Exercise 1: Operational Semantics

Consider the following class

```java
class IntWrapper {
    private int val;
    public IntWrapper(int val) {
        this.val = val;
    }
    public void inc() {
        this.val = this.val + 1;
    }
}
```

Use the rules for the operational semantics of Java to compute the result of the expression `((new IntWrapper(4)).inc())`. For simplicity, we ignore the implicit call of the constructor of the super class, and start in the state `(Norm, ∅, ∅)`.

Exercise 2: Map implementation

On the lecture’s webpage you find the skeleton for a Map data structure that implements functions mapping keys to values using sorted binary trees. Your task is to implement and specify this data structure. The specification should be given in terms of a model field of type `JMLValueToObjectMap`, so make yourself familiar with the methods provided by this class.

(a) Implement and specify a method

```java
private /*@ pure @*/ JMLValueToObjectMap computeContent(Node n)
```

that computes the content of the binary tree.

(b) Implement and specify a method

```java
public /*@ pure @*/ boolean inDomain (Key k)
```

that checks whether there exists a node in the tree whose key is equal to `k`.

(c) Implement and specify a method

```java
public Object add(Key k, Object v)
```

that inserts a key/value pair into the tree. The tree should remain sorted. If a node with
that key already exists, the old mapping will be replaced. The methods return
null if the key was not mapped to a value before this method has been called,
and the old value otherwise.

(d) Write a small test program that creates an instance of your map and inserts some
key/value pairs where keys are of type IntKey. Compile and run your program with
the JML tools.

(e) **Bonus:** Give a class invariant that states sortedness of the tree. Test your invariant
with your example program.

*Hint:* You need to specify a property about all nodes in the tree. There are multiple
ways to do that, but all need at least one pure function.