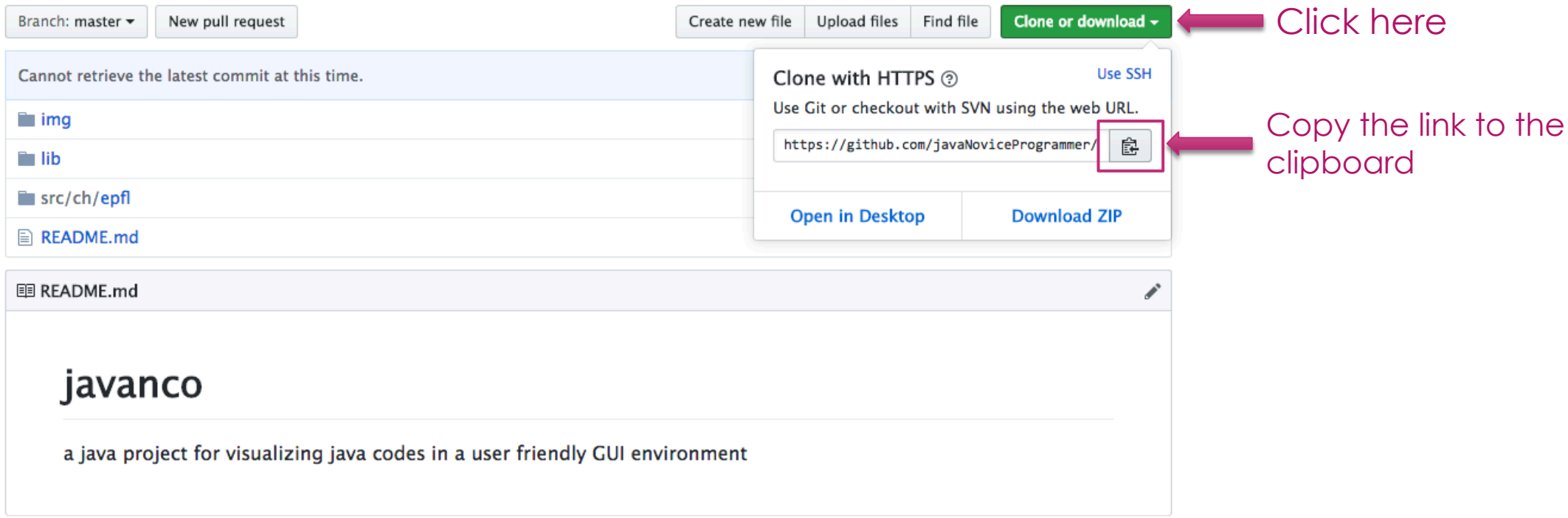
GUI Generation of Java Code

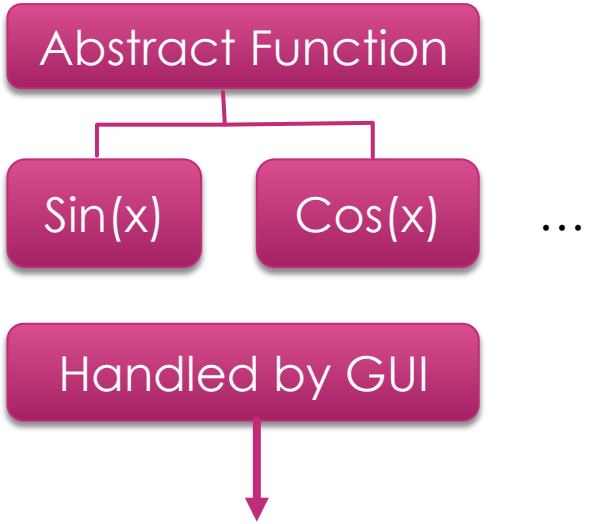
GUI Generation

- **Goal: given a java class, create a GUI →** represents the constructor of java class
  - To iterate over its parameters
  - To perform some calculations at each iteration
  - To present all the results using a result manager with plotting capability (visualization)
- Project **javanco** on github



GUI Generation

- Project **javanco** on github
  - Import into eclipse
  - Works based on reflections framework in java
  - Creates a **tree view** of the java code (swing GUI)
- How to use this?
- Let's look at an example:
- We have several functions: sin(x), cos(x), tan(x), …
- We want the user to select some of the them and plot them
  - Declare "Function" as an abstract class
  - Add concrete implementation for each function
- The test class should implement an interface: **Experiment** → "run" method
- Result manager uses **DataPoint**
- Add parameters and values to a DataPoint using "add property" or "add **result** property"

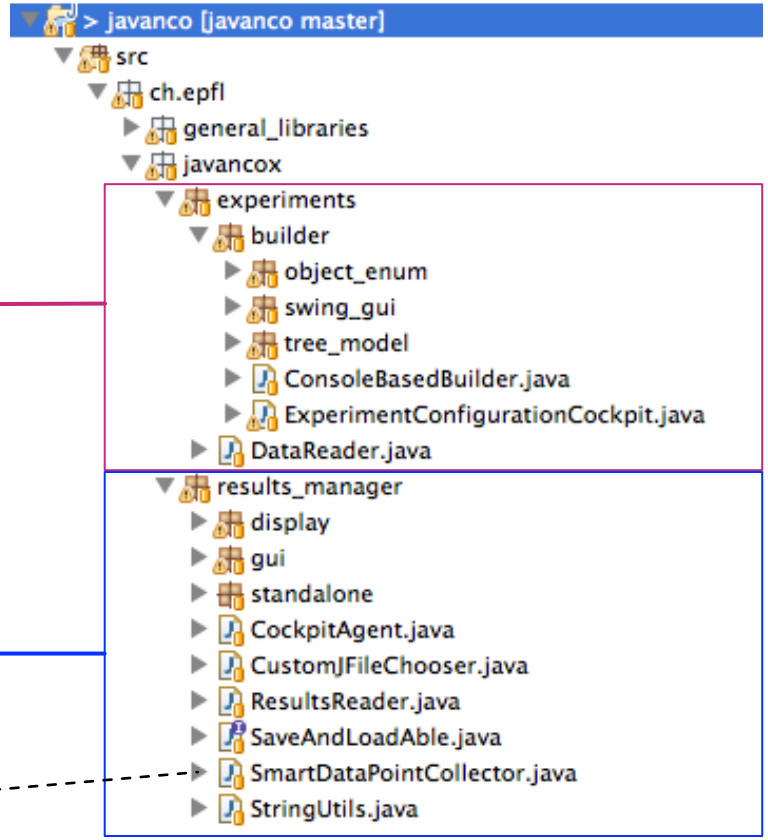


- User interaction
  - **Data collector**
    - Result Manager

GUI Generation

- Project **javanco** on github
  - Two top packages
    - general libraries
    - javacox (javanco execution)
      - ✓ Experiment builder
        - ➢ Takes care of class loading
        - ➢ Takes care of object enumeration
        - ➢ **Multi-thread** capability
      - ✓ Result manager
        - ➢ Takes care of collecting data
          - ✧ Smart datapoint collector
        - ➢ Takes care of displaying and plotting
        - ➢ Takes care of filtering data

Project javanco



GUI Generation

- Example: using our symbolic math project

```
public class TestJavanco implements Experiment {

      Function func ;
      double value ;

      public TestJavanco(
                  @ParamName(name="Choose Function") Function func,
                  @ParamName(name="Value") double value
                  ) {
            this.func = func ;
            this.value = value ;
      }

      @Override
      public void run(AbstractResultsManager man, AbstractResultsDisplayer dis) throws WrongExperimentException {
            DataPoint dp = new DataPoint() ;
            // add variables and values to dp
            dp.addProperty("X", value);
            dp.addResultProperty("Function: " + func.toString(), func.getValue(value));
            // add dp to manager
            man.addDataPoint(dp);
      }

      public static void main(String[] args) {
            String className = TestJavanco.class.getName() ;
            String packageName = "sym_math" ;
            String[] arguments = {"-c", className, "-p", packageName} ;
            ExperimentConfigurationCockpit.execute(arguments, true);
      }
}
```

Must implement "**Experiment**" interface to use the "run" method

Constructor for testing our symbolic Function class

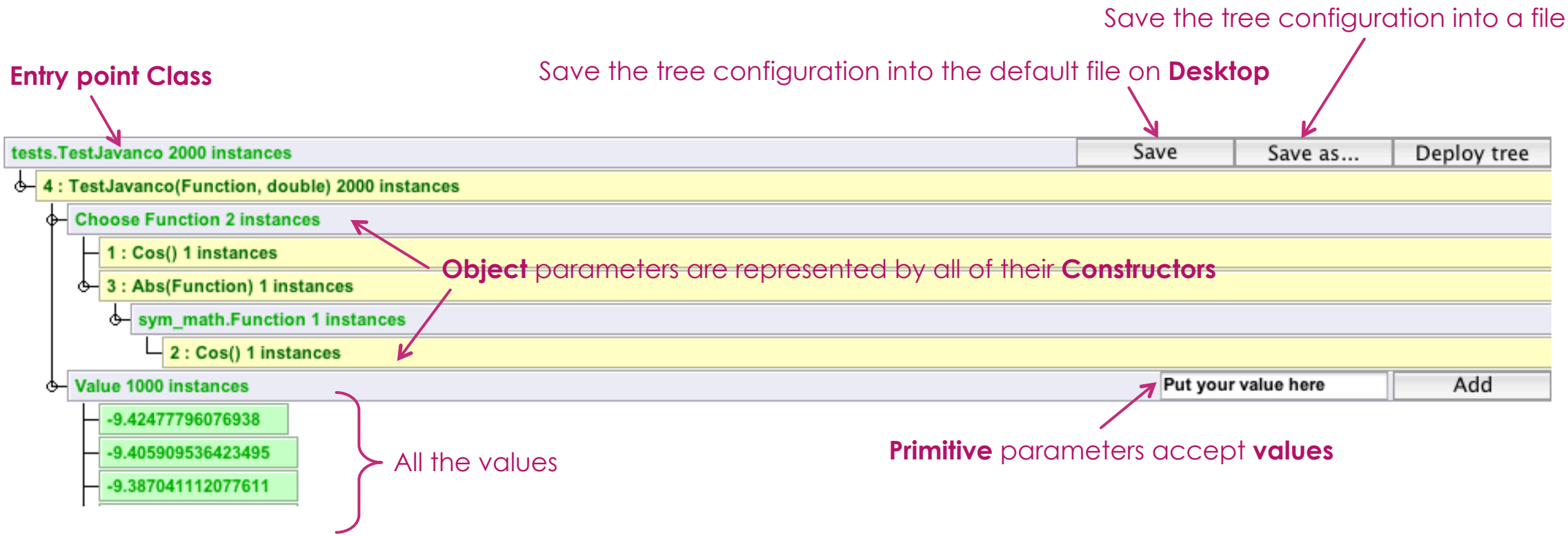
Annotation for GUI purposes

"**run**" method for Experiment. Must create datapoint and add the parameters as "property" or "result property"

"main" method to run "**Experiment Configuration**" class

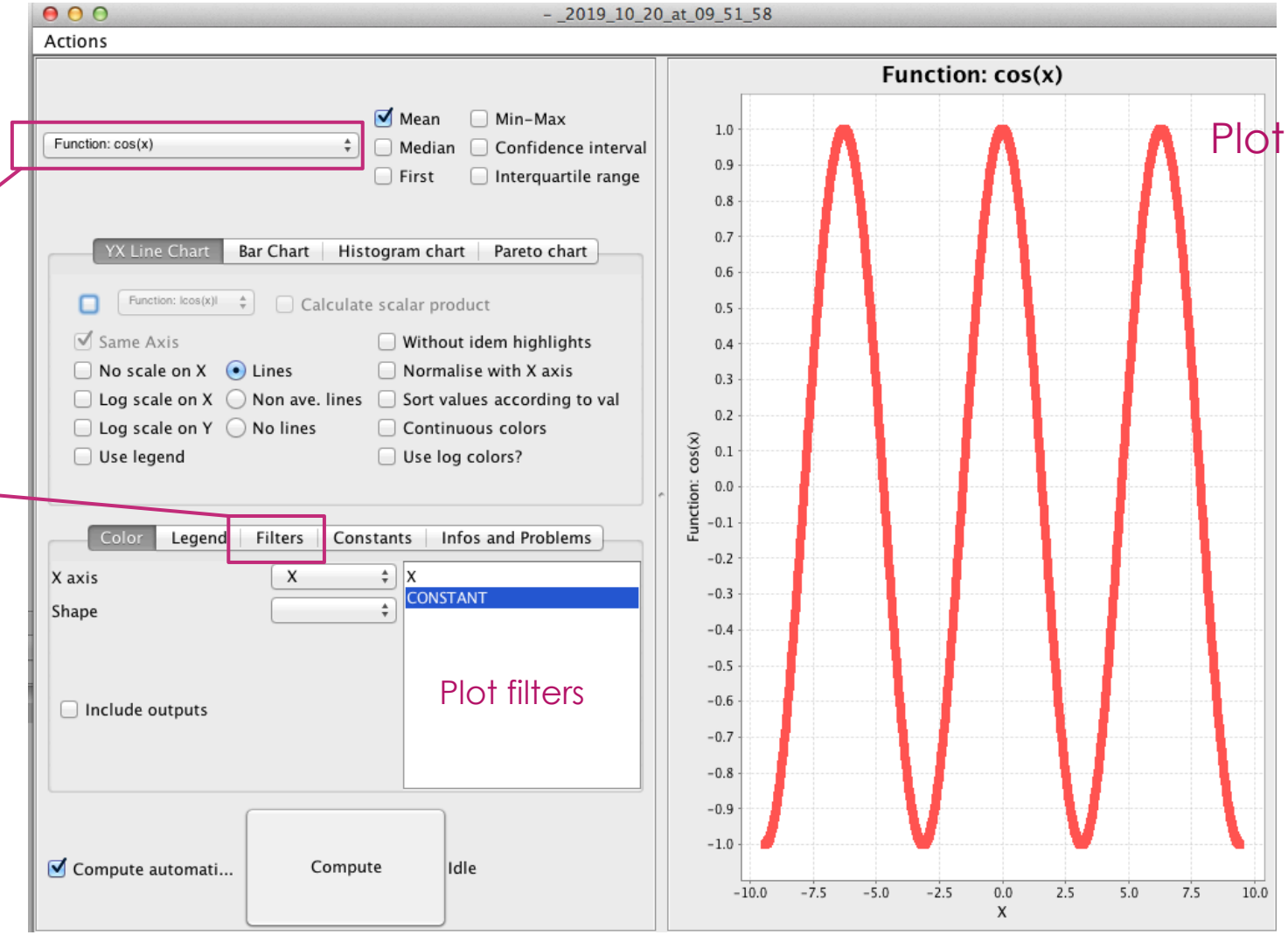
GUI Generation

- Example: using our symbolic math project
- Generated GUI (tree)
- Enumeration of objects is automatically taken care of



GUI Generation

- Example: using our symbolic math project
- Result manager
- Visualization
- Data Filtering



results

data filters

Plot filters

Plot (visualization)