Task 1 : Advanced Data Structures

Here is your task

Your task is to implement a novel data structure - your project lead is calling it a power of two max heap. The rest of your team is doing their best to come up with a better name. The requirements of the data structure are as follows:

- The heap must satisfy the heap property.
- Every parent node in the heap must have 2^x children.
- The value of x must be a parameter of the heap's constructor.
- The heap must implement an insert method.
- The heap must implement a pop max method.
- The heap must be implemented in Java.
- The heap must be performant.
- You must use a more descriptive variable name than x in your implementation.

Think carefully about how you implement each method, and manage the underlying data. Performance is critical, so keep cycles and memory usage to a minimum. Be sure to test your heap with very small and very large values of x. As always, keep a weather eye out for sneaky edge cases.

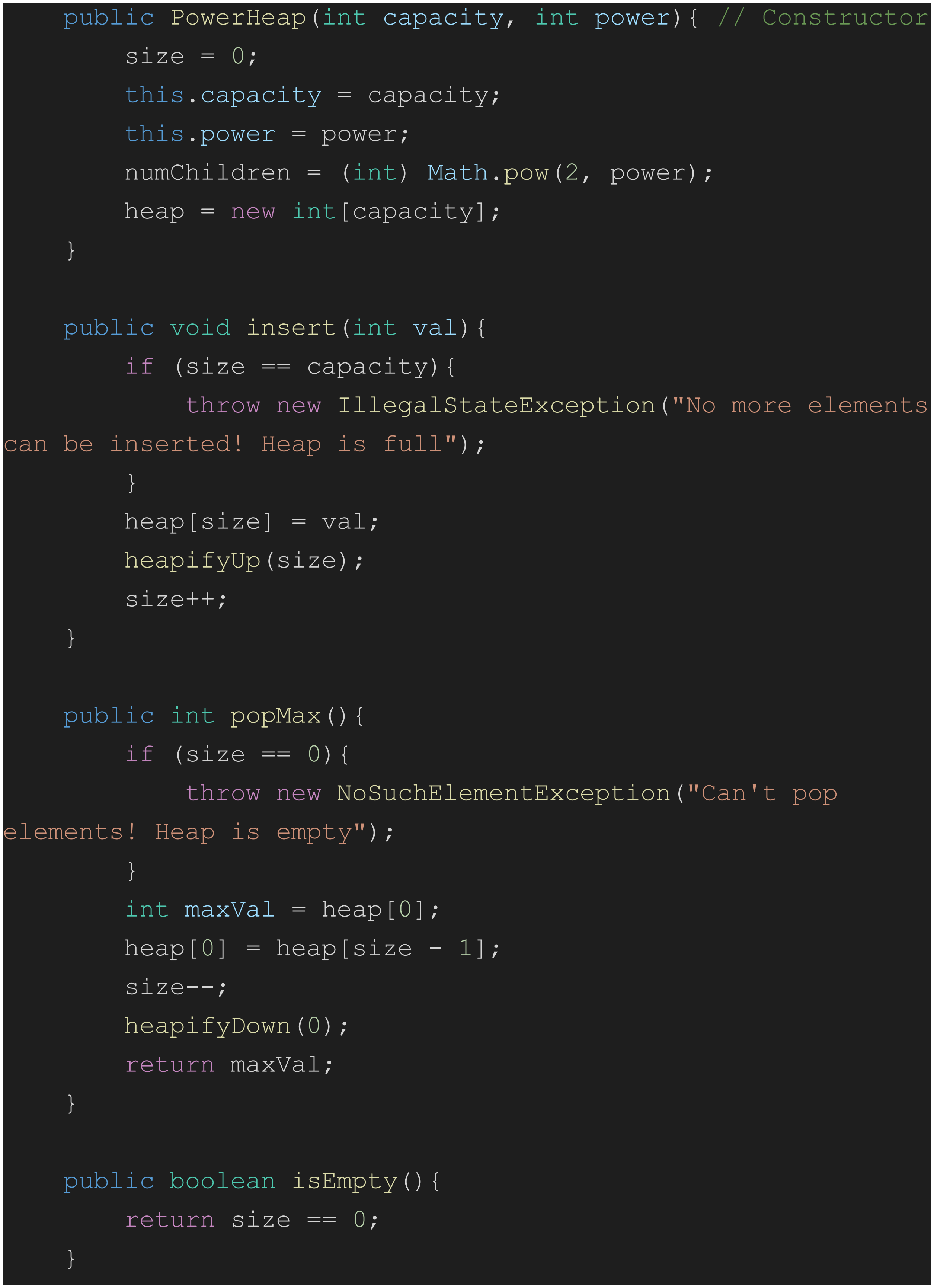
Solution:

```
import java.util.Arrays;
import java.util.NoSuchElementException;

public class PowerHeap{
    private int[] heap;
    private int size;
    private int capacity;
    private int power;
    private int numChildren;

    private void heapifyUp(int index){
        while (index > 0){
            int parentIndex = (index - 1) / numChildren;
            if (heap[index] > heap[parentIndex]){ // Checking
if heap property is satisfied or not
                swap(index, parentIndex);
                index = parentIndex;
            } else{
                break;
            }
        }
    }

    private void heapifyDown(int index){
        while (index < size){
            int largest = index;
            for (int i = 1; i <= numChildren; i++){ // Here
we find the child of parent having maximum value
                int childIndex = numChildren * index + i;
                if (childIndex < size && heap[childIndex] >
heap[largest]){
                    largest = childIndex;
                }
            }
            if (largest != index){
                swap(index, largest); // Swap with the
largest Child Value
                index = largest;
            } else{
                break;
            }
        }
    }
```

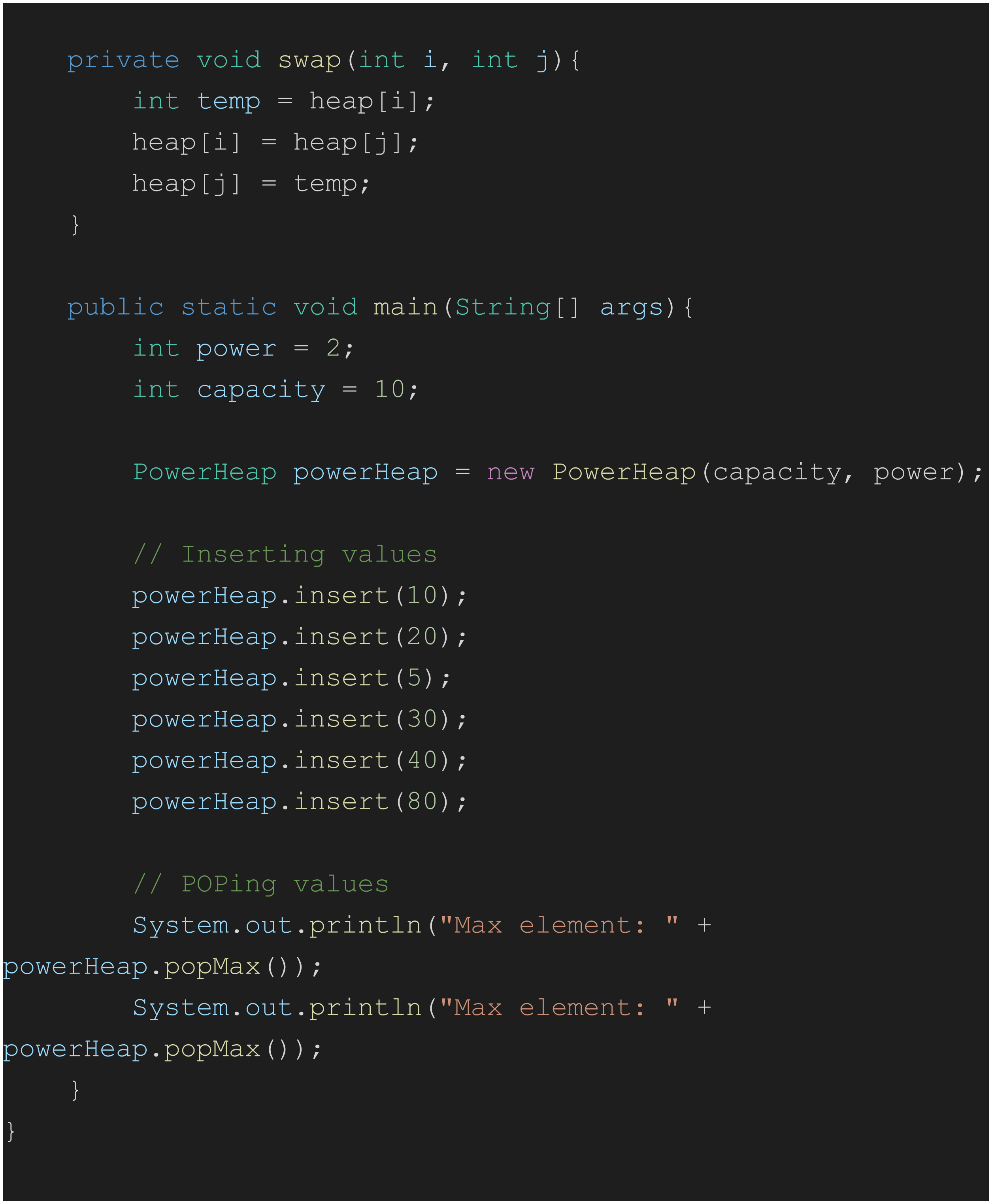


```java
    public PowerHeap(int capacity, int power){ // Constructor
        size = 0;
        this.capacity = capacity;
        this.power = power;
        numChildren = (int) Math.pow(2, power);
        heap = new int[capacity];
    }

    public void insert(int val){
        if (size == capacity){
            throw new IllegalStateException("No more elements can be inserted! Heap is full");
        }
        heap[size] = val;
        heapifyUp(size);
        size++;
    }

    public int popMax(){
        if (size == 0){
            throw new NoSuchElementException("Can't pop elements! Heap is empty");
        }
        int maxVal = heap[0];
        heap[0] = heap[size - 1];
        size--;
        heapifyDown(0);
        return maxVal;
    }

    public boolean isEmpty(){
        return size == 0;
    }
```



```
    private void swap(int i, int j){
        int temp = heap[i];
        heap[i] = heap[j];
        heap[j] = temp;
    }

    public static void main(String[] args){
        int power = 2;
        int capacity = 10;

        PowerHeap powerHeap = new PowerHeap(capacity, power);

        // Inserting values
        powerHeap.insert(10);
        powerHeap.insert(20);
        powerHeap.insert(5);
        powerHeap.insert(30);
        powerHeap.insert(40);
        powerHeap.insert(80);

        // POPing values
        System.out.println("Max element: " +
powerHeap.popMax());
        System.out.println("Max element: " +
powerHeap.popMax());
    }
}
```