4. 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.

AP® COMPUTER SCIENCE A
2008 SCORING GUIDELINES

Question 4: Checker Objects (Design)

Part A: SubstringChecker **4 points**

+1/2 `class SubstringChecker implements Checker`

+1/2 declare private instance variable of type `String`

+1 constructor
- **+1/2** `SubstringChecker(String` *goalString*`)`
- **+1/2** initialize instance variable to parameter

+2 `accept` method
- **+1/2** `public boolean accept(String` *text*`)`
- **+1 1/2** determine whether to accept
  - **+1/2** attempt to find instance variable in *text*
    (either call `indexOf, contains,` or compare with substrings)
  - **+1** return correct boolean value in all cases

Part B: AndChecker **4 points**

+1/2 `class AndChecker implements Checker`

+1/2 declare private instance variable(s) capable of storing two `Checker` objects

+1 constructor
- **+1/2** `AndChecker(Checker` *c1*`, Checker` *c2*`)`
- **+1/2** initialize instance variable(s) to parameters

+2 `accept` method
- **+1/2** `public boolean accept(String` *text*`)`
- **+1 1/2** determine whether to accept
  - **+1/2** attempt to call `accept(`*text*`)` on both stored `Checker`s
  - **+1** return correct boolean value in all cases

Part C: yummyChecker **1 point**

+1 correctly assign `yummyChecker`