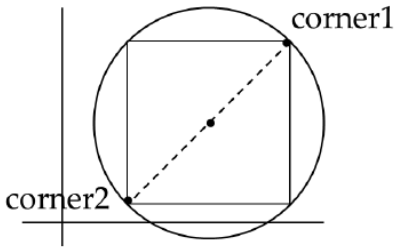
1. Fill in the representation invariant for this implementation of Circle. Assume that the concrete representation is two points directly across from each other, representing the endpoints of a diameter of the circle, as shown in the picture.

```
public class Circle3 {
     private Point corner1, corner2;
     // Abstraction function:
     // Rep invariant:
     //  __corner1 != null_&&_corner2 != null
     //  _____&& !corner1.equals(corner2)_______
}
```



2. `NonNullStringList` is a list of `Strings` with no `null` values in the list. Give two different concrete representations of `NonNullStringList` and write out the representation invariant for both. Note that your concrete representations must have some way to implement the three abstract operations provided (`add`, `remove`, `get`).
Hint: Recall the two implementations of List.

Concrete Representation 1:

```
public class NonNullStringList {
     // Rep invariant:
     //    arr != null && arr[0,count-1] != null &&
     //    count >= 0


     // Fields:
     private String[] arr;
     private int count;



     public void add(String s) { ... }
     public boolean remove(String s) { ... }
     public String get(int i) { ... }
}
```

```
Note: arr[0, count-1] is not code. It means from 0 to count-1,
including both sides. [] is kind of logical expression. [] means
including sides, () means not including sides.

For example: [0, 3] is 0, 1, 2, 3
(0, 3) is 1, 2
[0, 3) is 0, 1, 2
```

Concrete Representation 2:

```
public class NonNullStringList {
     // Rep invariant:
     // head.val != null, head.next.val != null, …
     // No cycle in ListNodes

     // Fields:
```

```
    public void add(String s) { ... }
    public boolean remove(String s) { ... }
    public String get(int i) { ... }
}
```

3. Comparing Specifications (18 sp mid)

Here are four possible specifications for a method `isMultiple` that checks whether one integer is a multiple of another.

```
public boolean isMultiple (int n, int f) {…
```

(S1)
```
@returns true if there exists g such that n = f * g
```

(S2)
```
@requires f ≠ 0
@returns true if there exists g such that n = f * g, otherwise false
```

(S3)
```
@requires f > 0
@returns true if there exists g such that n = f * g, otherwise false
```

(S4)
```
@returns true if there exists g such that n = f * g and g > 0,
         otherwise false
```

(a) Write transition relations for S2 and S3 and compare them.
S2: in the domain of S3, <(4,2), true>, <(2,1), true>, <(1,1), true>, <(1,3), false>, <(1, 4), false>
S3: <(4,2), true>, <(2,1), true>, <(1,1), true>, <(1,3), false>, <(1, 4), false>

S2 is part of S3, so S2 is stronger than S3.

(b) Write transition relations for S1 and S4 and compare them.

S1: <(4,2), true>, <(2,1), true>, <(4,-2), true>, <(1,3), false>, <(1, 4), false>
S4: <(4,2), true>, <(2,1), true>, <(4,-2), false>, <(1,3), false>, <(1, 4), false>

S1 is not part of S4 and S4 is not part of S1, so they are incomparable.

(c) Write out the logical formulas for S1 and S3 and compare them.

S1: true => (Nothing is modified AND returns true iff. ∃g s.t. n = f*g)
S3: (f > 0) => (Nothing is modified AND returns true iff. ∃g s.t. n = f*g, false otherwise)

(f > 0) => true
(Nothing is modified AND returns true iff. ∃g s.t. n = f*g, false otherwise) => (Nothing is modified AND returns true iff. ∃g s.t. n = f*g).

S3 has stronger requires and stronger returns, so is incomparable to S1

S1's Logical Formula does not imply S3's Logical Formula, nor is the reverse true, so S1 and S3 are incomparable.

(d) Write out the logical formulas for S2 and S4 and compare them

S2: (f ≠ 0) => (Nothing is modified AND returns true iff. ∃g s.t. n = f*g, false otherwise)
S4: true => (Nothing is modified AND returns true iff. ∃g s.t. g > 0 and n = f*g, false otherwise)

(Nothing is modified AND returns true iff. ∃g s.t. n = f*g, false otherwise) does not imply (Nothing is modified AND returns true iff. ∃g s.t. g > 0 and n = f*g, false otherwise), nor is the reverse true.

S2's Logical Formula does not imply S4's Logical Formula, nor is the reverse true, so S2 and S4 are incomparable.

4. `StringBag` implements a bag (a set that allows duplicate elements, i.e. a multiset) of Strings. `StringBag` uses an array to hold the elements and has an int `size` that records the size of the bag (the number of elements being used in the array). Assume that `items` will be replaced with a larger array and its elements will be copied over whenever `size` exceeds the size of the array.

```
import java.util.*;
public class StringBag {
   private int size; //# of strings in bag
   private String[] items; // the strings
   // constructor
   public StringBag(String[] vals) {
       size = vals.length;
       items = Arrays.copyOf(vals, size);  // new array copy of vals
   }

   //delete strings with length > n
   public void deleteLongStrings(int n) {
       int k = 0;
       while (k < size) {
         if (items[k].length() > n)
           { items[k] = items[size-
           1]; size = size - 1;
         } else {
           k = k + 1;
         }
       }
   }
```

```
   public boolean add(String s) {
       // not implemented
       return false;
   }
}
```

1) Give a suitable Representation Invariant (RI) for this class. (Remember that this RI should be sufficient to guarantee that the existing code executes successfully.)

items != null && 0 <= size <= items.length &&
for 0 <= k < size, items[k] != null
(It would also be possible to have a RI that allows items=null && size=0 if the StringBag is empty, which is more complicated, but could be done.)

2) The client would like to add an observer method `getItems` that returns an array with the `String`s that are currently in the `StringBag`. Below is our implementation of `getItems`:

```
// return the current strings in this StringBag to the caller
public String[] getItems() {
  return items;
}
```

Is this method correct? In other words, does it return the correct information to the client?

No. It returns the entire contents of the items array, even though some or all of the array elements might not be defined since they are in the section of the array items[size..items.length-1], i.e., the part of the array that is not being currently used.

3) Are there any potential representation exposure or other problems with this method? If so, what can go wrong? Otherwise, briefly explain why not.

Yes. Client code could alter the contents of the StringBag by modifying elements of the returned array, and could cause store nulls in the array, which would violate the representation invariant.

be done to fix the problems. You do not need to write any code, but you can if it helps illustrate your answer.

The most reasonable solution is for getItems to allocate a new String array with length equal to size, copy the contents of items[0..size-1] to the new array, and return that new array to the caller. One good way to do this would be to write return Arrays.copyOf(items,size);.
(Note: this does not create a representation exposure problem, since Strings are immutable. There is no need to make copies of the strings themselves.)

5. IntStack (code is on next page) (17 sp mid)

(a) Give a suitable abstract description of the class as would be written in the JavaDoc comment above the `IntStack` class heading.

An IntStack is a finite stack of integers with a fixed capacity. A typical value would be s0, s1, s2, …, sn, where s0 is at the bottom of the stack and sn is at the top.

(b) Give a suitable Representation Invariant (RI) for this class. (Remember that this RI should be sufficient to guarantee that the existing code executes successfully.)

vals is not null, top >= 0, and for 0 <= k < top, vals[k] has been initialized with stack elements.
Note: it would be incorrect to say that vals[k] is not null, since int values cannot be null – they are not references.

```
public class IntStack {
  private int[] vals;  //stack
  private int top;     // top
  private static int defaultCapacity = 100;

  // construct new stack with default capacity
  public IntStack() {
    vals = new
    int[defaultCapacity]; top = 0;
  }

  // construct new stack with given capacity
  public IntStack(int capacity) {
    vals = new
    int[capacity]; top = 0;
  }

  // operations
  public boolean push(int x) {
    if (top == vals.length)
      return false;
```

```
    return true;
  }

  public int pop() {
    if (top == 0) { // throw runtime exception if empty
      throw new NoSuchElementException();
    }
    top--;
    return vals[top];
  }

  public int size() {
    return top;
  }
}
```