2. The `Book` class is used to store information about a book. A partial `Book` class definition is shown.

```
public class Book
{
   /** The title of the book */
   private String title;

   /** The price of the book */
   private double price;

   /** Creates a new Book with given title and price */
   public Book(String bookTitle, double bookPrice)
   {  /* implementation not shown */  }

   /** Returns the title of the book */
   public String getTitle()
   {  return title;  }

   /** Returns a string containing the title and price of the Book */
   public String getBookInfo()
   {
      return title + "-" + price;
   }

   // There may be instance variables, constructors, and methods that are not shown.
}
```

You will write a class `Textbook`, which is a subclass of `Book`.

A `Textbook` has an edition number, which is a positive integer used to identify different versions of the book. The `getBookInfo` method, when called on a `Textbook`, returns a string that also includes the edition information, as shown in the example.

Information about the book title and price must be maintained in the `Book` class. Information about the edition must be maintained in the `Textbook` class.

The `Textbook` class contains an additional method, `canSubstituteFor`, which returns `true` if a `Textbook` is a valid substitute for another `Textbook` and returns `false` otherwise. The current `Textbook` is a valid substitute for the `Textbook` referenced by the parameter of the `canSubstituteFor` method if the two `Textbook` objects have the same title and if the edition of the current `Textbook` is greater than or equal to the edition of the parameter.

GO ON TO THE NEXT PAGE.

The following table contains a sample code execution sequence and the corresponding results. The code execution sequence appears in a class other than `Book` or `Textbook`.

Statement	Value Returned (blank if no value)	Class Specification
`Textbook bio2015 = new Textbook("Biology", 49.75, 2);`		`bio2015` is a `Textbook` with a title of `"Biology"`, a price of `49.75`, and an edition of `2`.
`Textbook bio2019 = new Textbook("Biology", 39.75, 3);`		`bio2019` is a `Textbook` with a title of `"Biology"`, a price of `39.75`, and an edition of `3`.
`bio2019.getEdition();`	`3`	The edition is returned.
`bio2019.getBookInfo();`	`"Biology-39.75-3"`	The formatted string containing the title, price, and edition of `bio2019` is returned.
`bio2019.canSubstituteFor(bio2015);`	`true`	`bio2019` is a valid substitute for `bio2015`, since their titles are the same and the edition of `bio2019` is greater than or equal to the edition of `bio2015`.
`bio2015.canSubstituteFor(bio2019);`	`false`	`bio2015` is not a valid substitute for `bio2019`, since the edition of `bio2015` is less than the edition of `bio2019`.
`Textbook math = new Textbook("Calculus", 45.25, 1);`		`math` is a `Textbook` with a title of `"Calculus"`, a price of `45.25`, and an edition of `1`.
`bio2015.canSubstituteFor(math);`	`false`	`bio2015` is not a valid substitute for `math`, since the title of `bio2015` is not the same as the title of `math`.

Write the complete `Textbook` class. Your implementation must meet all specifications and conform to the examples shown in the preceding table.

Begin your response at the top of a new page in the Free Response booklet and fill in the appropriate circle indicating the question number.
If there are multiple parts to this question, write the part letter with your response.

3. Users of a website are asked to provide a review of the website at the end of each visit. Each review, represented by an object of the `Review` class, consists of an integer indicating the user's rating of the website and an optional `String` comment field. The comment field in a `Review` object ends with a period (`"."`), exclamation point (`"!"`), or letter, or is a `String` of length `0` if the user did not enter a comment.

```
public class Review
{
   private int rating;
   private String comment;

   /** Precondition: r >= 0
    *      c is not null.
    */
   public Review(int r, String c)
   {
      rating = r;
      comment = c;
   }

   public int getRating()
   {
      return rating;
   }

   public String getComment()
   {
      return comment;
   }

   // There may be instance variables, constructors, and methods that are not shown.
}
```

GO ON TO THE NEXT PAGE.

Question 2: Class — 9 points

Canonical solution

9 points

```
public class Textbook extends Book
{
   private int edition;

   public Textbook(String tbTitle, double tbPrice,
                   int tbEdition)
   {
      super(tbTitle, tbPrice);
      edition = tbEdition;
   }

   public int getEdition()
   {
      return edition;
   }

   public boolean canSubstituteFor(Textbook other)
   {
      return other.getTitle().equals(getTitle()) &&
             edition >= other.getEdition();
   }

   public String getBookInfo()
   {
      return super.getBookInfo() + "-" + edition;
   }
}
```

`Textbook`

	Scoring Criteria	Decision Rules	
1	Declares class header (must not be `private`): `class Textbook extends Book`		**1 point**
2	Declares constructor header: `public Textbook(String ___, double ___, int ___)`		**1 point**
3	Constructor calls `super` as the first line with the appropriate parameters		**1 point**
4	Declares appropriate `private` instance variable and uses appropriate parameter to initialize it	Responses **will not** earn the point if they • omit the keyword `private` • declare the variable outside the class, or in the class within a method or constructor • redeclare and use the instance variables of the superclass	**1 point**

5	Declares at least one required method and all declared headers are correct:
	`public boolean canSubstituteFor(Textbook ___)`
	`public int getEdition()`
	`public String getBookInfo()`
		Responses **will not** earn the point if they
		• exclude `public`
			1 point

6	`getEdition` returns value of instance variable	Responses **will not** earn the point if they • fail to create an instance variable for the edition	**1 point**
7	`canSubstituteFor` determines whether `true` or `false` should be returned based on comparison of book titles and editions (*algorithm*)	Responses **can** still earn the point even if they • fail to return (*return is not assessed for this method*) • access the edition without calling `getEdition` • redeclare and use the `title` variable of the superclass instead of calling `getTitle` Responses **will not** earn the point if they • fail to use `equals` • call `getTitle` incorrectly in either case	**1 point**
8	`getBookInfo` calls `super.getBookInfo`	Responses **can** still earn the point even if they • redeclare and use the instance variables of the superclass Responses **will not** earn the point if they • include parameters	**1 point**

9	Constructs information string	Responses **can** still earn the point even if they • call `super.getBookInfo` incorrectly • fail to call `super.getBookInfo` and access `title` and `price` directly • fail to return (*return is not assessed for this method*) Responses **will not** earn the point if they • omit the literal hyphen(s) in the constructed string • omit the edition in the constructed string • concatenate strings incorrectly	**1 point**

Question-specific penalties
None

Total for question 2 9 points

Applying the Scoring Criteria

Apply the question scoring criteria first, which always takes precedence. Penalty points can only be deducted in a part of the question that has earned credit via the question rubric. No part of a question (a, b, c) may have a negative point total. A given penalty can be assessed only once for a question, even if it occurs multiple times or in multiple parts of that question. A maximum of 3 penalty points may be assessed per question.

1-Point Penalty

v) Array/collection access confusion (`[] get`)
w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check)
x) Local variables used but none declared
y) Destruction of persistent data (e.g., changing value referenced by parameter)
z) Void method or constructor that returns a value

No Penalty

- Extraneous code with no side-effect (e.g., valid precondition check, no-op)
- Spelling/case discrepancies where there is no ambiguity*
- Local variable not declared provided other variables are declared in some part
- `private` or `public` qualifier on a local variable
- Missing `public` qualifier on class or constructor header
- Keyword used as an identifier
- Common mathematical symbols used for operators (× • ÷ ≤ ≥ <> ≠)
- `[]` vs. `()` vs. `<>`
- `=` instead of `==` and vice versa
- `length/size` confusion for array, `String`, `List`, or `ArrayList`; with or without `( )`
- Extraneous `[]` when referencing entire array
- `[i,j]` instead of `[i][j]`
- Extraneous size in array declaration, e.g., `int[size] nums = new int[size];`
- Missing `;` where structure clearly conveys intent
- Missing `{ }` where indentation clearly conveys intent
- Missing `( )` on parameter-less method or constructor invocations
- Missing `( )` around `if` or `while` conditions

Spelling and case discrepancies for identifiers fall under the "No Penalty" category only if the correction can be **unambiguously inferred from context, for example, "ArayList" instead of "ArrayList". As a counterexample, note that if the code declares `"int G=99, g=0;"`, then uses `"while (G < 10)"` instead of `"while (g < 10)"`, the context does **not** allow for the reader to assume the use of the lower case variable.*