

用java 设计一个通用的ADT优先级队列，并用一个通用的ADT堆(min-heap)作为数据结构来实现它。



```
        this.name = name;
        this.grade = grade;
    }

    public String getName() {
        return name;
    }

    public int getGrade() {
        return grade;
    }

    @Override
    public int compareTo(Student other) {
        return Integer.compare(grade, other.grade);
    }
}

public class MinHeapExample {
    public static void main(String[] args) {
        MinHeap<Student> heap = new MinHeap<>(10);

        heap.insert(new Student("Alice", 80));
        heap.insert(new Student("Bob", 90));
        heap.insert(new Student("Charlie", 70));

        while (!heap.isEmpty()) {
            System.out.println(heap.removeMin().getName());
        }
    }
}
```

在这个示例中，我们使用泛型 `T extends Comparable<T>` 来表示元素类型 `T` 必须实现 `Comparable` 接口。在 `swim()` 方法和 `sink()` 方法中，我们直接调用了元素类中的 `compareTo()` 方法来比较元素之间的大小关系。在 `Student` 类中，我们实现了 `Comparable` 接口，并通过 `grade` 字段来定义学生成绩的大小关系。

需要注意的是，在使用 `Comparable` 接口时，堆中的所有元素都必须属于同一个具有可比性的类，即它们应该具有相同的比较规则。否则，将无法保证堆的性质得到满足。

2023/3/29 18:23:23



MinHeap 的capacity 应该传多少



2023/3/29 18:23:34

MinHeap 的 `capacity` 参数取决于具体应用场景和要存储的元素数量。

如果您预先知道将要存储的元素数量，可以将 `capacity` 设置为该值。这样可以避免在向堆中添加元素时，不断地调整数组大小导致的性能开销。但是，如果实际存储的元素数量超过了 `capacity`，则会出现数组溢出的情况。

如果您无法预先确定将要存储的元素数量，可以根据需要动态调整数组大小。例如，在Java中，可以使用 `java.util.ArrayList` 类作为堆的底层数据结构，并在需要时自动增加其大小。

另外，也可以考虑将 `capacity` 设置为一个较大的值，以确保有足够的空间存储所有可能的元素。但这样也会带来一定的空间浪费。

2023/3/29 18:23:48



MinHeap 的capacity 太小会空指针咋办



2023/3/29 18:24:02

如果将 MinHeap 的 `capacity` 参数设置得太小，可能会导致在向堆中添加元素时出现空指针异常（`NullPointerException`）。

如果发生了空指针异常，我们需要检查以下几点：

- 1. 检查堆的 `size` 是否已经达到了 `capacity` 的上限。如果是，则不能再向堆中添加更多的元素，需要删除一些元素或者增加 `capacity`。
- 2. 检查插入的元素是否为 `null`。如果插入的元素为 `null`，则会出现空指针异常。
- 3. 检查 MinHeap 类中的代码逻辑，特别是 `insert()` 方法和相关私有方法（如 `swim()` 和 `sink()`），确保没有出现任何潜在的空指针异常情况。

为了避免空指针异常，建议将 `capacity` 设置为足够大的值，以便能够存储所有可能的元素。另外，在向堆中添加元素之前，最好先对元素进行非空校验，并且在代码中谨慎地处理空元素的情况。



来说点什么...

