猫狗队列

【题目】

宠物、狗和猫的类如下：

```
public class Pet {
        private String type;

        public Pet(String type) {
                this.type = type;
        }

        public String getPetType() {
                return this.type;
        }
}

public class Dog extends Pet {
        public Dog() {
                super("dog");
        }
}

public class Cat extends Pet {
        public Cat() {
                super("cat");
        }
}
```

实现一种狗猫队列的结构，要求如下：

- 用户可以调用 add 方法将 cat 类或 dog 类的实例放入队列中；
- 用户可以调用 pollAll 方法，将队列中所有的实例按照进队列的先后顺序依次弹出；
- 用户可以调用 pollDog 方法，将队列中 dog 类的实例按照进队列的先后顺序依次弹出；
- 用户可以调用 pollCat 方法，将队列中 cat 类的实例按照进队列的先后顺序依次弹出；
- 用户可以调用 isEmpty 方法，检查队列中是否还有 dog 或 cat 的实例；
- 用户可以调用 isDogEmpty 方法，检查队列中是否有 dog 类的实例；
- 用户可以调用 isCatEmpty 方法，检查队列中是否有 cat 类的实例。

【难度】

士 ★☆☆☆

【解答】

本题考查实现特殊数据结构的能力以及针对特殊功能的算法设计能力。

本题为开放类型的面试题，希望读者能有自己的实现，在这里列出几种常见的设计错误：

- cat 队列只放 cat 实例，dog 队列只放 dog 实例，再用一个总队列放所有的实例。
  错误原因：cat、dog 以及总队列的更新问题。
- 用哈希表，key 表示一个 cat 实例或 dog 实例，value 表示这个实例进队列的次序。
  错误原因：不能支持一个实例多次进队列的功能需求，因为哈希表的 key 只能对应一个 value 值。
- 将用户原有的 cat 或 dog 类改写，加一个计数项来表示某一个实例进队列的时间。
  错误原因：不能擅自改变用户的类结构。

本题实现将不同的实例盖上时间戳的方法，但是又不能改变用户本身的类，所以定义一个新的类，具体实现请参看如下的 PetEnterQueue 类。

```
public class PetEnterQueue {
        private Pet pet;
        private long count;

        public PetEnterQueue(Pet pet, long count) {
                this.pet = pet;
                this.count = count;
        }

        public Pet getPet() {
                return this.pet;
        }

        public long getCount() {
                return this.count;
        }

        public String getEnterPetType() {
                return this.pet.getPetType();
        }
}
```

PetEnterQueue 类在构造时，pet 是用户原有的实例，count 就是这个实例的时间戳。

我们实现的队列其实是 PetEnterQueue 类的实例。大体说来，首先有一个不断累加的数据项，用来表示实例进队列的时间；同时有两个队列，一个是只放 dog 类实例的队列 dogQ，另一个是只放 cat 类实例的队列 catQ。

在加入实例时，如果实例是 dog，就盖上时间戳，生成对应的 PetEnterQueue 类的实例，然后放入 dogQ；如果实例是 cat，就盖上时间戳，生成对应的 PetEnterQueue 类的实例，然后放入 catQ。具体过程请参看如下 DogCatQueue 类的 add 方法。

只想弹出 dog 类的实例时，从 dogQ 里不断弹出即可，具体过程请参看如下 DogCatQueue 类的 pollDog 方法。

只想弹出 cat 类的实例时，从 catQ 里不断弹出即可，具体过程请参看如下 DogCatQueue 类的 pollCat 方法。

想按实际顺序弹出实例时，因为 dogQ 的队列头表示所有 dog 实例中最早进队列的实例，同时 catQ 的队列头表示所有的 cat 实例中最早进队列的实例。则比较这两个队列头的时间戳，谁更早，就弹出谁。具体过程请参看如下 DogCatQueue 类的 pollAll 方法。

DogCatQueue 类的整体代码如下：

```
public class DogCatQueue {
        private Queue<PetEnterQueue> dogQ;
        private Queue<PetEnterQueue> catQ;
        private long count;

        public DogCatQueue() {
                this.dogQ = new LinkedList<PetEnterQueue>();
                this.catQ = new LinkedList<PetEnterQueue>();
                this.count = 0;
        }

        public void add(Pet pet) {
                if (pet.getPetType().equals("dog")) {
                        this.dogQ.add(new PetEnterQueue(pet, this.count++));
                } else if (pet.getPetType().equals("cat")) {
                        this.catQ.add(new PetEnterQueue(pet, this.count++));
                } else {
                        throw new RuntimeException("err, not dog or cat");
                }
        }

        public Pet pollAll() {
                if (!this.dogQ.isEmpty() && !this.catQ.isEmpty()) {
                        if(this.dogQ.peek().getCount()   <   this.catQ.peek().Get
Count()) {
                                return this.dogQ.poll().getPet();
                        } else {
                                return this.catQ.poll().getPet();
```

```
                }
            } else if (!this.dogQ.isEmpty()) {
                return this.dogQ.poll().getPet();
            } else if (!this.catQ.isEmpty()) {
                return this.catQ.poll().getPet();
            } else {
                throw new RuntimeException("err, queue is empty!");
            }
        }

        public Dog pollDog() {
            if (!this.isDogQueueEmpty()) {
                return (Dog) this.dogQ.poll().getPet();
            } else {
                throw new RuntimeException("Dog queue is empty!");
            }
        }

        public Cat pollCat() {
            if (!this.isCatQueueEmpty()) {
                return (Cat) this.catQ.poll().getPet();
            } else
                throw new RuntimeException("Cat queue is empty!");
        }

        public boolean isEmpty() {
            return this.dogQ.isEmpty() && this.catQ.isEmpty();
        }

        public boolean isDogQueueEmpty() {
            return this.dogQ.isEmpty();
        }

        public boolean isCatQueueEmpty() {
            return this.catQ.isEmpty();
        }

    }
```

用一个栈实现另一个栈的排序

【题目】

一个栈中元素的类型为整型，现在想将该栈从顶到底按从大到小的顺序排序，只许申请一个栈。除此之外，可以申请新的变量，但不能申请额外的数据结构。如何完成排序？