Pods: created by: Darshan Karia & Anchit Sharma for CS251B

Running Instructions:

The project can be opened with eclipse IDE, built and run to execute. In case the project gives troubles executing, it might be because of usage of external jar file "jgraph.jar", which I used by our component diagram panel. This can be fixed by importing jgraph.jar located under lib folder of the project.

Component creating instructions:

Constraints:

- The component should be in the package "components". E.g. **package** components;
- "Component" and "Interface" classes are in package "model". E.g. **import** model.*;
- Our Component Super Class has **protected** ArrayList<Interface> requiredList; variable for purpose of ease.
- While saving Pods container file, the user has to specify ".pod" at the end of the file name. E.g. "sample.pod" is valid file name.

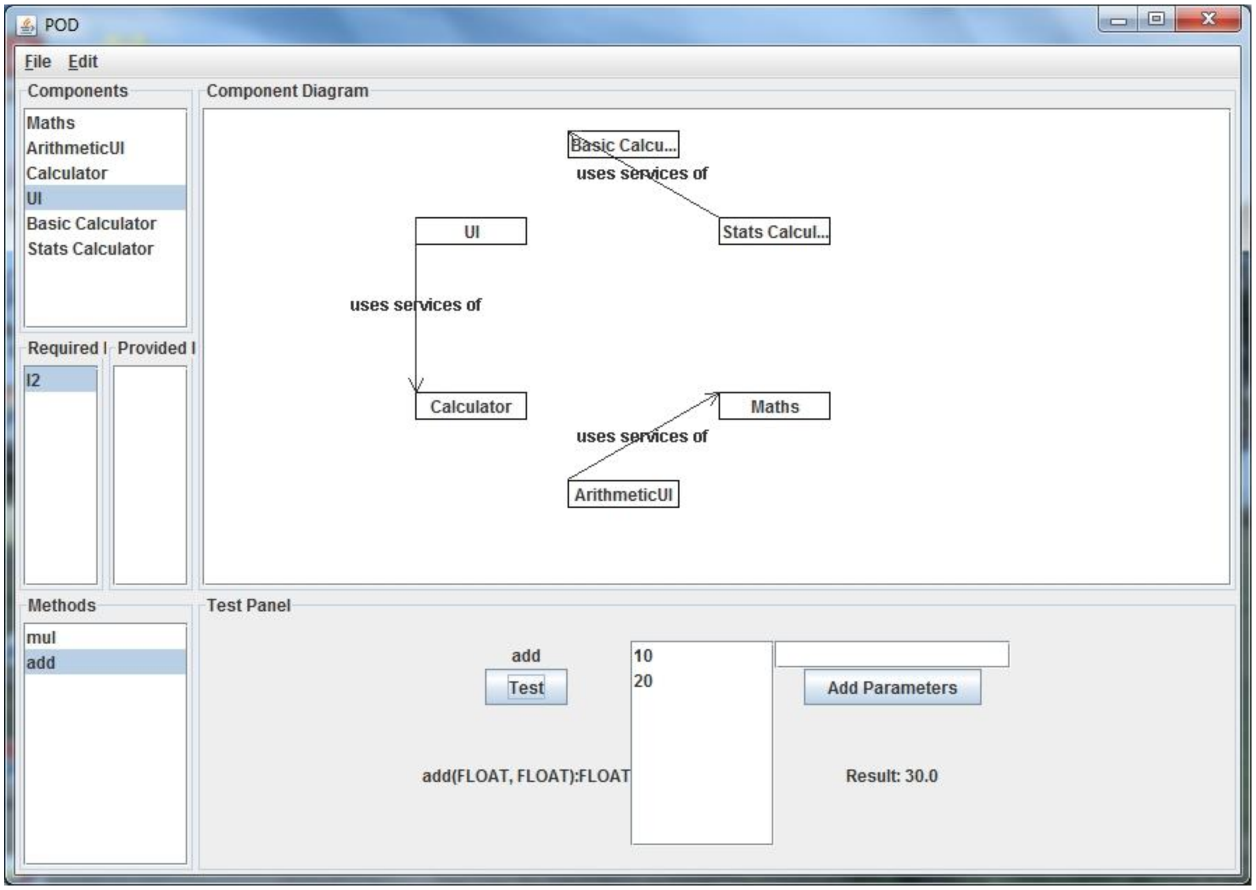
Additional Features:

- Extended Component Class should call super constructor with name of component as parameter. E.g. **super**("Basic Calculator");
- Although we have a default constructor for Interface class which takes no parameters, we also have a constructor with one parameter, i.e. name of the Interface, which makes it easier to identify the interface in JList of GUI. E.g. iArith = **new** Interface("intfArithmetic");
- While adding operations to an Interface, we have provided both the functionality of adding only function name or adding function name with its complete MapType.
    - E.g.

```
public void put(String funcName, MapType mapType){
        operations.put(funcName, mapType);
}
public void addOperation(String funcName){
        operations.put(funcName, null);
}
```

Project Features:

- The component diagram creates a graphical view of the current component objects and their connections with the components providing them service.
- There are JLists for viewing Components, Interfaces and Methods.
- On the bottom right corner there is a Test Panel where parameters can be added in the JList by clicking on the add parameters button and testing the function selected by clicking on the test button.
- Save/Save As/Open of container files works successfully.
- Components can be added from Edit Menu and to remove a component completely from the project (including JList, Component Diagram and YellowPages), select the desired component from Components JList and click on Remove Component from Edit Menu.
- Screenshot of a working container is shown below:



Components: Maths, ArithmeticUI, Calculator, UI, Basic Calculator, Stats Calculator

Component Diagram: Basic Calcu... uses services of Stats Calcul...; UI uses services of Calculator; ArithmeticUI uses services of Maths

Required I: I2

Methods: mul, add

add

10
20

add(FLOAT, FLOAT):FLOAT

Result: 30.0

Sample Component File:

```
package components;

import model.*;
/*import container.Component;
import container.Interface;*/

public class StatsCalculator extends Component {

	private Interface iAddAndDiv;
	private Interface iStats;

	private float sum = 0;
	private float count = 0;

	public StatsCalculator() {
		super("Stats Calculator");
		iAddAndDiv = new Interface("iAddDiv");
		iAddAndDiv.add("add");
		iAddAndDiv.add("div");
		put(iAddAndDiv, null);
		iStats = new Interface("iStats");
		iStats.add("add");
		iStats.add("getAvg");
		iStats.add("getCount");
		add(iStats);

	}

	public Object execute(String operation, Interface service, Object[] args)
			throws Exception {
		Object result = null;
		if (operation.equals("add")) {
			Object[] inputs = new Object[]{args[0], new Float(sum)};
			sum = (Float)invoke("add", iAddAndDiv, inputs);
			inputs = new Object[]{new Float(count), new Float(1)};
			count = (Float)invoke("add", iAddAndDiv, inputs);
		} else if (operation.equals("getCount")) {
			return (Float)count;
		} else if (operation.equals("getAvg")) {
			Object[] inputs = new Object[]{new Float(sum), new Float(count)};
			result = (Float)invoke("div", iAddAndDiv, inputs);
		} else {
			throw new Exception("unrecognized operation: " + operation);
		}
		return result;
	}

}
```

Component.java used in our project:

```
package model;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

@SuppressWarnings("serial")
public class Component implements Serializable{

	protected String name;
	protected ArrayList<Interface> provided;	//not present in new structure
	public Map<Interface,Component> required;
	protected ArrayList<Interface> requiredList;
	protected Container container;

	public Component(String name){
		this.name = name;
		provided = new ArrayList<Interface>();
		required = new HashMap<Interface,Component>();
		requiredList = new ArrayList<Interface>();
	}
	public Component(){
		this("c");
	}

	public String getName(){
		return name;
	}

	public void addConnectionToRequired(Interface intf, Component cmp){
		required.put(intf, cmp);
	}

	public ArrayList<Interface> getProvided(){
		return provided;
	}

	public ArrayList<Interface> getRequiredList(){
		return requiredList;
	}

	public Object execute(String opName, Interface service, Object[] args) throws Exception{
		return new Object();
	}

	public Object invoke(String opName, Interface service, Object[] args) throws Exception{
	    Component provider = required.get(service);
	    if (provider == null) {
	      provider = view.ContainerView.container.findComponent(service);
	      required.put(service, provider);
```

```
          }
          return provider.execute(opName, service, args);
        }

        public void setContainer(Container container) {
                this.container = container;
        }

        public void add(Interface intf){
                provided.add(intf);
        }

        public boolean validate(String operation, Interface service){
                return true;
        }

        public void put(Interface intf, Component cmp){
                required.put(intf, cmp);
                requiredList.add(intf);
        }

        public String toString(){
                return this.getName();
        }
}
```