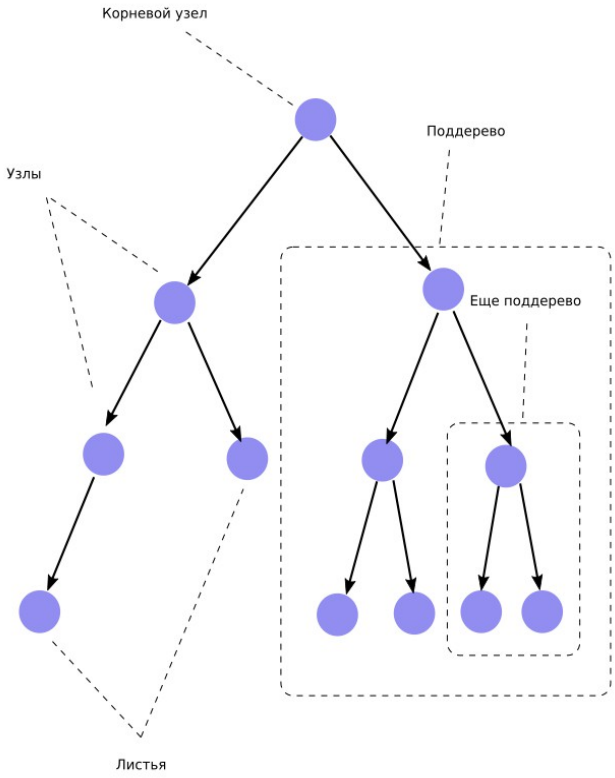
# 1. Дятел-отладчик

Задание выполняется в тренажере номер 2. Альтернативно, для любознательных*, можно сделать Android-приложение с кнопками и подходящим UI.

Деревом в программировании называется рекурсивная структура данных, состоящая из одного или нескольких узлов (node). Каждый узел дерева может иметь одну или несколько ссылок на другие узлы, из которых, соответственно, могут расти “поддеревья”. Конечные узлы, которые не имеют ссылок на поддеревья, называются листьями.

Поддеревья, как правило, не имеют прямой ссылки на узел, из которого они растут. Глубина вложенности поддеревьев не ограничена. Обратите внимание, что дерево не обязано быть симметричным.

Дерево, в котором каждый узел имеет ссылки не более чем на два поддерева, называется бинарным. Обычно также бинарное дерево предполагает определенную сортировку значений в узлах дерева, но для нашей задачи это не актуально.




Итак, имеется дерево произвольной формы. В узлах дерева живут жучки (bugs) - в каждом узле от 0 до 10 жучков.

Дятел-отладчик (debugger) ползает по дереву в поисках жучков. Дятел начинает поиск жучков с корневого узла, далее двигается в левое или правое поддерево, оттуда - в следующее поддерево, и так далее. По мере нахождения жучков, дятел их кушает. Добравшись до листа (или ранее, при желании), дятел слетает с дерева и возвращается обратно к корню дерева, где начинает поиск заново (выбирая другой маршрут, при желании). Дятел не может спуститься только на одну-две ветки “вниз” (в сторону корня) - он может двигаться по веткам только в сторону листьев, либо слететь сразу в самый низ, к корню.

Задание:

Создать класс, обозначающий узел дерева.

```
class TreeNode{
   private int bugsCount; //Кол-во жучков, оставшихся в этом узле
   private TreeNode leftSubTree; //Ссылка на левое поддерево
   private TreeNode rightSubTree; //Ссылка на правое поддерево

   //Координаты узла на рисунке. Используются только для облегчения рисования дерева на экране, никак не используются дятлом
   private int x;
   private int y;
}
```

Создать класс, обозначающий дятла

```
class Woodpecker {
   private int bugsEatenCount; //Кол-во съеденных дятлом жучков
   private TreeNode currentPosition; //Текущее местонахождение дятла
}
```

В вышеупомянутых классах запрещается создавать поля иные чем описано в задании. При этом разрешается создавать произвольные методы на усмотрение студента

В классе Program собрать дерево из 10-15 узлов, создать поля “дерево” и “дятел”:

```
TreeNode root; //Дерево
Woodpcker woodpecker; //Дятел
```

Обратите внимание, что мы не создаем отдельный класс для дерева - дерево целиком в нашей задаче вполне однозначно определяется его корневым узлом - зная корневой узел, можно “по ссылкам” найти все остальное.

Написать следующую программу:

- Программа должна нарисовать дерево и дятла на нем (дятел рисуется схематично, навык художника-орнитолога не требуется). При этом нужно нарисовать узлы дерева и связи между ними. Также для каждого узла нужно нарисовать количество жучков, которые там еще остались, а для дятла - нарисовать количество съеденных им жучков.

- Нажатие кнопок влево-вправо должно приводить к тому, что дятел будет переползать на левое/правое поддерево.

- Нажатие кнопки A (например - при желании можно выбрать для этой цели кнопку B), должно приводить к тому, что дятел съест одного жучка из того узла, на котором он сейчас сидит.

- Нажатие кнопки вверх (например - при желании можно выбрать для этой цели кнопку вниз), должно приводить к тому, что дятел слетает в дерева обратно к его корню

- Каждое действие должно сопровождаться обновлением актуальной картины мира на экране.

## 2*

Избавиться от x и y в классе TreeNode (допускается перенос их в какие-то другие классы) без ущерба для функционала и красоты кода.

## 3*

Добавить в класс TreeNode поле

```
   private TreeNode parentNode; //Ссылка на узел, из которого растет текущий
узел
```

Используя это поле, обеспечить возможность дятлу возвращаться “назад” не только сразу к корню дерева, но и к предыдущему узлу.

Реализовать методы класса TreeNode таким образом, чтобы при любых обстоятельствах (даже в случае ошибки программиста) при сборке дерева обеспечивалась его защита от некорректной сборки: Поле parent в дочернем узле и поля leftSubTree или rightSubTree в родительском узле должны корректно соответствовать друг другу во всех случаях.