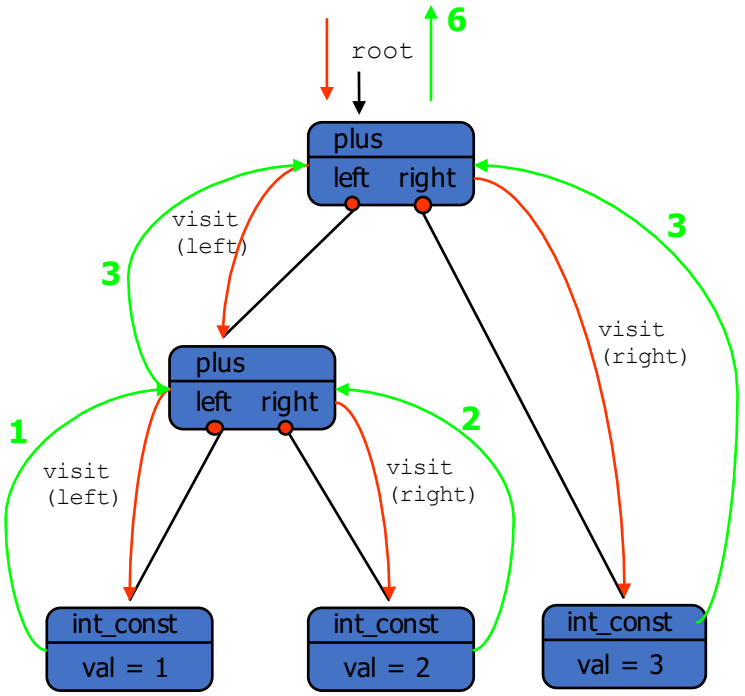
- Ogni nodo dell'albero è istanza di una classe (in questo esempio: plus e int_const)
- Ognuna di queste classi contiene: un metodo accept, uguale per tutti, ed i propri dati
- Per ogni particolare visita dell'albero va creata una classe che implementa Visitor
- In questo esempio si è creata una visita che valuta l'albero (classe ExprEvalVisitor). Si potrebbe creare un'altra visita che stampa l'albero, ad es. class PrintTreeVisitor, e tante altre, ma le classi plus and int_const resterebbero inalterate.

# Pattern visitor on trees




```
class plus extends Expression {
public:
    Object accept(Visitor v) {
     return v.visit(this);
   }
   Expression left, right;
}
```

```
class int_const extends Expression {
public:
    Object accept(Visitor v) {
     return v.visit(this);
   }
   int val;
}
```

```
class ExprEvalVisitor implements Visitor {
public Object visit(plus e) {
    int leftVal=
     ((Integer)e.left.accept(this))).intValue();
    int rightVal=
     ((Integer)e.right.accept(this)).intValue();
    return leftVal + rightVal;
  }
  Object visit(int_const e) {
    return e.val;
  }
  ...
}
```

- La classe che implementa la visita ha un metodo visit polimorfo per ciascun tipo di nodo. Il metodo visit prende in input un nodo compie l'operazione da svolgere per quel nodo (in modo preorder, inorder o postorder) e lancia accept sui suoi figli

La visita si lancia in questo modo:

```
ExprEvalVisitor ev = new ExprEvalVisitor();
Integer result = (Integer)root.accept(ev);
```

(alternative: let accept do tree walking)