Expression Evaluation

Expression Evaluation

- Works for **real** numbers: 1.2, π, e, -sqrt(2), …
- Exp4j project
  - You can get it from **Maven central**
- Here's how to use:

1. Define the expression as a string
2. Define the variables map
3. Set the variable-value
4. Evaluate the expression

- MathUtils class
- **MathUtils.evaluate**(String, Map<String, Double>)
- **MathUtils.evaluate**(String)
  - Example: string = "1.2^3 * pi"

https://search.maven.org/

Apache Maven
maven.apache.org

```
<dependency>
  <groupId>net.objecthunter</groupId>
  <artifactId>exp4j</artifactId>
  <version>0.4.8</version>
</dependency>
```

Maven dependency for Exp4j

Expression Evaluation

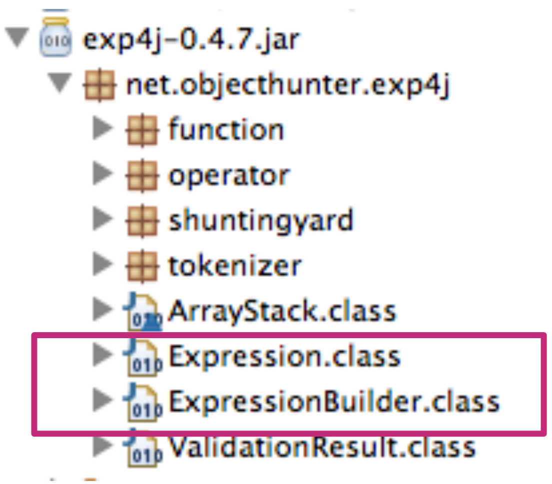
- **Exp4j** project
- **Here's a java code snippet**

`String expression = …`	**Define string expression, e.g. "x^2-sin(x)*y"**
`Map<String, Double> vars = …`	**Define variable-value map**
`ExpressionBuilder eb = new ExpressionBuilder(expression);`	**Create the builder**
`eb.variables(vars.keySet());`	**Set the variables of expression from the map**
`Expression ex = eb.build();`	**Build the expression**
`ex.setVariables(vars);`	**Set the values of the variables**
`double result = ex.evaluate();`	**Evaluate the expression and get the result**

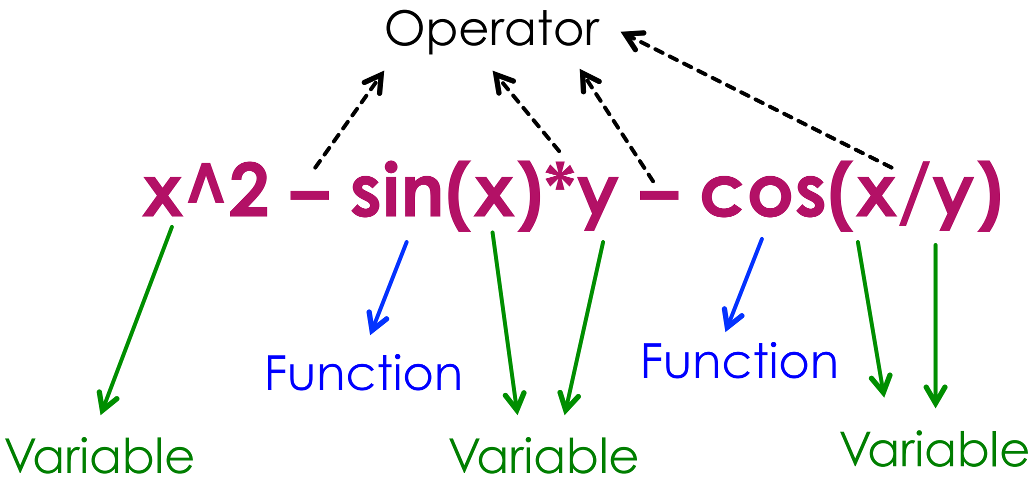
Tip: you can chain all the lines together

Expression Evaluation

- **How does it work?**
  - Based on **shunting-yard algorithm**
  - Operators and functions are pre-defined
  - Variables are user defined
    - ✓ Tokenizer



Main classes we use



Expression Evaluation

- **How does it work?**
  - Based on **shunting-yard algorithm**
  - Operators and functions are pre-defined

- exp4j-0.4.7.jar
  - net.objecthunter.exp4j
    - function
    - operator
    - shuntingyard
    - tokenizer
    - ArrayStack.class
    - Expression.class
    - ExpressionBuilder.class

Main classes we use

Shunting-yard algorithm **(from Wikipedia)**

From Wikipedia, the free encyclopedia

This article includes a list of references, related reading or external links, **but its sources remain unclear because it lacks inline citations**. Please help to improve this article by introducing more precise citations. *(August 2013) (Learn how and when to remove this template message)*

In computer science, the **shunting-yard algorithm** is a method for parsing mathematical expressions specified in infix notation. It can produce either a postfix notation string, also known as Reverse Polish notation (RPN), or an abstract syntax tree (AST). The algorithm was invented by Edsger Dijkstra and named the "shunting yard" algorithm because its operation resembles that of a railroad shunting yard. Dijkstra first described the Shunting Yard Algorithm in the Mathematisch Centrum report MR 34/61.

Like the evaluation of RPN, the shunting yard algorithm is stack-based. Infix expressions are the form of mathematical notation most people are used to, for instance "3 + 4" or "3 + 4 × (2 − 1)". For the conversion there are two text variables (strings), the input and the output. There is also a stack that holds operators not yet added to the output queue. To convert, the program reads each symbol in order and does something based on that symbol. The result for the above examples would be (in Reverse Polish notation) "3 4 +" and "3 4 2 1 − × +", respectively.

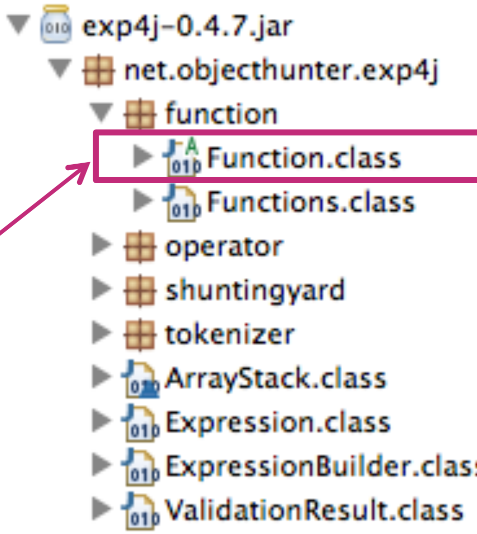
The shunting-yard algorithm was later generalized into operator-precedence parsing.

Contents [hide]

https://en.wikipedia.org/wiki/Shunting-yard_algorithm

Expression Evaluation

- **How to add your custom-defined function?**
  - Operators and functions are pre-defined
  - You can extend "Function" class



Function Class

Define your custom function class

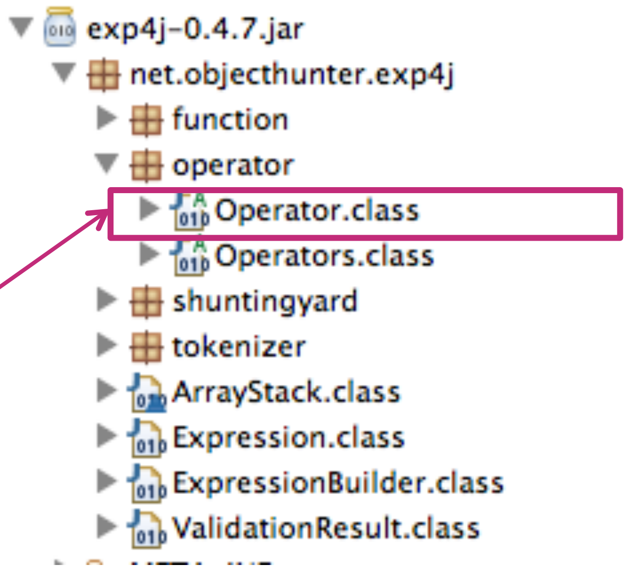
```
public class CustomFunction extends Function {

    public CustomFunction(String name) {
        super(name);
    }

    @Override
    public double apply(double... args) {
        return Math.pow(args[0], 2.0);
    }

}
```

vararg

```
CustomFunction func1 = new CustomFunction("my_func") ;
String s = "x + my_func(x)" ;
ExpressionBuilder eb = new ExpressionBuilder(s) ;
eb.variables("x") ;
eb.functions(func1) ;
Expression ex = eb.build() ;
ex.setVariable("x", 2.0) ;
double result = ex.evaluate() ;
```

Expression Evaluation

- **How to add your custom-defined operator?**
  - Operators and functions are pre-defined
  - You can extend “Operator” class
  - Example: “**” is raising to some power in python: a**2 = a^2



Operator Class

Define your custom operator class

```java
public class CustomOperator extends Operator {

	public CustomOperator(String symbol, int numberOfOperands,
boolean leftAssociative, int precedence) {
		super(symbol, numberOfOperands, leftAssociative, precedence);
	}

	@Override
	public double apply(double... args) {
		return Math.pow(args[0], args[1]);
	}

}
```

```java
	CustomOperator op1 = new CustomOperator("**", 2,
false, Operator.PRECEDENCE_POWER) ;
	String s = "x**5" ;
	ExpressionBuilder eb = new ExpressionBuilder(s) ;
	eb.variables("x") ;
	eb.operator(op1) ;
	Expression ex = eb.build() ;
	ex.setVariable("x", 2.0) ;
	double result = ex.evaluate() ;
	System.out.println(result);
```

Expression Evaluation

- **Evaluation time benchmarking**
  - Run the evaluation of **"sin(x)"** for 1000000 doubles in **[0, π]** using exp4j
  - Run the same evaluation using **Math.sin(x)** function (calls native method)
  - Timing with static method from System class (Long value): ***System.currentTimeMillis()***
  - Timer class in **mathLib.util** package

Simple timing:

```
long start = System.currentTimeMillis()

// do stuff

long end = System.currentTimeMillis()
long duration = end – start
System.out.println(duration) // msec
```

Timer class:

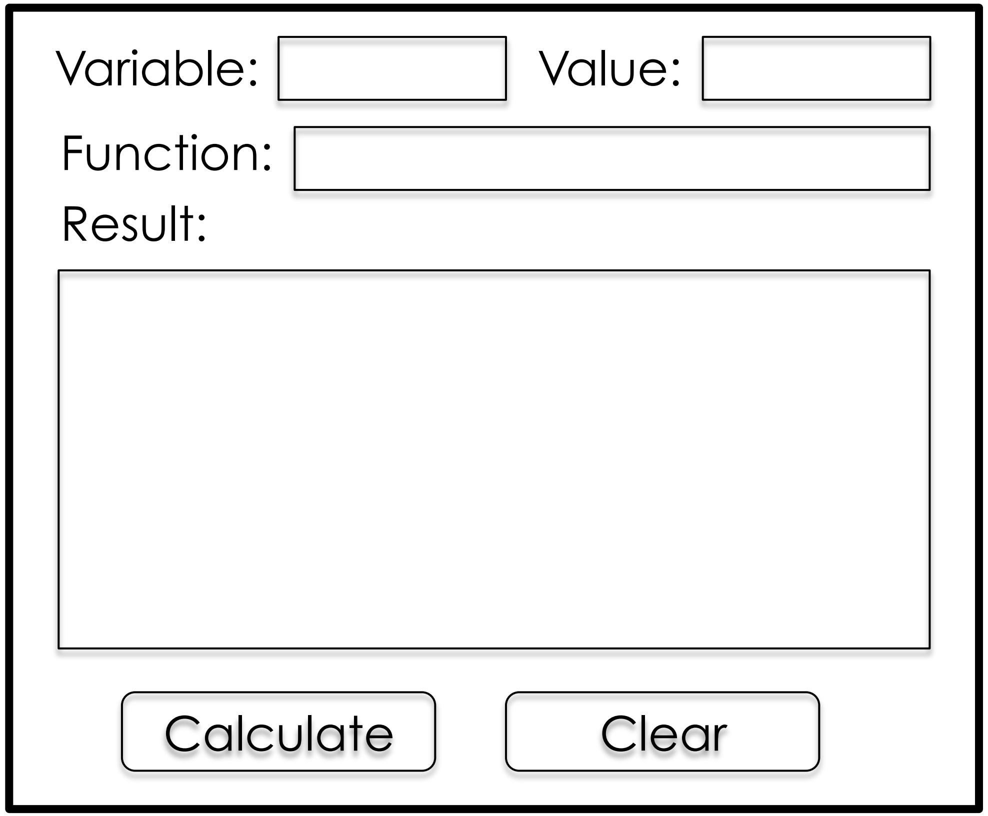
```
Timer timer = new Timer()
timer.start()

// do stuff

timer.stop()
System.out.println(timer)
```

Calling "toString()" method in Timer class

Expression Evaluation

- **Exercise 1: Simple Swing app for expression evaluation**

Single variable function evaluator



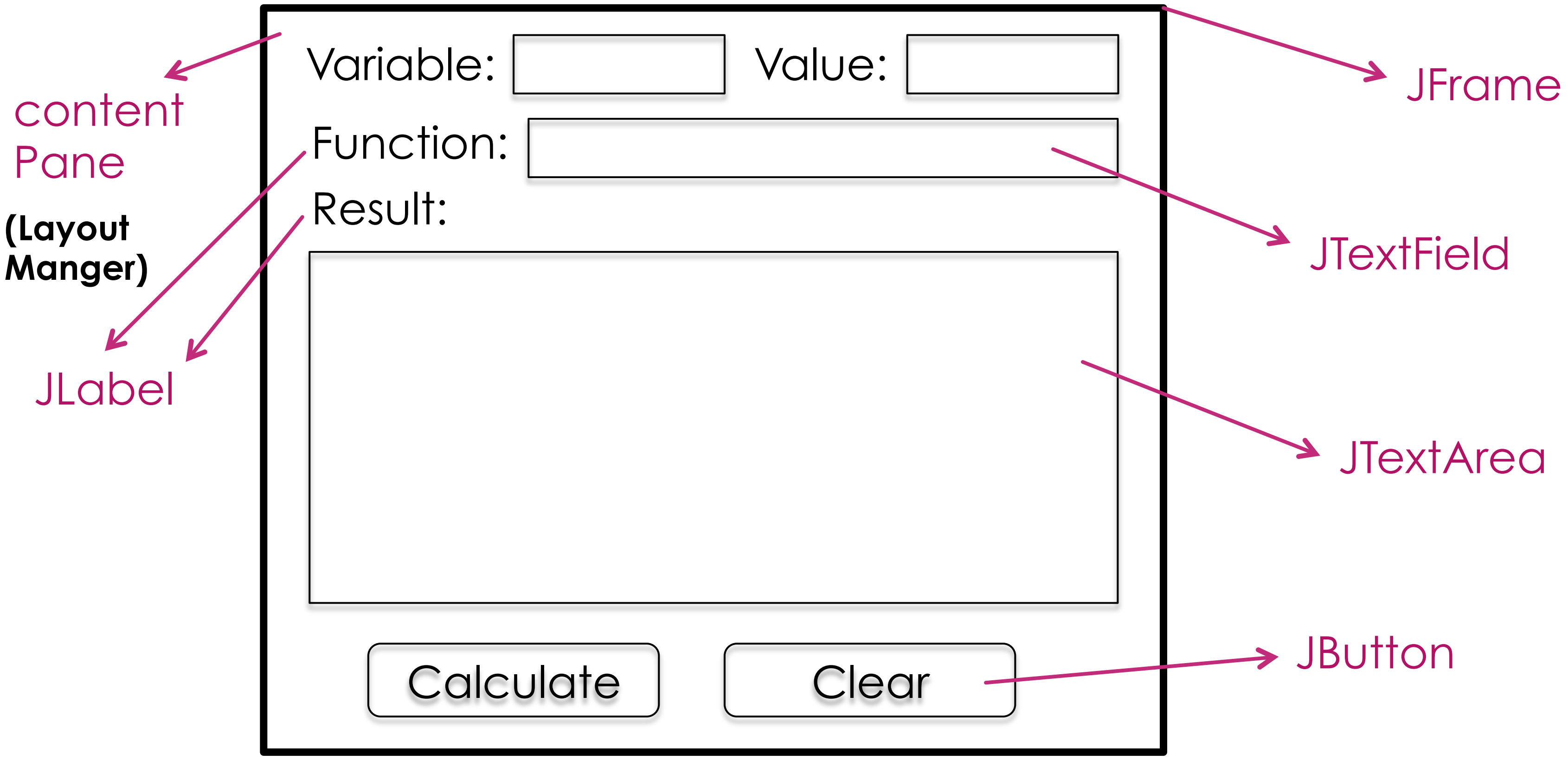
- Option 1: Java Swing (javax, awt, SWT)
  - ✓ Use **Window Builder** in Eclipse (plugin)
- Option 2: JavaFX
  - ✓ Use **Scene Builder**
  - ✓ Allows CSS styling

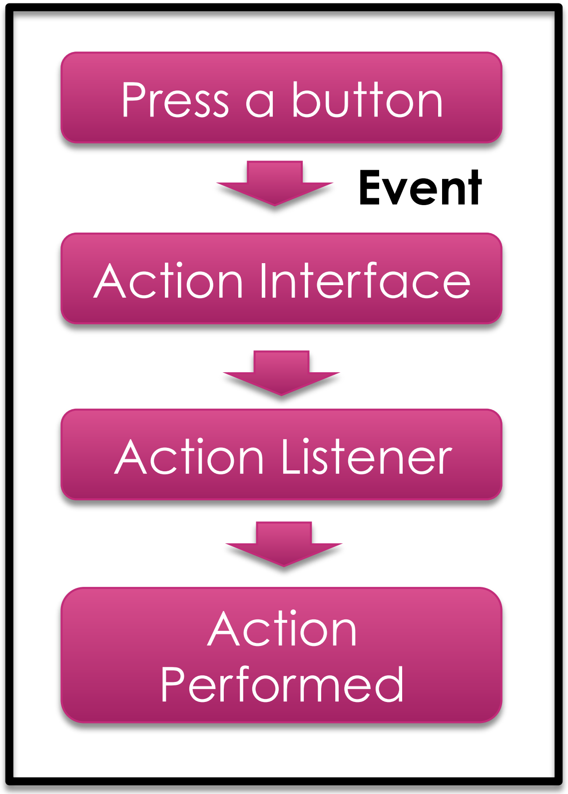
CSS: cascading style sheet

Expression Evaluation

- **Exercise 1: Simple Swing app for expression evaluation**
- Option 1: Java Swing (javax)
  - ✓ Use **Window Builder** in Eclipse (plugin)

Single variable function evaluator



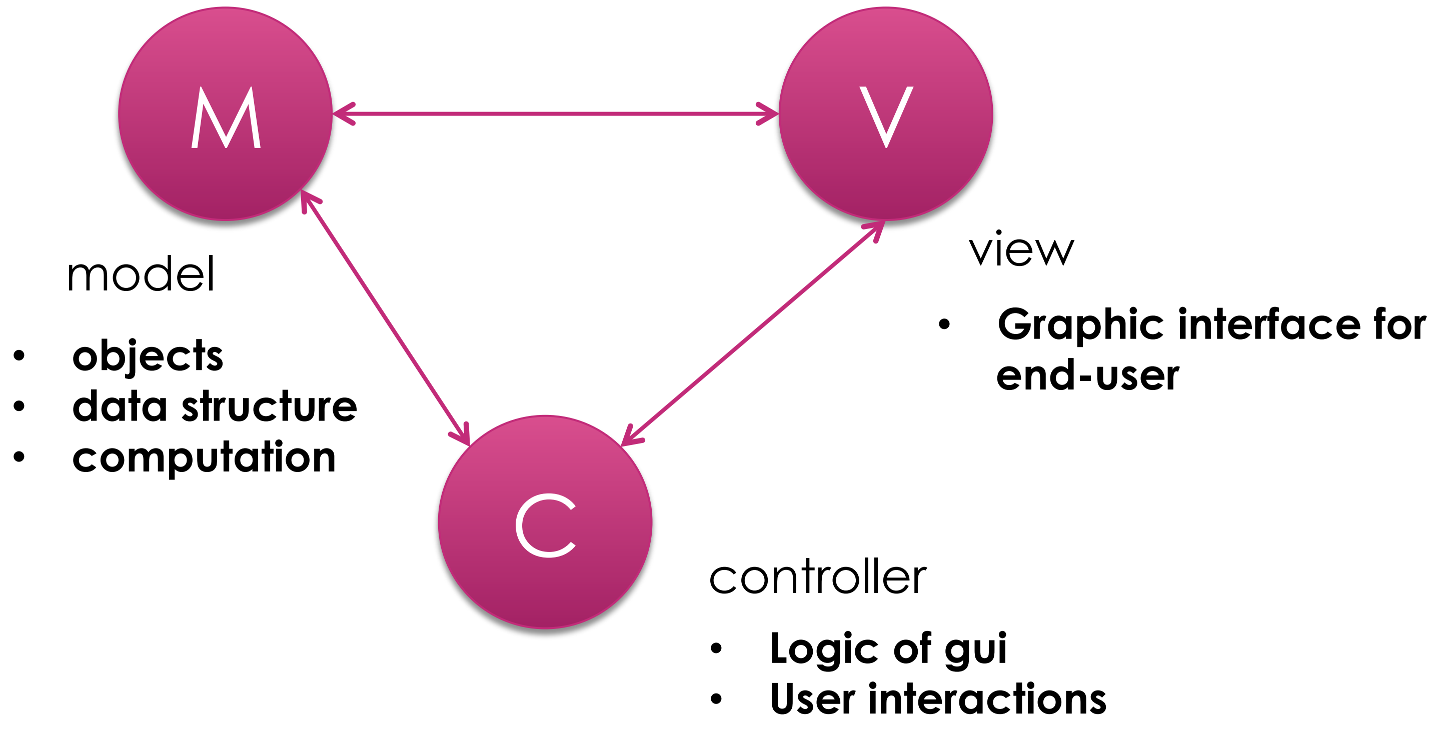


Handling user events

Tip: JFrame is serializable and can be saved.

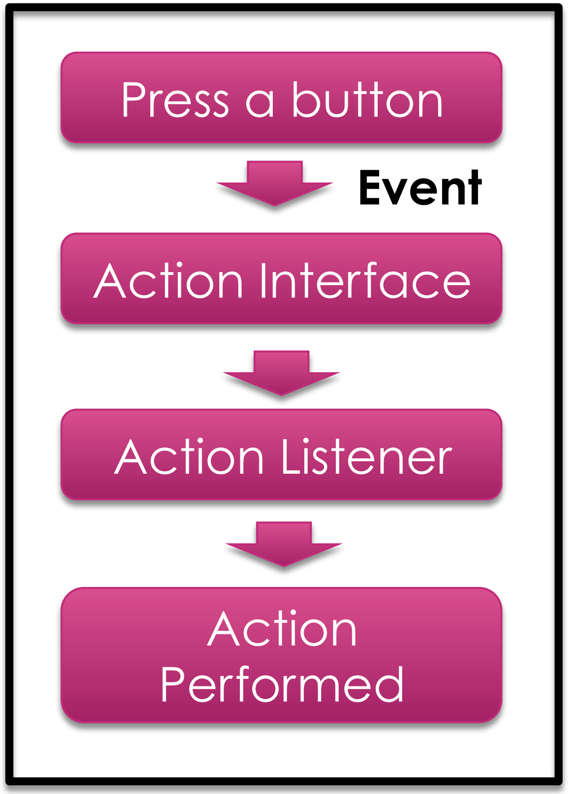
Expression Evaluation

- **Model-View-Controller (MVC) design pattern**
  - **Popular** paradigm for designing GUI
  - Controller glues model and view together



Tip: You only need to save model.

- Option 1: Java Swing (javax)
  - ✓ Use **Window Builder** in Eclipse (plugin)



Handling user events