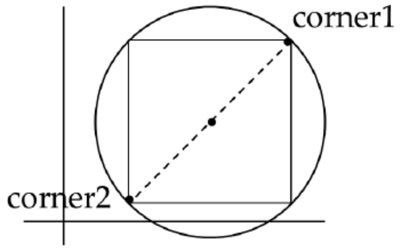
1. Fill in the representation invariant for this implementation of Circle. Assume that the concrete representation is two points directly across from each other, representing the endpoints of a diameter of the circle, as shown in the picture.

```
 public class Circle3 {
     private Point corner1, corner2;
     // Abstraction function:
     // Rep invariant:
     // ________________________________________
     // ________________________________________
}
```



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:



     // Fields:




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

Concrete Representation 2:
```
public class NonNullStringList {
     // Rep invariant:



     // Fields:




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

3. Comparing Specifications

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.

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

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

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

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;
          }
        }
    }


    // add string to bag (expand bag as needed) and return success
    public boolean add(String s) {
        // not implemented
        return false;
   }
}
```

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

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?

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

4) If there are problems with `getItems`, describe how to fix them and still provide an observer method that supplies the information desired by the caller. Please briefly describe what needs to be done to fix the problems. You do not need to write any code, but you can if it helps illustrate your answer.

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.

(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.)

```
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;
    vals[top] = x;
    top++;
    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;
  }
}
```