Digital sounds can be represented as an array of integer values. For this question, you will write two unrelated methods of the `Sound` class.

A partial declaration of the `Sound` class is shown below.

```
public class Sound
{
  /** the array of values in this sound; guaranteed not to be null */
  private int[] samples;

  /** Changes those values in this sound that have an amplitude greater than limit.
   *  Values greater than limit are changed to limit.
   *  Values less than -limit are changed to -limit.
   *  @param limit the amplitude limit
   *         Precondition: limit ≥ 0
   *  @return the number of values in this sound that this method changed
   */
  public int limitAmplitude(int limit)
  {  /* to be implemented in part (a) */  }

  /** Removes all silence from the beginning of this sound.
   *  Silence is represented by a value of 0.
   *  Precondition: samples contains at least one nonzero value
   *  Postcondition: the length of samples reflects the removal of starting silence
   */
  public void trimSilenceFromBeginning()
  {  /* to be implemented in part (b) */  }

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

(a) The volume of a sound depends on the amplitude of each value in the sound. The amplitude of a value is its absolute value. For example, the amplitude of -2300 is 2300 and the amplitude of 4000 is 4000.

Write the method `limitAmplitude` that will change any value that has an amplitude greater than the given limit. Values that are greater than `limit` are replaced with `limit`, and values that are less than `-limit` are replaced with `-limit`. The method returns the total number of values that were changed in the array. For example, assume that the array `samples` has been initialized with the following values.

40	2532	17	-2300	-17	-4000	2000	1048	-420	33	15	-32	2030	3223

When the statement

```
int numChanges = limitAmplitude(2000);
```

is executed, the value of `numChanges` will be 5, and the array `samples` will contain the following values.

40	2000	17	-2000	-17	-2000	2000	1048	-420	33	15	-32	2000	2000

Complete method `limitAmplitude` below.

```
/** Changes those values in this sound that have an amplitude greater than limit.
 *  Values greater than limit are changed to limit.
 *  Values less than -limit are changed to -limit.
 *  @param limit the amplitude limit
 *         Precondition: limit ≥ 0
 *  @return the number of values in this sound that this method changed
 */
public int limitAmplitude(int limit)
```

(b) Recorded sound often begins with silence. Silence in a sound is represented by a value of 0.

Write the method `trimSilenceFromBeginning` that removes the silence from the beginning of a sound. To remove starting silence, a new array of values is created that contains the same values as the original `samples` array in the same order but without the leading zeros. The instance variable `samples` is updated to refer to the new array. For example, suppose the instance variable `samples` refers to the following array.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Value	0	0	0	0	-14	0	-35	-39	0	-7	16	32	37	29	0	0

After `trimSilenceFromBeginning` has been called, the instance variable `samples` will refer to the following array.

Index	0	1	2	3	4	5	6	7	8	9	10	11
Value	-14	0	-35	-39	0	-7	16	32	37	29	0	0

Complete method `trimSilenceFromBeginning` below.

```
/** Removes all silence from the beginning of this sound.
 *  Silence is represented by a value of 0.
 *  Precondition: samples contains at least one nonzero value
 *  Postcondition: the length of samples reflects the removal of starting silence
 */
public void trimSilenceFromBeginning()
```

Question 1: Sound

Part (a) `limitAmplitude` **4½ points**

Intent: Change elements of `samples` *that exceed* `±limit`; *return number of changes made*

+3 Identify elements of `samples` to be modified and modify as required
- **+1** Consider elements of `samples`
  - **+½** Accesses more than one element of `samples`
  - **+½** Accesses every element of `samples` (*no bounds errors*)
- **+2** Identify and change elements of `samples`
  - **+½** Compares an element of `samples` with `limit`
  - **+½** Changes at least one element to `limit` or `-limit`
  - **+1** Changes all and only elements that exceed `±limit` to `limit` or `-limit` appropriately

+1½ Calculate and return number of changed elements of `samples`
- **+1** Initializes and updates a counter to achieve correct number of changed samples
- **+½** Returns value of an updated counter (*requires array access*)

Part (b) `trimSilenceFromBeginning` **4½ points**

Intent: Remove leading elements of `samples` *that have value of* 0, *potentially resulting in array of different length*

+1½ Identify leading-zero-valued elements of `samples`
- **+½** Accesses every leading-zero element of `samples`
- **+½** Compares 0 and an element of `samples`
- **+½** Compares 0 and multiple elements of `samples`

+1 Create array of proper length
- **+½** Determines correct number of elements to be in resulting array
- **+½** Creates new array of determined length

+2 Remove silence values from `samples`
- **+½** Copies some values other than leading-zero values
- **+1** Copies all and only values other than leading-zero values, preserving original order
- **+½** Modifies instance variable `samples` to reference newly created array

Question-Specific Penalties

–1 Array identifier confusion (e.g., `value` instead of `samples`)

–½ Array/collection modifier confusion (e.g., using `set`)

Question 1: Sound

Part (a):

```
public int limitAmplitude(int limit) {
  int numChanged = 0;
  for (int i = 0; i < this.samples.length; i++) {
    if (this.samples[i] < -limit) {
      this.samples[i] = -limit;
      numChanged++;
    }
    if (this.samples[i] > limit) {
      this.samples[i] = limit;
      numChanged++;
    }
  }
  return numChanged;
}
```

Part (b):

```
public void trimSilenceFromBeginning() {
  int i = 0;
  while (this.samples[i] == 0) {
    i++;
  }
  int[] newSamples = new int[this.samples.length - i];
  for (int j = 0; j < newSamples.length; j++) {
    newSamples[j] = this.samples[j+i];
  }
  this.samples = newSamples;
}
```