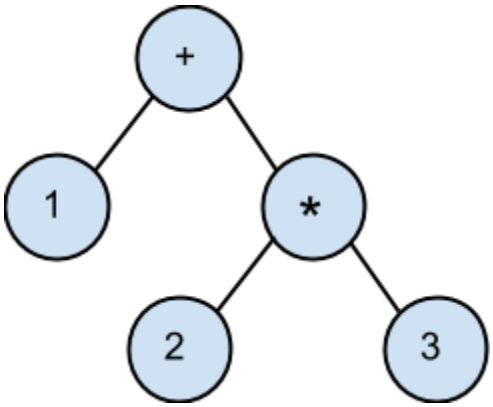
სემინარი 6. პოლიმორფიზმი if კონსტრუქციების ნაცვლად

სავარჯიშო

არითმეტიკული გამოსახულება წარმოადგინეთ ხის სახით და შექმენით მონაცემთა სტრუქტურა ხის წვეროების შესანახად.

მაგალითი: 1 + 2 * 3



თითოეულ კვანძს ჰქონდეს მინიმუმ ორი მეთოდი:

`double evaluate()` - დააბრუნოს გამოთვლილი მნიშვნელობა კვანძისთვის

`String toString()` - დააბრუნოს სტრიქონის სახით ჩაწერილი გამოსახულება გამოთვლის გარეშე, ფრჩხილებიც დასვას. სტრიქონი უნდა მოიცავდეს კვანძის ქვეხეში არსებულ სხვა გამოსახულებებსაც, მაგალითად (1 + (2 * 3))

ვარიანტი 1

```java
public class Node {
    char operator;
    double value;
    Node left;
    Node right;

    public double evaluate(){
        switch (operator) {
            case '#': return value;
            case '+': return left.evaluate() + right.evaluate();
            case '*': return left.evaluate() * right.evaluate();
            // add new operators here
        }
        return 0;
    }

    @Override
    public String toString() {
        switch (operator) {
            case '#': return value + "";
            case '+': return "(" + left.toString() + " + " + right.toString() + ")";
            case '*': return "(" + left.toString() + " * " + right.toString() + ")";
            // add new operators here
        }
        return null;
    }
}
```

ვარიანტი 2

```java
public abstract class Node {
    public abstract double evaluate();
}
class ValueNode extends Node {
    double value;

    @Override
    public double evaluate() {
        return value;
    }

    @Override
    public String toString() {
        return value + "";
    }
}

class OpNode extends Node {
    char operator;
    Node left;
    Node right;

    @Override
    public double evaluate() {
        switch (operator) {
            case '+': return left.evaluate() + right.evaluate();
            case '*': return left.evaluate() * right.evaluate();
            // add new operators here
        }
        return 0;
    }

    @Override
    public String toString() {
        switch (operator) {
            case '+': return "(" + left.toString() + " + " + right.toString() + ")";
            case '*': return "(" + left.toString() + " * " + right.toString() + ")";
            // add new operators here
        }
        return null;
    }
}
```

ვარიანტი 3

```java
public abstract class Node {
    public abstract double evaluate();
}

class ValueNode extends Node {
    double value;

    @Override
    public double evaluate() {
        return value;
    }

    @Override
    public String toString() {
        return value + "";
    }
}

abstract class OpNode extends Node {
    Node left;
    Node right;
}

class AdditionNode extends OpNode {
    @Override
    public double evaluate() {
        return left.evaluate() + right.evaluate();
    }

    @Override
    public String toString() {
        return "(" + left.toString() + " + " + right.toString() + ")";
    }
}


class MultiplicationNode extends OpNode {
    @Override
    public double evaluate() {
        return left.evaluate() * right.evaluate();
    }

    @Override
```

```
    public String toString() {
        return "(" + left.toString() + " * " + right.toString() + ")";
    }
}
```