Lab: Checker

A *checker* is an object that examines strings and *accepts* those strings that meet a particular criterion.

The `Checker` interface is defined below.

```
public interface Checker
{
  /**
   *  @param text a string to consider for acceptance
   *  @return true if this Checker accepts text;  false otherwise
   */
  boolean accept(String text);
}
```

In this question, you will write two classes that implement the `Checker` interface. You will then create a `Checker` object that checks for a particular acceptance criterion.

(a) A `SubstringChecker` accepts any string that contains a particular substring. For example, the following `SubstringChecker` object `broccoliChecker` accepts all strings containing the substring `"broccoli"`.

```
Checker broccoliChecker = new SubstringChecker("broccoli");
```

The following table illustrates the results of several calls to the `broccoliChecker accept` method.

Method Call	Result
`broccoliChecker.accept("broccoli")`	`true`
`broccoliChecker.accept("I like broccoli")`	`true`
`broccoliChecker.accept("carrots are great")`	`false`
`broccoliChecker.accept("Broccoli Bonanza")`	`false`

Write the `SubstringChecker` class that implements the `Checker` interface. The constructor should take a single `String` parameter that represents the particular substring to be matched.

(b) Checkers can be created to check for multiple acceptance criteria by combining other checker objects. For example, an `AndChecker` is a `Checker` that is constructed with two objects of classes that implement the `Checker` interface (such as `SubstringChecker` or `AndChecker` objects). The `AndChecker accept` method returns `true` if and only if the string is accepted by both of the `Checker` objects with which it was constructed.

In the code segment below, the `bothChecker` object accepts all strings containing both `"beets"` and `"carrots"`. The code segment also shows how the `veggies` object can be constructed to accept all strings containing the three substrings `"beets"`, `"carrots"`, and `"artichokes"`.

```
Checker bChecker = new SubstringChecker("beets");
Checker cChecker = new SubstringChecker("carrots");
Checker bothChecker = new AndChecker(bChecker, cChecker);

Checker aChecker = new SubstringChecker("artichokes");
Checker veggies = new AndChecker(bothChecker, aChecker);
```

The following table illustrates the results of several calls to the `bothChecker accept` method and the `veggies accept` method.

Method Call	Result
`bothChecker.accept("I love beets and carrots")`	`true`
`bothChecker.accept("beets are great")`	`false`
`veggies.accept("artichokes, beets, and carrots")`	`true`

Write the `AndChecker` class that implements the `Checker` interface. The constructor should take two `Checker` parameters.

(c) Another implementation of the `Checker` interface is the `NotChecker`, which contains the following:

- A one-parameter constructor that takes one `Checker` object
- An `accept` method that returns `true` if and only if its `Checker` object does NOT accept the string

A `NotChecker` accepts any string that does NOT contain a particular substring. For example, the following `NotChecker` object `broccoliChecker` accepts all strings NOT containing the substring `"broccoli"`.

```
Checker broccoliChecker = new SubstringChecker("broccoli");

Checker notBrocChecker = new NotChecker(broccoliChecker);
```

The following table illustrates the results of several calls to the `NotChecker accept` method.

Method Call	Result
`notBrocChecker.accept("broccoli")`	`false`
`notBrocChecker.accept("I like broccoli")`	`false`
`notBrocChecker.accept("carrots are great")`	`true`
`notBrocChecker.accept("Broccoli Bonanza")`	`true`

Write the `NotChecker` class that implements the `Checker` interface. The constructor should take one `Checker` parameter.

(d) Using any of the classes `SubstringChecker`, `AndChecker`, and `NotChecker`, construct a `Checker` that accepts a string if and only if it contains neither the substring `"artichokes"` nor the substring `"kale"`. Assign the constructed `Checker` to `yumChecker`. Consider the following incomplete code segment.

```
Checker aChecker = new SubstringChecker("artichokes");
Checker kChecker = new SubstringChecker("kale");
Checker yumChecker;
/* code to construct and assign to yumChecker */
```

The following table illustrates the results of several calls to the `yummyChecker` `accept` method.

Method Call	Result
`yumChecker.accept("chocolate truffles")`	`true`
`yumChecker.accept("kale is great")`	`false`
`yumChecker.accept("Yuck: artichokes & kale")`	`false`

In writing your solution, you may use any of the classes specified for this problem. Assume that these classes work as specified, regardless of what you wrote in parts (a) and (b). You may assume that the declarations for `aChecker`, `kChecker`, and `yumChecker` in the code segment above have already been executed.

Write your `/* code to construct and assign to yumChecker */`.