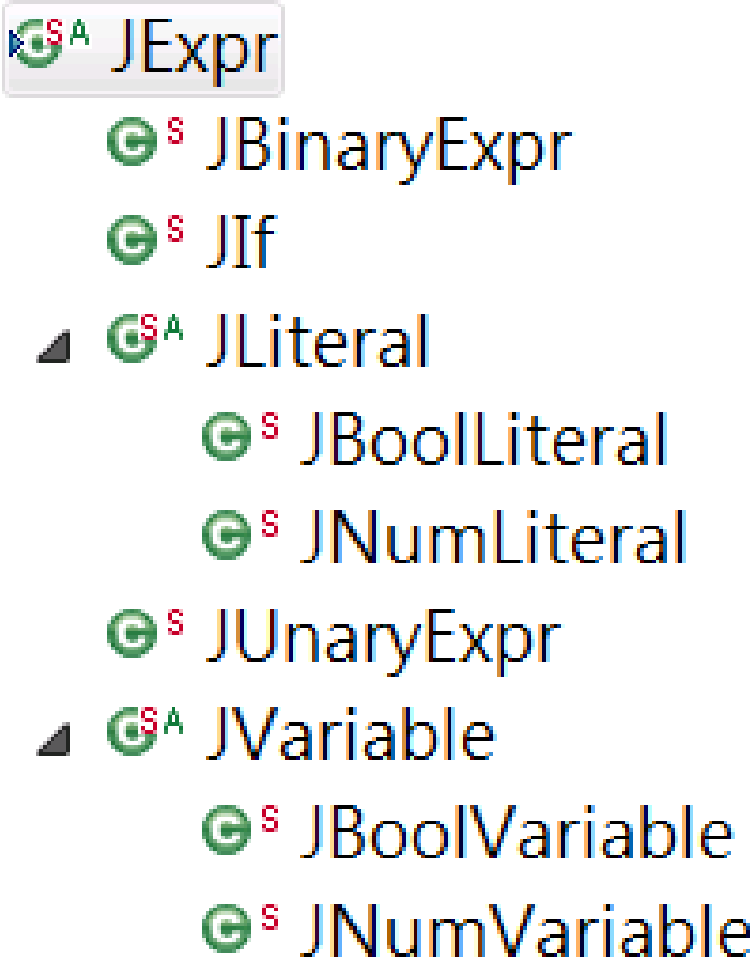
# JP2016

# HW3

The Representations and Analysis of Arithmetic and Logical Expressions

# The Expression Hierarchy in HW3

- inside hw3.HW3



# hw3.HW3.JValue

```
public static abstract class JValue {}

public static class JBoolValue extends JValue {
    public boolean v ;
    public JBoolValue(boolean v){
        this.v = v ;
    }
    public String toString(){return ""+v;}
}

public static class JNumValue extends JValue {
    public double v ;
    public JNumValue(double v){ this.v = v ;}

    public String toString(){return ""+v;}
}
```

# hw3.HW3.JType

```
public static class JType {
    String name;

    private JType(String n) {
        name = n;
    }

    public final static JType JBOOLEAN = new JType("boolean");
    public final static JType JNUMBER = new JType("number") ;

    public String toString() {return name;}

}
```

# Some Examples

```
// e1 = 5
JExpr e1 =  JNUM(5);

// e2 = x + y
JExpr e2 = new JBinaryExpr(ADD, new JNumVariable("x"), new JNumVariable("y"));

// e3 = B * B - 4 * A * C
JExpr e3 = JSUB(
            JMUL(JVAR("B"), JVAR("B")),
            JMUL(JLIT(4), JMUL(JVAR("A"), JVAR("C")))
            );

// e4 = x + y * 2

JExpr e4 = JADD(JVAR("x"), JMUL(JVAR("y"), JNUM(2))) ;
```

# Some Examples

```
// e5 = !(x > y + 4 ) ?  b1 : b2 XOR b3
JExpr e5 = JIF(
            JNOT( JGT(JVAR("x"), JADD(JVAR("y"), JNUM(4))) ),
            JVAR("b1"),
            JXOR( JVAR("b2"), JVAR("b3")));

// e6 = !(x > y + 4 ) ?  3 : b2 XOR b3  *** not wellType!
    JExpr e6 = JIF(
                JNOT( JGT(JVAR("x"), JADD(JVAR("y"), JNUM(4))) ),
                JNUM(3),
                JXOR( JVAR("b2"), JVAR("b3")));

// e7 = x + b1 * true   *** not wellType!

JExpr e7 = JADD(JVAR("x"), JMUL(JVAR("b1"), JLIT(true))) ;
```

```
// e8 = b1 * true ? x + z : x / false   *** not wellType!

JExpr e8 = JIF( JMUL(JVAR("b1"), JLIT(true)),
                JADD(JVAR("x"), JVAR("z")),
                JVAR("b3")
              );

// e9 = e5 * e4   *** not wellType!

        JExpr e9 = JMUL(e4, e5);


// e10 = e5? e4 : e2

JExpr e10 = JIF(e5, e4, e2);
```

# To-DO List

```java
    /**
     * Given a JExpr expr, find all variables occurring in it.
     * Note that no duplicated variables are allowed in the result list.
     * @param expr a wellTyped JExpr.
     * @return
     */
    public static List getVariables(JExpr expr){
```

```java
/**
 * Given a JExpr expr which is assumed to be well-typed, find the type of expr.
 * @param expr a wellTyped JExpr.
 * @return
 */
public static JType getType(JExpr expr){
    // put you code here!
    // cases you need to check :
    // 1. expr = e1 op e2 => return JBOOLEAN if op is logical or comparison op and return JNUMBER otherwie
    // 2. expr = op e1    => return JBOOLEAN if op is ! and JNUMBER if op is -.
    // 3. expr is a literal => return JBOOLEAN if expr is a booealn and JNUMBER if it is a number.
    // 4. expr is a variable => return JBOOLEAN if expr is a boolean variable or JNUMBER otherwise.
    // 5. expr is e?e1:e2 => return the type of e1 (or e2).
    return null ;
}
```

# To-DO List

```
/**
 * Check if the input argument expression is well-typed.
 *
 * A JExpr is wellTyped if is has no type errors.
 * <ul> Well-typed examples:
 * <li> 3 + 5;
 * <li> 4 > x ? 4 : 6;
 * <li> x == 4 ? true : 4 < 3;
 * <li> 4 != 4 || false == true ;
 * </ul>
 * <ul> Ill-typed examples:
 * <li> 3 +  true;
 * <li> 4 > x ? 6 : false;
 * <li> true && 5
 * </ul>
 *
 * @param expr
 * @return
 */
public static boolean isWellTyped(JExpr expr){
    // put you code here!
    return false ;
}
```

# To-DO List

```
/**
 * Given a well-typed JExpr expr, find the value of this expr by evaluating it.
 * Examples:
 *  4 + 5        => return 9
 *  true ? 4 : 7     => return 4
 *  false ? false : true     => return true
 *  x + 5        => return varMap.get(x) + 5, where varMap is the the static map
 *                  value of all variables.
 * @param expr a wellTyped JExpr.
 * @return
 *
 */
public static JValue getValue(JExpr expr){
    // put you code here!
    return null ;
}
```

# To-DO List

```
/**
 *  We can use JExpr.toString() to get a string representation
 *  of an expression. Following are some examples output: (3 + 4), x,
 *  (3>4 ? 3 : (4+5)), ((3 + 4) * 5), ((4*5)+ 3)). It is noted that some parenthesis
 *  '(', ')' in the output are necessary like (3+4) * 5, while others like those in
 *  ((4*5)+3) are not needed [4*5+3 is our expected result].
 *  This method require you to generate a string representation of the input JExpr like those produced by JExpr.toString()
 *      *
 *  The requirement is that all unnecessary parentheses '(',')' should be appear in the output.
 *
 *  Assumptions:
 *  <ul>
 *  <li> Precedences of operations are given as follows:
 *      <ul>
 *      <li> 1. unary op > binary op > ternary (?:).
 *      <li> 2. (*,/) > (+,-) > (<,<=,>,>=) > (==,!=) > (OR, XOR) > AND </li>
 *      </ul> The precedence of each operation is given in the array JExpr.PRECEDENCEand can be queried by
 *        JExpr.prec(op).
 *
 *  <li>Assume all operations are left-associative. i.e. e1 o e2 o e3 means ((e1 o e2) o e3).
 *  </ul>
 *  Note You may add additional methods/fields to HW3 for your purpose provided they do not override existing ones.
 */
public static String prettyString(JExpr expr) {
    return null ;
}
```