1. **Model**

   Suppose that you wanted to create a graphical user interface for the game tic-tac-toe. What data would the model contain?

2. **Implement the image viewer model**

   (a) Implement the `getCurrentImage` method for the image viewer.

```
/**
 * Get the current image in the model. Returns an empty image if the
 * model has no images.
 *
 * @return the current image if the model has one or more images, or an
 *         empty image otherwise
 */
public ImageIcon getCurrentImage() {




}
```

   (b) Implement the `getNextImage` method for the image viewer.

```
/**
 * Get the next image in the model and makes the next image the current
 * image. Returns the current image if the current image is the last
 * image. Returns an empty image if the model has no images.
 *
 * @return the next image in the model
 */
public ImageIcon getNextImage() {









}
```

3. **View**

   Suppose that you wanted to create a graphical user interface for the game tic-tac-toe. Imagine how the player would interact with the view. Sketch what the view would look like. What components (buttons, labels, etc) would make up the view?

4. **Controller**

   Suppose that graphical user interface for your tic-tac-toe application has 9 buttons, one for each of the possible locations on the 3-by-3 grid that the game is played on. When the player clicks on one of the buttons to play their turn what sequence of events does the controller need to orchestrate?

5. **Recursion base cases**

State the base case for each of the following recursive methods:

(a)
```
/**
 * Return the sum of the integers 1 through n where n is a strictly
 * positive integer. Note that the sum might overflow if n is too
 * large; this method does not check if the sum overflows (i.e.,
 * it's the client's problem).
 *
 * @pre. n is greater than 0
 *
 * @param n
 *            a strictly positive number
 * @return the sum 1 + 2 + ... + n
 */
public static int sum(int n)
```

(b)
```
/**
 * Returns a new string equal to the reversal of string s. The
 * reversal of the empty string is equal to the empty string.
 *
 * @pre. s is not null
 *
 * @param s
 *            a string
 * @return a string equal to the reversal of s
 */
public static String reverse(String s)
```

(c)
```
/**
 * Returns the minimum element in the list t. This method does not
 * modify the list <code>t</code>.
 *
 * @pre. t.size() is greater than 0
 *
 * @param t
 *            a non-empty list
 * @return the minimum element in t
 */
public static int min(List<Integer> t)
```

6. **Recursion recursive cases**

State the recursive case for each of the following recursive methods:

(a)
```
/**
 * Return the sum of the integers 1 through n where n is a strictly
 * positive integer. Note that the sum might overflow if n is too
 * large; this method does not check if the sum overflows (i.e.,
 * it's the client's problem).
 *
 * @pre. n is greater than 0
 *
 * @param n
 *            a strictly positive number
 * @return the sum 1 + 2 + ... + n
 */
public static int sum(int n)
```

(b)
```
/**
 * Returns a new string equal to the reversal of string s. The
 * reversal of the empty string is equal to the empty string.
 *
 * @pre. s is not null
 *
 * @param s
 *            a string
 * @return a string equal to the reversal of s
 */
public static String reverse(String s)
```

(c)
```
/**
 * Returns the minimum element in the list t. This method does not
 * modify the list <code>t</code>.
 *
 * @pre. t.size() is greater than 0
 *
 * @param t
 *            a non-empty list
 * @return the minimum element in t
 */
public static int min(List<Integer> t)
```