Ramaiah Institute of Technology, Bangalore
Department of Information Science and Engineering

Name & USN:	Dhruthick Gowda M, 1MS17IS040
Design Pattern:	Command Pattern
Course Code and Name:	IS62B Object Oriented Analysis and Design Patterns

1. Case Study - Physical Units (of length) Converter

The command pattern is a behavioral design pattern that converts requests from the user into objects. I demonstrate the usage of this pattern in a simple desktop application built using Java's Swing library. The application's function is to convert the values of physical quantities (here, length) to one of its other various units of measurement. The physical value of length can vary from angstroms in the atomic world to light-years in space, while we generally use meters, kilometers, feet, inches, etc in daily life. The application is a simple calculator that returns the value of a measurement in a different unit. For length, three groups are considered as follows:

- The general units of length
    - Meters
    - Kilometers
    - Feet
    - Yards
    - Inches
    - Furlongs
    - Miles
    - Nautical miles
- The astronomical units of length
    - Light years
    - Gigameters
    - Parsecs
    - Astronomical units
    - Lunar length
    - Light hours
- The atomic units of length
    - Micrometers
    - Picometers
    - Angstroms
    - Atomic units
    - Electron radius
    - Microinches

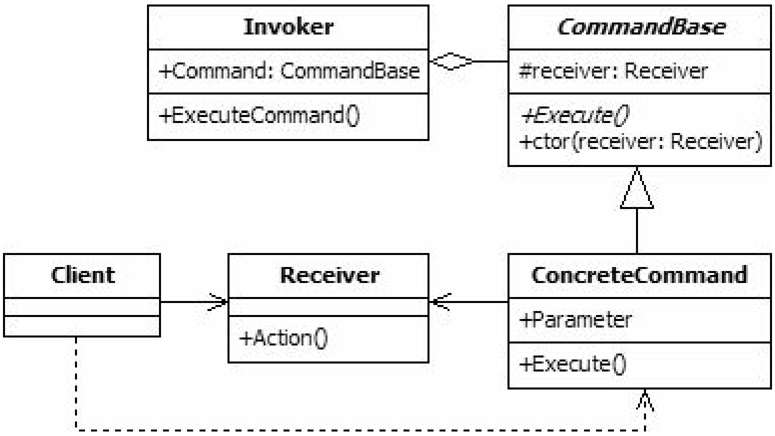
In each group, the user can enter the value in one unit, and then, choose to convert the value to another unit of his choice within the group by clicking on the "Convert" button.

The functionalities of the application can be further extended to that of a miscellaneous calculator that also converts units of other types of physical measurements like mass, time, current, temperature, amount of substance, and even the various currencies in the world.

2. The Command Pattern

The command pattern is a design pattern that encapsulates all of the information required for a request to be executed within a single object. As the pattern defines a manner for controlling communication between classes or entities, it is classified as a behavioral pattern.

The general structure of this pattern is described in the figure below:



- **Client**: This class consumes the classes of the command design pattern. It generates the command objects and associates them to the respective receivers. *Example: The Demo class in the case study.*

- **Receiver**: Receiver objects include the methods that are executed when one or more commands are invoked. *Example: The Calculator (JTextArea) class in the case study.*

- **CommandBase**: This abstract class or interface is the base for all the command objects. It includes an abstract method "execute" that has to be defined by its subclasses or implementations. *Example: The Command interface in the case study.*

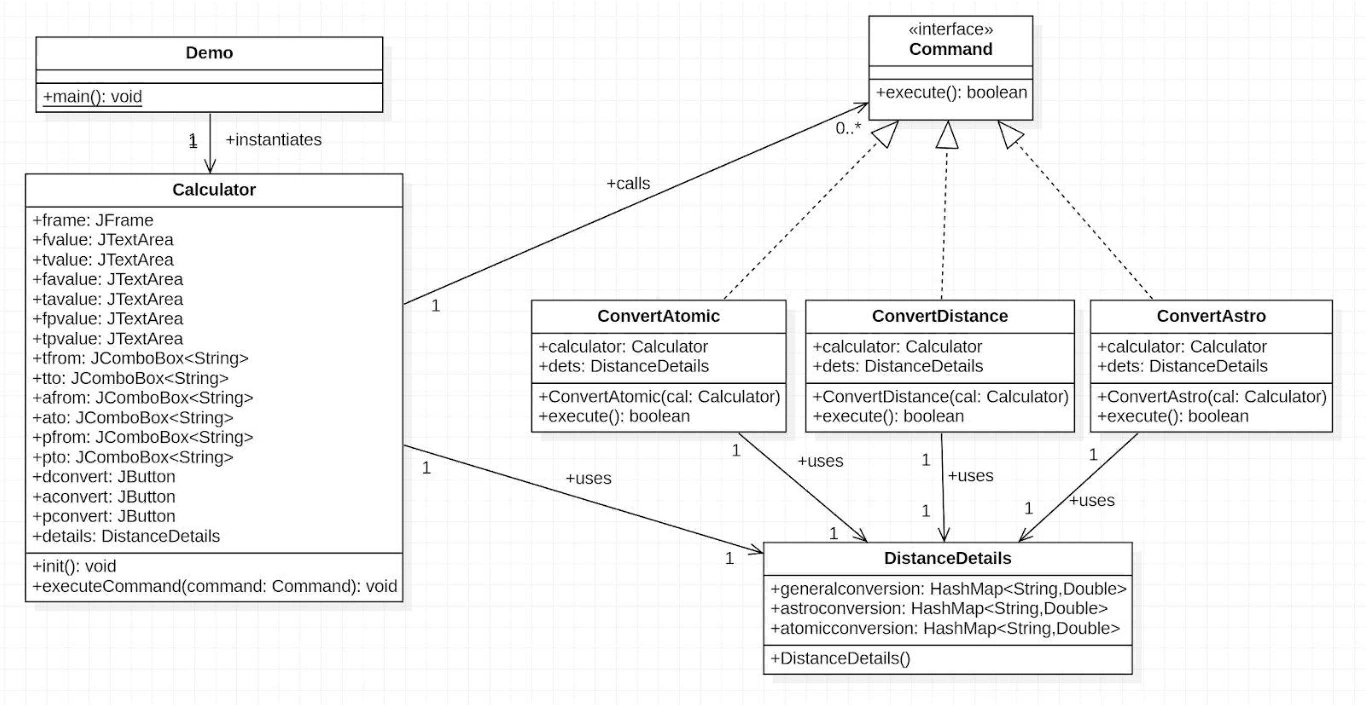
- **ConcreteCommand**: Concrete command classes are subclasses or implementations of the CommandBase abstract class or interface. In addition to implementing the execute method, they contain all of the information that is required to properly perform the action. *Example: The ConvertAstro, ConvertAtomic, and ConvertDistance classes in the case study.*

- **Invoker**: The Invoker object launches the execution of commands and could be controlled by the Client object, as in the case study. However, the invoker may be detached from the client. For example, the client could produce a queue of commands that are executed regularly by a timed event. *Example: The Calculator (JButton) class in the case study.*

The key features of the command pattern include:

- It facilitates code scalability, allowing the programmer to incorporate new commands without having to change the existing code.
- It enhances loose-coupling between the invoker and the receiver.
- It is essentially an object-oriented callback.

3. Class Diagram



4. Implementation

//DistanceDetails.java

```
import java.util.*;
public class DistanceDetails {
        String general[]= {"meters","miles","inches","kilometers"
```

```
            ,"feet","yards","nautical miles","furlongs"},astronomical[]= {
            "gigameters","light years","astronomical units","parsecs","lunar
distance","light hours"}, atomic[]= {"micrometers","picometers","microinches",
                    "angstrom","atomic units","electron radius"};
    double generalval[]= {1,1609.344,0.0254,1000,0.3048,0.9144,1852,201.168},
            astronomicalval[]= {1,9460730.47258,149.597870691,
            30856775.8128,0.384000098304,1079.2528488},
            atomicval[]= {354869043.8833,354.8690438833,9013673.714636,
                    35486.90438833,18778.86243974,1};
    HashMap<String,Double> generalconversion,astroconversion,atomicconversion;
    DistanceDetails(){
        generalconversion=new HashMap<String,Double>();
        astroconversion=new HashMap<String,Double>();
        atomicconversion=new HashMap<String,Double>();
        for(int i=0;i<8;i++)
            generalconversion.put(general[i], generalval[i]);
        for(int i=0;i<6;i++) {
            astroconversion.put(astronomical[i], astronomicalval[i]);
            atomicconversion.put(atomic[i], atomicval[i]);
        }
    }
}
```

//Command.java (CommandBase)

```
public interface Command {
    public boolean execute();
}
```

//ConvertDistance.java (ConcreteCommand)

```
public class ConvertDistance implements Command {
    Calculator calculator;
    DistanceDetails dets;
    ConvertDistance(Calculator cal){
        calculator=cal;
        dets=new DistanceDetails();
    }
    @Override
    public boolean execute() {
        // TODO Auto-generated method stub
        String from, to;
        double fromv,tov;
        from=(String)calculator.tfrom.getSelectedItem();
        to=(String)calculator.tto.getSelectedItem();
        fromv=Double.parseDouble(calculator.fvalue.getText());
        tov=fromv*dets.generalconversion.get(from)/dets.generalconversion.get(to);
        calculator.tvalue.setText(tov+"");
        return true;
    }
}
```

//ConvertAstro.java (ConcreteCommand)

```
public class ConvertAstro implements Command {
    Calculator calculator;
    DistanceDetails dets;
    ConvertAstro(Calculator cal){
        calculator=cal;
        dets=new DistanceDetails();
    }
    @Override
    public boolean execute() {
        // TODO Auto-generated method stub
        String from, to;
        double fromv,tov;
        from=(String)calculator.afrom.getSelectedItem();
        to=(String)calculator.ato.getSelectedItem();
        fromv=Double.parseDouble(calculator.favalue.getText());
        tov=fromv*dets.astroconversion.get(from)/dets.astroconversion.get(to);
        calculator.tavalue.setText(tov+"");
        return true;
    }
}
```

//ConvertAtomic.java (ConcreteCommand)

```
public class ConvertAtomic implements Command {
    Calculator calculator;
    DistanceDetails dets;
    ConvertAtomic(Calculator cal){
        calculator=cal;
        dets=new DistanceDetails();
    }
    @Override
    public boolean execute() {
        // TODO Auto-generated method stub
        String from, to;
        double fromv,tov;
        from=(String)calculator.pfrom.getSelectedItem();
        to=(String)calculator.pto.getSelectedItem();
        fromv=Double.parseDouble(calculator.fpvalue.getText());
        tov=fromv*dets.atomicconversion.get(from)/dets.atomicconversion.get(to);
        calculator.tpvalue.setText(tov+"");
        return true;
    }
}
```

//Calculator.java (Invoker & Receiver)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Calculator {
        DistanceDetails details=new DistanceDetails();
        JFrame frame;
        JPanel distancepanel1,distancepanel2,distancepanel3,
        astropanel1,astropanel2,astropanel3,
        atomicpanel1,atomicpanel2,atomicpanel3;
        JComboBox<String> tfrom,tto,afrom,ato,pfrom,pto;
        JTextArea fvalue,tvalue,favalue,tavalue,fpvalue,tpvalue;
        JButton dconvert,aconvert,pconvert;
        public void init() {
                frame=new JFrame("Calculator");
                frame.setVisible(true);
                frame.setLayout(new GridLayout(0,1));
                Calculator calculator=this;
                frame.setSize(800, 200);
                distancepanel1=new JPanel();
                distancepanel1.add(new JLabel("Distance",JLabel.CENTER));
                distancepanel1.setLayout(new FlowLayout());
                frame.add(distancepanel1);
                distancepanel2=new JPanel();
                distancepanel2.setLayout(new FlowLayout());
                tfrom=new JComboBox<String>(details.general);
                tto=new JComboBox<String>(details.general);
                distancepanel2.add(new JLabel("From",JLabel.CENTER));
                distancepanel2.add(tfrom);
                distancepanel2.add(new JLabel("To",JLabel.CENTER));
                distancepanel2.add(tto);
                frame.add(distancepanel2);
                distancepanel3=new JPanel();
                distancepanel3.setLayout(new FlowLayout());
                fvalue=new JTextArea(3,20);
                distancepanel3.add(fvalue);
                tvalue=new JTextArea(3,20);
                distancepanel3.add(tvalue);
                dconvert=new JButton("Convert");
                distancepanel3.add(dconvert);
                frame.add(distancepanel3);
                dconvert.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
```

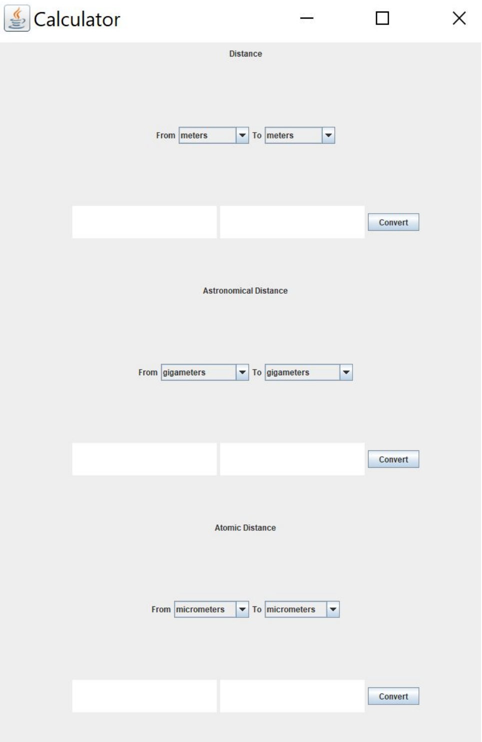
```
        executeCommand(new ConvertDistance(calculator));
    }
});
            astropanel1=new JPanel();
            astropanel1.add(new JLabel("Astronomical Distance",JLabel.CENTER));
            astropanel1.setLayout(new FlowLayout());
            frame.add(astropanel1);
            astropanel2=new JPanel();
            astropanel2.setLayout(new FlowLayout());
            afrom=new JComboBox<String>(details.astronomical);
            ato=new JComboBox<String>(details.astronomical);
            astropanel2.add(new JLabel("From",JLabel.CENTER));
            astropanel2.add(afrom);
            astropanel2.add(new JLabel("To",JLabel.CENTER));
            astropanel2.add(ato);
            frame.add(astropanel2);
            astropanel3=new JPanel();
            astropanel3.setLayout(new FlowLayout());
            favalue=new JTextArea(3,20);
            astropanel3.add(favalue);
            tavalue=new JTextArea(3,20);
            astropanel3.add(tavalue);
            aconvert=new JButton("Convert");
            astropanel3.add(aconvert);
            frame.add(astropanel3);
            aconvert.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        executeCommand(new ConvertAstro(calculator));
    }
});
            atomicpanel1=new JPanel();
            atomicpanel1.add(new JLabel("Atomic Distance",JLabel.CENTER));
            atomicpanel1.setLayout(new FlowLayout());
            frame.add(atomicpanel1);
            atomicpanel2=new JPanel();
            atomicpanel2.setLayout(new FlowLayout());
            pfrom=new JComboBox<String>(details.atomic);
            pto=new JComboBox<String>(details.atomic);
            atomicpanel2.add(new JLabel("From",JLabel.CENTER));
            atomicpanel2.add(pfrom);
            atomicpanel2.add(new JLabel("To",JLabel.CENTER));
            atomicpanel2.add(pto);
            frame.add(atomicpanel2);
            atomicpanel3=new JPanel();
            atomicpanel3.setLayout(new FlowLayout());
        fpvalue=new JTextArea(3,20);
        atomicpanel3.add(fpvalue);
        tpvalue=new JTextArea(3,20);
        atomicpanel3.add(tpvalue);
        pconvert=new JButton("Convert");
        atomicpanel3.add(pconvert);
        frame.add(atomicpanel3);
        pconvert.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
      executeCommand(new ConvertAtomic(calculator));
    }
   });
    }
    private void executeCommand(Command command) {
        if(command.execute()) {
            System.out.println("Success");
        }
    }
}
```

//Demo.java (Client)

```
public class Demo {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Calculator calculator=new Calculator();
        calculator.init();
    }
}
```

SNAPSHOTS OF THE RESULT (in the following order)

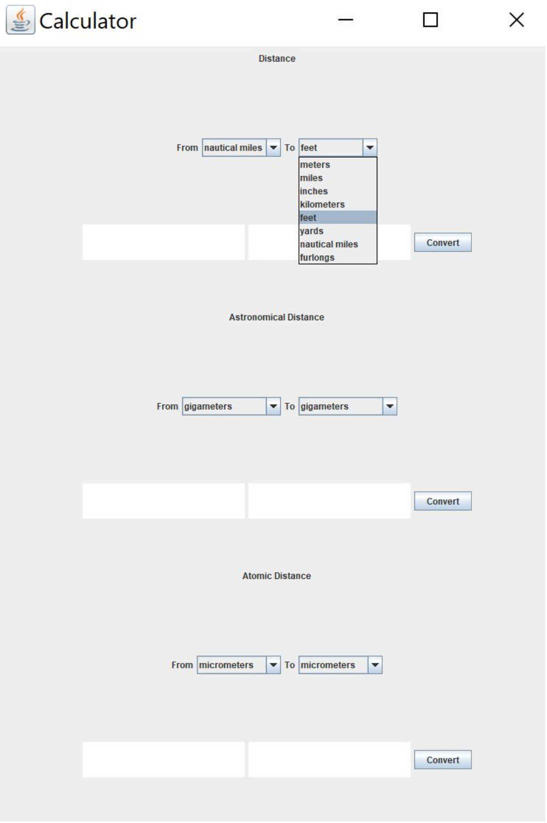
- *At start*
- *General distance - selection of units*
- *General distance - conversion*
- *Astronomical distance - selection of units*
- *Astronomical distance - conversion*
- *Atomic distance - selection of units*
- *Atomic Distance - conversion*
- *Some more conversions*



From meters To meters

From gigameters To gigameters

From micrometers To micrometers


Calculator

Distance

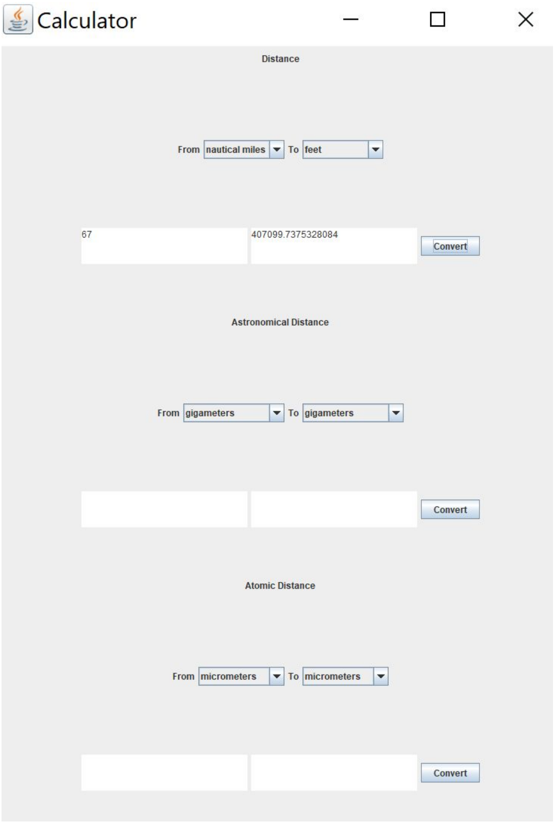
From nautical miles To feet

Astronomical Distance

From gigameters To gigameters

Atomic Distance

From micrometers To micrometers


Calculator

Distance

From nautical miles To feet

67

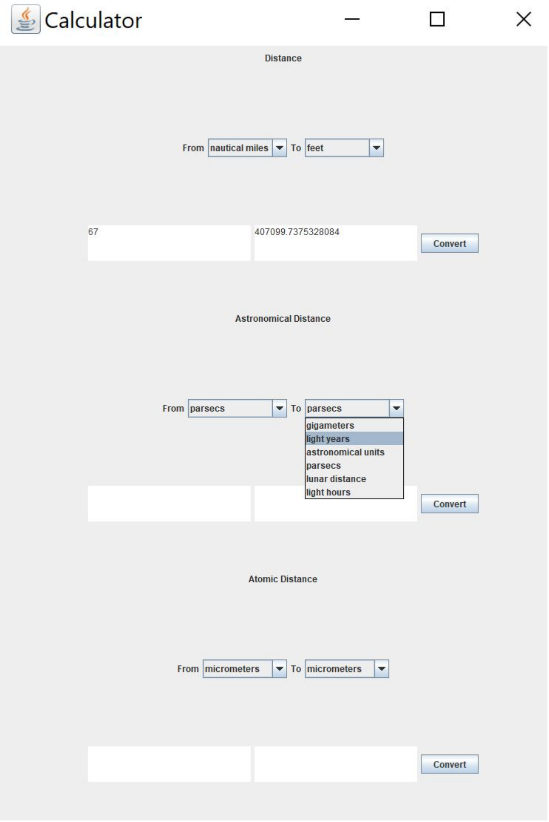
407099.7375328084

Astronomical Distance

From gigameters To gigameters

Atomic Distance

From micrometers To micrometers


Calculator

Distance

From nautical miles To feet

67

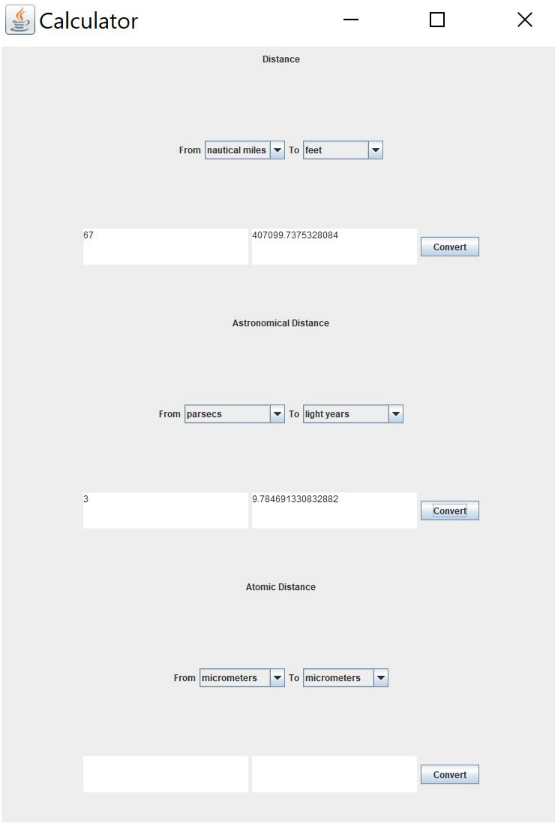
407099.7375328084

Astronomical Distance

From parsecs To parsecs

Atomic Distance

From micrometers To micrometers


Calculator

Distance

From nautical miles To feet

67

407099.7375328084

Astronomical Distance

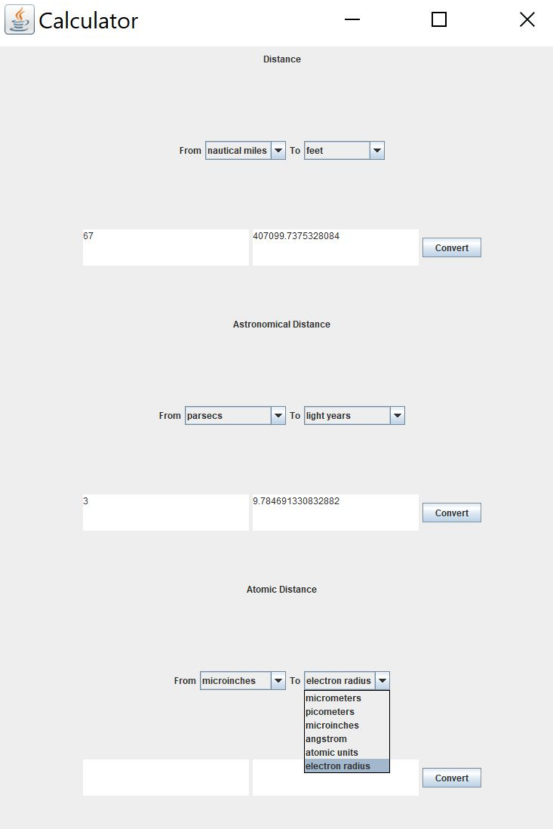
From parsecs To light years

3

9.784691330832882

Atomic Distance

From micrometers To micrometers



Calculator

Distance

From nautical miles To feet

67 | 407099.7375328084

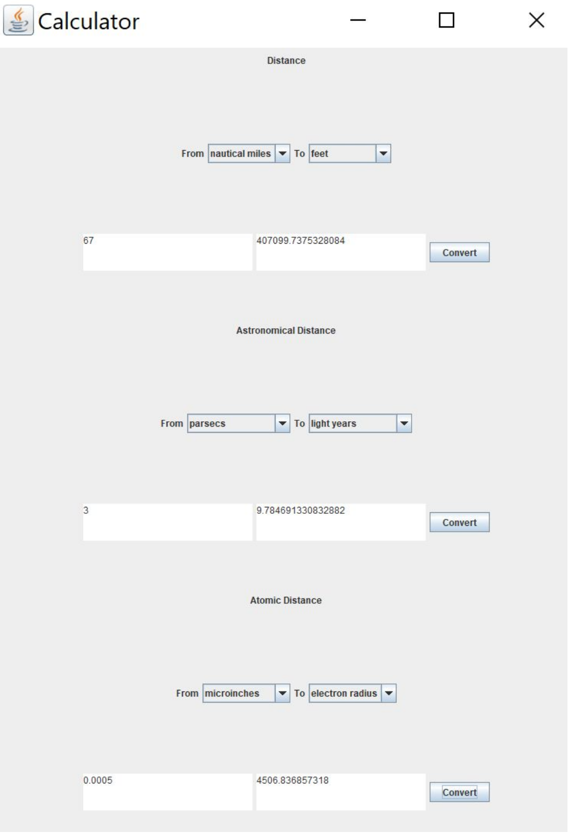
Astronomical Distance

From parsecs To light years

3 | 9.784691330832882

Atomic Distance

From microinches To electron radius



Calculator

Distance

From nautical miles To feet

67 | 407099.7375328084

Astronomical Distance

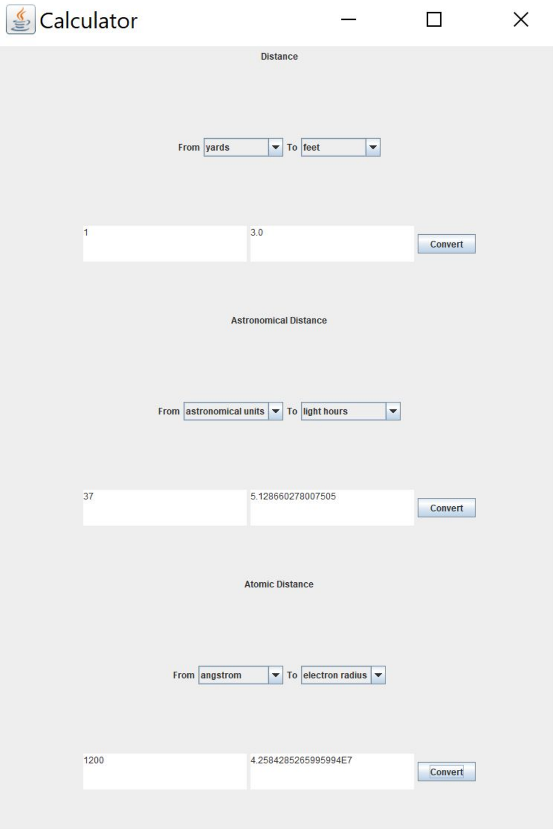
From parsecs To light years

3 | 9.784691330832882

Atomic Distance

From microinches To electron radius

0.0005 | 4506.836857318



Calculator

Distance

From yards To feet

1 | 3.0

Astronomical Distance

From astronomical units To light hours

37 | 5.128660278007505

Atomic Distance

From angstrom To electron radius

1200 | 4.2584285265995994E7