B.  Write the LotteryTicket method  getPercentAccuracy(),as started below. getPercentAccuracy() will receive a list of the winning lottery numbers and determine how many of the numberList numbers match the numbers in the winning lottery number list.   
  
   Assuming numberList contains [1,2,3,4,5,6,7,8,9,10]  
   Assuming winningNumbers contains [11,12,3,14,5,16,17,8,9,10]  
  
   The call getPercentAccuracy() would return .5 because 50 percent of the numbers from numberList match the numbers from winningNumbers.  
  
   You may call the method from part a, assuming the method works as specified regardless of what you wrote.  
     
      /\*\*       @param winningNumbers contains 10 positive non-decimal values  
      \*              **Precondition :**  length == 10  
      \*        @return the percentage of values in numberList that match values in winningNumbers  
     \*/

My answer:

  public double getPercentAccuracy(int[] winningNumbers)  
      {

          double p = 0;

          int i = 0;  
           for(int item : winningNumbers)

            {

             if(numberList.search(item)==true)

              {

               if(item==numberList(i))

               p++;  
              }

             i++;  
           }  
        return p/10.0;   
      }

Correct answer:

   public double getPercentAccuracy(int[] winningNumbers)  
      {

        double p = 0;

        for(int item : winningNumbers)

             if(search(item))

p++;

return p/10.0;

}

In my original answer, I wasn’t sure whether or not the numbers had to have the same indexes or if they just had to be in the array to count as accurate. I also used an object to call a method which I did not need to do in this case because the methods were in the same class and both non-static.

B.  Write the Row method  tallestBlock().  tallestBlock() will go through the list and return the Block that is the tallest.  The tallest Block is the block with the greatest height.      
  
// myBlocks contains at least one Block

My answer:

**public**Block tallestBlock()  
{   
 int tall = Integer.MIN\_VALUE;

 for(int i = 0; i<myBlocks.size(); i++)

  {

   if(myBlocks.get(i).getHeight()>tall)

    tall = i;

   }

 return myBlocks.get(tall).getHeight();  
  
}

Correct answer:

**public**Block tallestBlock()  
{   
 Block tall = new Block();

 for(Block item : myBlocks)

    tall.setHeight(Math.max(item.getHeight(),tall.getHeight()));

 return tall;  
  
}

In my original answer, I returned an int instead of a block object. I fixed this issue by creating a new block that will be set to the block object in myBlocks that has the tallest height. I also made my code more efficient with the use of Math.max and a for each loop.

C.  Write the Row method  level().  level() will go through the list and change the height of every Block to the height of the tallest Block.  You may use tallestBlock from part B and assume it works as intended.  
  
**My answer:**

**public void** level()        
{ 

for(int i = 0; i<myBlocks.size(); i++)

   if(myBlocks.get(i).getHeight()!= myBlocks.tallestBlock())

    myBlocks.get(i).setHeight(myBlocks.tallestBlock));  
        
  
}

**Correct answer:**

**public void** level()        
{

for(Block item : myBlocks)

item.setHeight(tallestBlock().getHeight());  
}

My original code was messed up because I was comparing an int to a block and putting a block with parameters that called for an int. I also used an ArrayList to call a method which I did not need to do because they are both non-static and in the same class. I made my code shorter by using a for each loop and getting rid of the wrong and unnecessary if statement.