International

4. A manufacturer wants to keep track of the average of the ratings that have been submitted for an item using a running average. The algorithm for calculating a running average differs from the standard algorithm for calculating an average, as described in part (a).

A partial declaration of the `RunningAverage` class is shown below. You will write two methods of the `RunningAverage` class.

```
public class RunningAverage
{
    /** The number of ratings included in the running average. */
    private int count;

    /** The average of the ratings that have been entered. */
    private double average;

    // There are no other instance variables.

    /** Creates a RunningAverage object.
    * Postcondition: count is initialized to 0 and average is
    * initialized to 0.0.
    */


    public RunningAverage()
    { /* implementation not shown */ }

    /** Updates the running average to reflect the entry of a new
    * rating, as described in part (a).
    */

    public void updateAverage(double newVal)
    { /* to be implemented in part (a) */ }

    /** Processes num new ratings by considering them for inclusion
    * in the running average and updating the running average as
    * necessary. Returns an integer that represents the number of
    * invalid ratings, as described in part (b).
    * Precondition: num > 0
    */

    public int processNewRatings(int num)
    { /* to be implemented in part (b) */ }

    /** Returns a single numeric rating.
    */public double getNewRating()
    { /* implementation not shown */ }

}
```

International

(a) Write the method `updateAverage`, which updates the `RunningAverage` object to include a new rating. To update a running average, add the new rating to a calculated total, which is the number of ratings times the current running average. Divide the new total by the incremented count to obtain the new running average.

For example, if there are 4 ratings with a current running average of 3.5, the calculated total is 4 times 3.5, or 14.0. When a fifth rating with a value of 6.0 is included, the new total becomes 20.0. The new running average is 20.0 divided by 5, or 4.0.

Complete method `updateAverage`.

```
/** Updates the running average to reflect the entry of a new
 * rating, as described in part (a).
 */
public void updateAverage(double newVal)
{
```

(b) Write the `processNewRatings` method, which considers `num` new ratings for inclusion in the running average. A helper method, `getNewRating`, which returns a single rating, has been provided for you.

The running average must only be updated with ratings that are greater than or equal to zero. Ratings that are less than 0 are considered invalid and are not included in the running average.

The `processNewRatings` method returns the number of invalid ratings. See the table below for three examples of how calls to `processNewRatings` should work.

Statement	Ratings Generated	`processNewRatings` Return Value	Comments
`processNewRatings(2)`	`2.5, 4.5`	`0`	Both new ratings are included in the running average.
`processNewRatings(1)`	`-2.0`	`1`	No new ratings are included in the running average.
`processNewRatings(4)`	`0.0, -2.2, 3.5, -1.5`	`2`	Two new ratings (`0.0` and `3.5`) are included in the running average.

Complete method `processNewRatings`. Assume that `updateAverage` works as specified, regardless of what you wrote in part (a). You must use `getNewRating` and `updateAverage` appropriately to receive full credit.

```
/** Processes num new ratings by considering them for inclusion
 * in the running average and updating the running average as
 * necessary. Returns an integer that represents the number of
 * invalid ratings, as described in part (b).
 * Precondition: num > 0
*/
public int processNewRatings(int num)
{
```