

Overview

In this assignment you will be implementing a weather forecaster. It involves writing 3 different classes plus a driver program. It is recommended that you write one class at a time and then write a driver (tester) program to ensure that it works before putting everything together. Alternately, you can use the Interactions tab of jGRASP to test some of the early classes. This is discussed in the next few pages.

Goals

Write a multi-class program with three classes and a driver.

What to Implement

Based on the following UML diagrams, implement the three required classes, `Temperature`, `Season`, and `Weather`. You will also implement a program that uses all of these classes in some fashion, `Forecaster.java`.

Class and File Naming

Name your classes as follows:

`Temperature` in source file `Temperature.java`

`Season` in source file `Season.java`

`Weather` in source file `Weather.java`

`Forecaster` in source file `Forecaster.java`

How To Not Be Overwhelmed By This Assignment

Work on each class individually, and only in little bits at a time. The suggested order of development for the classes is:

1. `Season`
2. `Temperature`
3. `Weather`

As you write each of these, test each method as you write it. Methods of testing are discussed in the next section, "How to Test Individual Classes".

After these three classes are developed, you can then work on the `Forecaster` program that will be discussed later in this document.

Plan to spend the full time available on this assignment. This is not an assignment that can be written in a night or two. If you parcel out your development over the course of the entire time available, it should be manageable. This is especially important because you can't submit to WebCAT until you have all four classes written and can submit them all at the same time.

Since the *Submission Energy* feature of WebCAT is turned on, you only have 3 submissions per hour once you are at the point of testing all four files working together. This means that a lot of testing needs to be done locally before you're ready to submit to WebCAT.

How to Test Individual Classes

As you write each of the classes below, you should test them as you implement various fields and methods. One way to do this is to write an accompanying driver program. For instance, as you write the `Temperature` class, you might also write `TemperatureDriver.java` which includes a `main()` method and does basic calls on each of the methods. This would give you good practice working with each of the methods in the class to build on for future use in other classes.

Another, quicker way to test classes as you write them is to use Interactions pane/tab of jGRASP. When you have written a little bit of code (i.e., implemented some fields or a single method of the class), you can compile that code in jGRASP as normal and then go to the "Interactions" tag in the bottom window of jGRASP (alongside "Compile Messages", "jGRASP Messages", and "Run I/O").

After you have compiled the class you're working on, selected the "Interactions" tab, and then clicked in that window at the bottom of jGRASP, you can interactively create and work with objects, inspecting the object contents (fields) in the "Workbench" window on the left side of jGRASP.

Using this method of code writing can speed up the development process dramatically because you can immediately catch issues with your code and test things immediately.

It is suggested that you write a method and then compile and test that method immediately instead of trying to write all of the methods at once and then test them. Once you are fairly certain one method works correctly, move on to the next method. You may discover in later testing that something may be broken and have to come back and fix it, but test each method as thoroughly as possible after you write it.

As an example, when writing the `Temperature` class, implement the instance variables and write the default constructor and then test it to make sure it works correctly.

Notice in the example on the next page that you're not required to use semicolons to end statements *in the Interactions tab* because you're not necessarily writing complete Java statements, you're just testing individual methods. Additionally, a number of different methods are being tested in the example image displayed, but only after they have each been individually tested incrementally.

The `Season` Class

The first, shortest, and easiest class you should write is the `Season` class. This class represents a season and should be saved in the file `Season.java`. Use the UML diagram below for requirements.

Season
- season : String
«constructor» Season() + getSeason(): String + setSeason(newSeason: String) : void + toString() : String + equals(other: Season) : boolean

Here are brief descriptions of the fields:

Field	Description	Notes
season	Stores name of current season	The value can be one of <code>spring</code> , <code>summer</code> , <code>fall</code> , <code>winter</code> , <code>autumn</code> in lowercase

Here are brief descriptions of all of the methods you should implement for the `Season` class. Make sure to name them and implement them exactly as you see them here as this is how Web-CAT will test them.

The `Season` Class, continued

PLEASE NOTE: The "Sample Interactions Tab Test Code" would be entered into the Interactions tab in jGRASP after you have added a method and successfully compiled the code. These samples are not comprehensive and may not test every single aspect of the methods. You should use these as jumping-off points for your own testing.

Method	Description	Sample Interactions Tab Test Code
<code>Season()</code>	Default constructor, defaults to <code>winter</code>	<pre>Season x = new Season(); Season y = new Season();</pre>
<code>getSeason()</code>	returns the current value of <code>season</code>	<pre>Season a = new Season() a.getSeason()</pre>
<code>setSeason(newSeason)</code>	Sets season to lowercase version of parameter <code>newSeason</code> if valid, regardless of capitalization (i.e., <code>SuMMer</code> and <code>summer</code> and <code>SUMMER</code> should all set <code>summer</code> for season); nothing done if <code>newSeason</code> is invalid.	<pre>Season b = new Season(); b.setSeason("summer") b.getSeason() b.setSeason("aUtUmN") b.getSeason() b.setSeason("potato") b.getSeason()</pre>
<code>toString()</code>	Simply returns the season, the same as <code>getSeason()</code>	<pre>Season c = new Season() c c.toString()</pre>
<code>equals(other)</code>	Returns equality of two <code>Season</code> objects if their seasons match; <code>autumn</code> and <code>fall</code> are considered the same.	<pre>Season d = new Season() Season e = new Season() d.equals(e) e.equals(d) e.setSeason("sumMER") d.equals(e)</pre>

The Temperature Class

The next class you should write is the `Temperature` class. This class represents a temperature and should be saved in the file `Temperature.java` .

Use the UML diagram below for requirements. Exact implementation is up to you. Pay careful attention to changing the scale or setting the temperature. For instance, if the temperature is set at 32° F and then the scale is changed to `C` with the `setScale()` method, how will you handle the temperature?

Temperature
- degrees : double - scale : char
«constructor» Temperature() «constructor» Temperature(temp : double, sc: char) + getTemp(): double + getScale() : char + set(temp: double, sc: char) : void + setTemp(temp: double) : void + setScale(sc: char) : void + toString() : String + equals(other: Temperature) : boolean

Here are brief descriptions of the fields:

Field	Description	Notes
degrees	Stores temperature	should be between -50° and +150° Fahrenheit, inclusive
scale	Indicates Celsius or Fahrenheit as	must be capital <code>C</code> or <code>F</code>

The `Temperature` Class, continued

Here are brief descriptions of all of the methods you should implement for the `Temperature` class. Make sure to name them and implement them exactly as you see them here as this is how Web-CAT will test them:

Method	Description	Sample Interactions Tab Test Code
<code>Temperature()</code>	Default constructor, sets temperature to 0° Celsius.	<code>Temperature a = new Temperature()</code>
<code>Temperature(temp,sc)</code>	Specific constructor, sets temperature and scale; make sure temperature falls within valid range and scale is correct.	<code>Temperature b = new Temperature(10,'C')</code> <code>Temperature c = new Temperature(-40,'F')</code> <code>Temperature d = new Temperature(-60,'F')</code> <code>Temperature e = new Temperature(151,'F')</code> <code>Temperature f = new Temperature(66,'C')</code> <code>Temperature f = new Temperature(-46,'C')</code> <code>Temperature g = new Temperature(10,'z')</code>
<code>getTemp()</code>	Returns the current temperature based on the scale.	<code>b.getTemp()</code> <code>c.getTemp()</code> <code>c.getTemp()</code> <code>d.getTemp()</code> <code>e.getTemp()</code> <code>f.getTemp()</code> <code>g.getTemp()</code>
<code>getScale()</code>	Returns the value of the <code>scale</code> field.	<code>b.getScale()</code> <code>c.getScale()</code> <code>c.getScale()</code> <code>d.getScale()</code> <code>e.getScale()</code> <code>f.getScale()</code> <code>g.getScale()</code>
<code>set(temp,sc)</code>	Sets temperature and scale; if either is invalid nothing happens	<code>b.set(10,'C')</code> <code>b.set(-40,'F')</code> <code>b.set(-60,'F')</code> <code>b.set(151,'F')</code> <code>b.set(66,'C')</code> <code>b.set(-46,'C')</code> <code>b.set(10,'z')</code>
<code>setTemp(temp)</code>	Validates the temperature for object's scale; no change to temperature if invalid; make sure to validate for given scale.	<code>Temperature j = new Temperature()</code> <code>j.getTemp()</code> <code>j.setTemp(12)</code> <code>j.getTemp()</code> <code>Temperature k = new Temperature(112,'F')</code> <code>k.setTemp(-14.2)</code>

The `Temperature` Class, continued even more

Method	Description	Sample Interactions Tab Test Code
<code>setScale(sc)</code>	Sets the scale; parameter can be either upper or lowercase, but stored scale should be uppercase <code>C</code> or <code>F</code> . If <code>sc</code> is different than the currently stored scale, consider how your object handles that.	<pre>Temperature h = new Temperature() h.getTemp() h.getScale() h.setScale('F') h.getTemp() h.getScale() h.setScale('C') h.getTemp()</pre>
<code>toString()</code>	returns <code>XX degrees Y</code> where <code>XX</code> is temperature (with all digits of precision) and <code>Y</code> is the scale. Example: <code>65.55555555555556 degrees C</code> .	<pre>Temperature b = new Temperature(10,'C') Temperature c = new Temperature(-40,'F') Temperature d = new Temperature(-60,'F') b c d.toString()</pre>
<code>equals(other)</code>	Returns equality of two <code>Temperature</code> objects regardless of scale, must be within 0.001 tolerance. In other words, one <code>Temperature</code> object that is storing 32° F and another that is storing 0° C should be equivalent if compared.	<pre>Temperature w = new Temperature() Temperature x = new Temperature() w.equals(x) x.equals(w) w.setTemp(10.2) w.equals(x)</pre>

The Weather Class

This class represents weather and should be saved in the file `Weather.java` . It depends on the `Temperature` and `Season` classes being fully and completely implemented first.

Use the UML diagram below for requirements.

Weather

- temp : Temperature

- humidity: int

- windspeed: int

- windDirection: String

- s: Season

«constructor» Weather()

«constructor» Weather(t: double, sc: char, whichSeason: String)

+ getSeason(): String

+ setSeason(newSeason: String) : void

+ getTemp() : Temperature

+ getHumidity() : int

+ getWindSp() : int

+ getWindDir() : String

+ setTemperature(t: Temperature) : void

+ setHumidity(h : int) : void

+ setWindSp(s : int) : void

+ setWindDir(dir: String) : void

+ toString() : String

+ equals(other: Season) : boolean

Here are brief descriptions of the fields:

Field	Description	Notes
temp	Stores the current temperature as a <code>Temperature</code> object	Rules of <code>Temperature</code> class apply
Humidity	Stores the value of the humidity	An <code>int</code> value between <code>0</code> and <code>100</code> inclusive
windSpeed	Stores the speed of the wind	Must be ≥ 0
windDirection	Stores the direction of the wind	Must be one of <code>N</code> , <code>E</code> , <code>S</code> , <code>W</code> , <code>NE</code> , <code>SE</code> , <code>SW</code> , <code>NW</code> ; must be stored in uppercase
s	Stores the current season as a <code>Season</code> object	Rules of <code>Season</code> class apply

The `Weather` Class, continued

Here are brief descriptions of all of the methods you should implement for the `Weather` class. Make sure to name them and implement them exactly as you see them here as this is how Web-CAT will test them.

PLEASE NOTE: Sample "Interactions" tab code has not been provided here to allow you to practice with developing your own testing methodology.

Method	Description
<code>Weather()</code>	The default constructor that sets a default <code>Temperature</code> , 50% humidity, windspeed of 0, wind from the <code>w</code> , default <code>Season</code>
<code>Weather(t,sc,whichSeason)</code>	Specific constructor that sets the <code>Temperature</code> field <code>temp</code> to <code>t</code> with scale <code>sc</code> , 50% humidity, windspeed of 0, wind from the <code>w</code> , and a <code>Season</code> object with season of <code>whichSeason</code> .
<code>getTemp()</code>	Returns <code>Temperature</code> object <code>temp</code>
<code>getHumidity()</code>	Returns the humidity of this object
<code>getWindSp()</code>	Returns the wind speed of this object
<code>getWindDir()</code>	Returns the windDirection of this object
<code>setTemperature(t)</code>	Sets a new temperature for the object
<code>setHumidity(h)</code>	Sets humidity with validation; does nothing otherwise
<code>setWindSp(s)</code>	Sets windspeed with validation; does nothing otherwise
<code>setWindDir(dir)</code>	Sets windDirection with validation; nothing otherwise
<code>toString()</code>	Returns The weather is currently TEMPERATURE with XX% humidity and a YY mph wind from the ZZ all on one line with exact spelling. TEMPERATURE is the output of the <code>temp</code> field, XX is the current value of the humidity, YY is the current value of the wind speed, and ZZ is the wind direction. Example: The weather is currently -10.0 degrees F with 50% humidity and a 0 mph wind from the W
<code>equals(other)</code>	Returns equality of two <code>Weather</code> objects with the temp and humidity of the two objects being equal.

The Driver Program: `Forecaster.java`

After these three classes have been developed, you also need to write a program with a `main()` method. This will be the driver for the entire collection of files/classes. This file should be saved as `Forecaster.java`.

This file will implement a menu-based program. All input should be validated and re-prompted until valid input is entered (i.e., a data validation loop should be used for all input). It should allow for the following options:

- 1. Set temperature preference (prompt for F or C until valid)
- 2. Set season (prompt for valid season)
- 3. Set weather (prompt for temp, humidity, windspeed in that order ... reprompt for all 4 if one or more values are invalid)
- 4. Get forecast (print forecast from below)
- 5. Print weather (print current weather)
- 6. Quit

In order to predict the weather for menu option 4, use the following information:

If the season for the Weather object is `winter` :

Output	Requirements
The forecast is frigid	The temperature is below 10° F and the wind is greater than 15 mph
The forecast is snow	The temperature is between 10° F and 30° F inclusive and the humidity is 80% or higher
The forecast is icy	The temperature is between 28° F and 33° F inclusive and the humidity is between 60 and 80 inclusive
The forecast is warm	The temperature is above 40° F
The forecast is cold	All other situations for <code>winter</code>

If the season for the Weather object is `summer` :

Output	Requirements
The forecast is steamy	The temperature is 32° C or above and the wind is less than 10 mph and the humidity at least 70%
The forecast is stormy	The temperature is 32° C or above and the wind is 20 mph or above and the humidity is at least 70%
The forecast is dry heat	The temperature is at least 30° C and the humidity is less than 50%
The forecast is hot and windy	The temperature is less than 32° C and at least 30° C and the wind is at least 20 mph
The forecast is hot	The temperature is at least 30° C
The forecast is warm	All other situations for <code>summer</code>

If the Season for the Weather object is `fall` or `autumn` :

Output	Requirements
The forecast is nice	The temperature is at least 65° F and less than 80° F and the wind is less than 15 mph and the humidity is 60% or less
The forecast is too warm	The temperature is at least 80° F
The forecast is chilly	The temperature is at least 40° F and less than 65° F and the wind is greater than 15 mph
The forecast is rainy	The humidity is at least 80
The forecast is typical	All other situations for <code>fall</code> or <code>autumn</code>

If the Season for the Weather object is `spring` :

Output	Requirements
The forecast is stormy	The temperature is greater than 65° F and less than or equal to 80° F and the wind is at least 20 mph and the humidity is at least 80%
The forecast is chilly	The temperature is less than 50° F
The forecast is windy	The humidity is less than 80% and the wind is at least 20 mph
The forecast is pleasant	All other situations for <code>spring</code>

Points to Think About

- You need to think about how you want to handle the conversions between scales in the `Temperature` class. You may store the value twice (once in C and once in F) with a "preferred" scale for display. Just remember to change both values if there's an update. You may store the value in one scale and then just convert upon display if needed. You may convert as needed and store the value in the preferred scale - just be careful when doing equals comparison that you convert to a single scale for comparison.
- Make sure both the `Temperature` and `Season` classes work before you attempt the `Weather` class.
- You may want to write `static` methods in the main `Forecaster` class in addition to `main()` to simplify the logic, but it's not required.
- Everything for `Season` should be in lower case.

Grading Notes

- You must have a completed Collaboration document in Canvas in order for the program to be graded.
- Your program should include the basic "header" documentation information seen in the example code. Replace the correct portions with your information.
- Look on Canvas for the "How to Do Your Program Header" file for more information under Resources
- Decimal values should be correct to the 3rd digit after the decimal point, though all digits will print (i.e., there is no need to restrict the output to just 3 digits of precision after the decimal point).

Grading Rubric

NOTE: If your submitted assignment does not compile, you will receive 0 points. We are unable to test code that does not compile. Insuring that your code compiles is critical when developing programs. We would rather grade an incomplete yet operational program than a program that we can't test.

Item	Points	Grader	Notes
Source code	8	GA	Does coding style make sense? Does program only use material covered in class? Are class designed sensibly? Are all instance variables / fields declared correctly?
Documentation and Style	7	WebCAT	Meet all Checkstyle coding style requirements for all files as discussed in Canvas.
Test Cases	30	WebCAT	Program must implement all methods correctly.
Extra Credit	Max +4.5	WebCAT	Submitting the program early is worth up to 4.5 points extra credit on the assignment, 1.5 points per day early. Submitting more than 3 days early defaults to 4.5 points extra credit.

There are two sample runs, which are both rather lengthy. The first sample run has all good input for all options. The second sample run demonstrates bad input and behaviors. Make sure you understand expected behavior.

Lengthy Sample Run With All Good Input:

```
Welcome to the Weather Forecaster.
```

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit

```
Enter choice: 5
```

```
The weather is currently 0.0 degrees C with 50% humidity and a 0 mph wind from the W
```

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit

```
Enter choice: 1
```

```
Enter temperature preference (C/F): F
```

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit

```
Enter choice: 2
```

```
Enter season: summer
```

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit

```
Enter choice: 3
```

```
Enter temperature: 8
```

```
Enter humidity: 30
```

```
Enter windspeed: 20
```

```
Enter wind direction: NW
```

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit

```
Enter choice: 5
```

The weather is currently 8.0 degrees F with 30% humidity and a 20 mph wind from the NW

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit

Enter choice: 4

The forecast is warm

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit

Enter choice: 1

Enter temperature preference (C/F): C

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit

Enter choice: 5

The weather is currently -13.3333333333332 degrees C with 30% humidity and a 20 mph wind from the NW

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit

Enter choice: 1

Enter temperature preference (C/F): F

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit

Enter choice: 5

The weather is currently 8.0 degrees F with 30% humidity and a 20 mph wind from the NW

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit

Enter choice: 2

Enter season: autumn
1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit
Enter choice: 4

The forecast is typical

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit
Enter choice: 2

Enter season: spring

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit
Enter choice: 4

The forecast is chilly

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit
Enter choice: 2

Enter season: FaLL

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit
Enter choice: 4

The forecast is typical

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit
Enter choice: 3

Enter temperature: 40
Enter humidity: 50
Enter windspeed: 20
Enter wind direction: W

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit

Enter choice: 5

The weather is currently 40.0 degrees F with 50% humidity and a 20 mph wind from the W

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit

Enter choice: 4

The forecast is chilly

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit

Enter choice: 1

Enter temperature preference (C/F): C

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit

Enter choice: 5

The weather is currently 4.44444444444445 degrees C with 50% humidity and a 20 mph wind from the W

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit

Enter choice: 4

The forecast is chilly

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast

5. Print the weather
6. Quit

Enter choice: 2

Enter season: spring

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit

Enter choice: 1

Enter temperature preference (C/F): F

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit

Enter choice: 2

Enter season: spring

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit

Enter choice: 3

Enter temperature: 75

Enter humidity: 50

Enter windspeed: 5

Enter wind direction: W

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit

Enter choice: 5

The weather is currently 75.0 degrees F with 50% humidity and a 5 mph wind from the W

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit

Enter choice: 4

The forecast is pleasant

```
1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit
Enter choice: 6
```

Lengthy Sample Run With Some Bad Input:

```
Welcome to the Weather Forecaster.
```

```
1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit
Enter choice: 0
```

```
Invalid selection.
```

```
1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit
Enter choice: 7
```

```
Invalid selection.
```

```
1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit
Enter choice: 1
```

```
Enter temperature preference (C/F): u
Enter temperature preference (C/F): Y
Enter temperature preference (C/F): l
Enter temperature preference (C/F): .
Enter temperature preference (C/F): &
Enter temperature preference (C/F): F
```

```
1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit
Enter choice: 2
```

```
Enter season: slimmer
Enter season: fault
```

Enter season: winner
Enter season: 78
Enter season: 2112
Enter season: winter

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit

Enter choice: 3

Enter temperature: -51
Enter humidity: 10
Enter windspeed: 20
Enter wind direction: NE
Invalid entry, try again.
Enter temperature: 151
Enter humidity: 10
Enter windspeed: 20
Enter wind direction: NE
Invalid entry, try again.
Enter temperature: 32
Enter humidity: -1
Enter windspeed: 10
Enter wind direction: NE
Invalid entry, try again.
Enter temperature: 32
Enter humidity: 101
Enter windspeed: 10
Enter wind direction: NE
Invalid entry, try again.
Enter temperature: 32
Enter humidity: 20
Enter windspeed: -1
Enter wind direction: NE
Invalid entry, try again.
Enter temperature: 32
Enter humidity: 20
Enter windspeed: 42
Enter wind direction: Y
Invalid entry, try again.
Enter temperature: 32
Enter humidity: 20
Enter windspeed: 25
Enter wind direction: EN
Invalid entry, try again.
Enter temperature: 32
Enter humidity: 20
Enter windspeed: 25
Enter wind direction: NE

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit

Enter choice: 5

The weather is currently 32.0 degrees F with 20% humidity and a 25 mph wind from the NE

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit

Enter choice: 4

The forecast is cold

1. Set temperature preference
2. Set season
3. Set weather
4. Get forecast
5. Print the weather
6. Quit

Enter choice: 6