Hanoi.java

```
package hanoi;

/**
 * Towers of Hanoi
 *
 * @author tadaki
 */
public class Hanoi {

    private Pillar pillars[];//3本の柱
    private final int n;//円盤の数
    private int count = 0;//円盤移動の回数
    private boolean debug = false;
    private String nl = System.getProperty("line.separator");

    /**
     * コンストラクタ
     *
     * @param n 円盤の数
     * @throws java.lang.Exception
     */
    public Hanoi(int n) throws Exception {
        this.n = n;
        //3本の柱を初期化
        pillars = new Pillar[3];
        for (int i = 0; i < 3; i++) {
            pillars[i] = new Pillar();
        }
        //0番の柱にn枚の円盤を設置
        //下に大きな番号の円盤を置く
        for (int i = n - 1; i >= 0; i--) {
            pillars[0].push(new Disk(i));
        }
        count = 0;
    }

    /**
     * 移動開始
     *
     * @throws java.lang.Exception
     */
    public void start() throws Exception {
        moveDisks(0, 2, n);
    }

    /**
```

```
     * move disks from to
     *
     * @param from 移動元
     * @param to 移動先
     * @param number 移動枚数
     * @throws Exception
     */
    private void moveDisks(int from, int to, int number)
            throws Exception {
        if (number == 1) {
            moveSingleDisk(from, to);
            return;
        }
        int o = 3 - (from + to); //other pillars
        moveDisks(from, o, number - 1);
        moveSingleDisk(from, to);
        moveDisks(o, to, number - 1);
    }

    /**
     * move single disk from to
     *
     * @param from 移動元
     * @param to 移動先
     * @throws Exception
     */
    private void moveSingleDisk(int from, int to)
            throws Exception {
        Disk d = pillars[from].pop();
        pillars[to].push(d);
        count++;
        if (debug) {
            showState();
        }
    }

    /**
     * 現在の情報を出力
     *
     */
    private void showState() {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < 3; i++) {
            sb.append(i).append("->")
              .append(pillars[i].getState()).append(nl);
        }
```

```
        System.out.println(sb.toString());
    }

    public boolean isDebug() {
        return debug;
    }

    public void setDebug(boolean debug) {
        this.debug = debug;
    }

    public int getCount() {
        return count;
    }

    static public void main(String args[]) throws Exception {
        Hanoi hanoi = new Hanoi(10);
        hanoi.setDebug(true);
        hanoi.start();
        System.out.println("移動回数: " + String.valueOf(hanoi.getCount()));
    }
}
```

Pillar.java

```java
package hanoi;

import java.util.Stack;

/**
 * Hanoiの塔の一つの柱
 *
 * @author tadaki
 */
public class Pillar {

    Stack<Disk> pillar;

    public Pillar() {
        pillar = new Stack<>();
    }

    /**
     * 円盤を一つ追加
     *
     * @param d 追加する円盤
     * @return
     * @throws Exception
     */
    public Disk push(Disk d) throws Exception {
        if (!canPush(d)) {//適切でない円盤を追加する場合に例外を投げる
            throw new Exception("prohibited operation");
        }
        return pillar.push(d);
    }

    /**
     * 一番上の円盤を取り出す
     *
     * @return 取り出した円盤
     */
    public Disk pop() {
        if (pillar.isEmpty()) {
            return null;
        }
        return pillar.pop();
    }

    /**
     * 円盤を追加できるかを調べる
     *
```

Pillar.java

```
     * @param d 追加しようとしている円盤
     * @return 追加できなならばtrue
     */
    public boolean canPush(Disk d) {
        boolean f = true;
        if (!pillar.isEmpty()) {
            Disk top = pillar.peek();
            if (d.compareTo(top) > 0) {
                f = false;
            }
        }
        return f;
    }

    /**
     * 現在のpillarの状態を文字列で返す
     *
     * @return
     */
    public String getState() {
        StringBuilder sb = new StringBuilder();
        pillar.stream().forEachOrdered(
                d -> {
                    sb.append(d).append(" ");
                });
        return sb.toString();
    }

}
```

Disk.java

```java
package hanoi;

/**
 * Hanoiの塔の円盤のクラス
 * @author tadaki
 */
public class Disk implements Comparable {

    private final int n;//円盤に付与された番号

    /**
     * コンストラクタ
     * @param n 円盤に付与する番号
     */
    public Disk(int n) {
        this.n = n;
    }

    /**
     * 他の円盤との番号の比較
     * @param o 比較対象
     * @return 自分の番号が小さければ負、同じならば0、
     *         大きいならば正の数を返す
     */
    @Override
    public int compareTo(Object o) {
        Disk target = (Disk) o;
        return n-target.n;
    }

    @Override
    public String toString() {
        return String.valueOf(n);
    }
}
```