1. [50\%] **Temperature** implements **TemperatureI** and used to convert a given value \( v \) to either Fahrenheit or Centigrade and to compare to freezing in the given temperature scale. Here is **TemperatureI**:

```java
public interface TemperatureI {
    public void setFahrenheit(); // regard \( v \) as \( v \) degrees F
    public void setCentigrade(); // regard \( v \) as \( v \) degrees C
    public float convert(float v); // converted \( v \) to C or F
    public boolean freezing(float v); // true if \( v \) is freezing
}
```

Write the class **Temperature** using the pure **State Pattern** consistent with this driver code:

```java
TemperatureI t = new Temperature();
float v = 20.0f; // \( v \) will be regarded as C then F

   t.setCentigrade(); // now \( v \) treated as 20.0 degrees C.
   t.freezing(v);     ==> false; // 20 deg C. is above freezing
   t.convert(v);      ==> 68.0; // and 20 deg C. = 68 deg F.
   t.setFahrenheit(); // now \( v \) treated as 20.0 degrees F.
   t.freezing(v);     ==> true; // 20 deg F. is below freezing
   t.convert(v);      ==> -6.7; // and 20 deg F. = -6.7 deg C.
```

Do **not** use instanceof, or boolean variables, etc. Use conditionals only to test a given float argument for freezing.

Recall:

- To convert \( v \) deg Centigrade to Fahrenheit: \((9.0f*v/5.0f) + 32.0f\)
- To convert \( v \) deg Fahrenheit to Centigrade: \((v - 32.0f)*5.0f/9.0f\)
- In Centigrade, a float value \( v \) is “freezing” if \((v <= 0.0f)\)
- In Fahrenheit, a float value \( v \) is “freezing” if \((v <= 32.0f)\)

Write **Temperature, State, and its subclasses Centigrade and Fahrenheit**.
2. [50%] A vegetable Garden contains an integer number of vegetables returned by \texttt{int \: getCount()}, and incremented by \texttt{plant(int \: n)} and decremented by \texttt{harvest(int \: n)}. The count is initially zero. Garden implements \texttt{GardenI}. Note that it accepts Visitors.

```java
public interface GardenI {
    public int  getCount(); // return number of vegetables in garden
    public void plant(int n); // add \( n \) vegetables to the garden
    public void harvest(int n); // take from \( 0-n \) vegetables from the garden
    public void acceptVisitor(Visitor v);
}
```

There are two kinds of Visitor to Garden: a Gardener and a Rabbit. A Gardener visits the Garden to either plant or harvest vegetables, and a Rabbit visits to just harvest a few for dinner. The Gardener plans how many vegetables to plant or harvest on a given visit (by methods \texttt{expectToPlant} and \texttt{expectToHarvest}). A Rabbit plans how many vegetables it expects to harvest when visiting. Here is the driver:

```java
GardenI garden = new Garden();
Gardener gloria = new Gardener();
Rabbit roger = new Rabbit();

gloria.expectToPlant(10); // goal is to plant 10 new veggies,
gloria.expectToHarvest(3); // and also harvest 3 veggies,
garden.acceptVisitor(gloria); // next time gloria visits the garden
roger.expectToHarvest(3); // and roger hopes to eat three,
garden.acceptVisitor(roger); // when he visits the garden.
```

\textit{Write Garden, Visitor, Gardener, and Rabbit.}
public class Temperature implements TemperatureI {
    private Fahrenheit fahrenheit = new Fahrenheit();
    private Centigrade centigrade = new Centigrade();
    private State state = fahrenheit;

    public void setCentigrade() { state = centigrade; }
    public void setFahrenheit() { state = fahrenheit; }
    public float convert(float v) { state.convert(v); }
    public boolean freezing(float v) { state.freezing(v); }
}

abstract public class State {
    abstract public float convert(float v);
    abstract public boolean freezing(float v);
}

public class Fahrenheit extends State {
    public float convert(float v) { return (v - 32.0f)*5.0f/9.0f; }
    public boolean freezing(float v) { return v <= 32.0f; }
}

public class Centigrade extends State {
    public float convert(float v) { return (9.0f*v/5.0f) + 32.0f; }
    public boolean freezing(float v) { return v <= 0.0f; }
}
public class Garden {
    private int count;

    public Garden() { count = 0; }

    public int getCount() { return count; }

    public void plant(int n) { count += n; }

    public void harvest(int n) {
        count -= n;
        if (count < 0) count = 0;
    }

    public void acceptVisitor(Visitor v) { v.visit(this); }
}

abstract public class Visitor {
    abstract public void visit(Garden g);
}

public class Gardener extends Visitor {
    private int numberToPlant;
    private int numberToHarvest;

    public Gardener() {
        numberToPlant = 0;
        numberToHarvest = 0;
    }

    public void expectToPlant (int n) { numberToPlant = n; }
    public void expectToHarvest(int n) { numberToHarvest = n; }

    public void visit(Garden g) {
        g.plant(numberToPlant);
        g.harvest(numberToHarvest);
    }
}

public class Rabbit extends Visitor {
    private int numberToHarvest;

    public Rabbit() { numberToHarvest = 0; }

    public void expectToHarvest(int n) { numberToHarvest = n; }

    public void visit(Garden g) { g.harvest(numberToHarvest); }
}