
 public void removeCat(){  
 this.mCatCages.removeCat();  
 }  
 public void makeDogCage(){  
 this.mDogCages = new DogCage();  
 }  
 public void removeDogCage(){  
 this.mDogCages = null;  
 }  
 public void addDog(){  
 this.mDogCages.addDog();  
 }  
 public void removeDog(){  
 this.mDogCages.removeDog();  
 }  
}  
  
class CatCage{  
 private Vector<Cat> mCats = new Vector<>();  
 public void addCat(){  
 mCats.add(new Cat());  
 }  
 public void removeCat(){  
 mCats.remove(mCats.size() - 1);  
 }  
}  
  
class DogCage{  
 private Vector<Dog> mDogs = new Vector<>();  
 public void addDog(){  
 mDogs.add(new Dog());  
 }  
 public void removeDog(){  
 mDogs.remove(mDogs.size() - 1);  
 }  
}  
  
class Cat{  
 private int mState;  
 public void sick(){}  
 public void recover(){}  
 public void dead(){}  
}  
  
class Dog{  
 private int mState;  
 public void sick(){}  
 public void recover(){}  
 public void dead(){}  
}  
  
class test\_t{  
 public static void main(String[] args){  
 ZooDirector zd = new ZooDirector();  
 zd.control(1); // make cat cage  
 zd.control(5); // make dog cage  
 zd.control(3); // add cat to cage  
 zd.control(7); // add dog to cage  
 }  
}