Consider the partially completed Java language code fragments for classes `Image`, `ImageSensor`, `ImageProcessor`, and `ImagingSystem` provided below.

The `ImageSensor` captures an `Image` every 5000 milliseconds and inserts to a frame buffer. The `ImageProcessor` removes each `Image` from the buffer and processes it. The system is implemented as an embedded system powered by a battery.

Assume an execution scenario of the `ImagingSystem` that has one `ImageSensor` instance and one `ImageProcessor` instance as shown in Figure Q2.1.

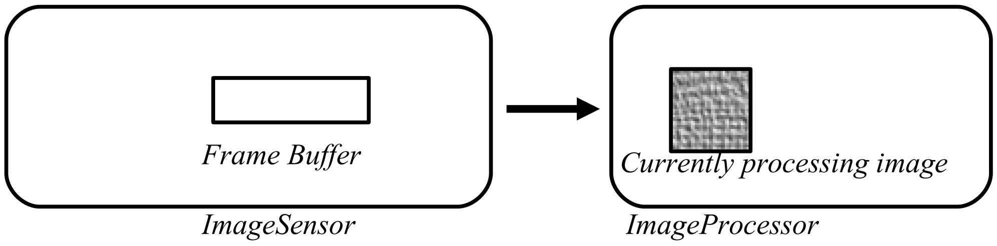

Figure Q2.1

```
class Image {
  ...
}

class ImageSensor implements Runnable {
 private Image [] buffer;
 private int size;
 private int index;

 ImageSensor(int size) {
  this.size = size; index = 0; buffer = new Image[size];
 }

 // insert a new image object to the first free slot in the buffer
 // if buffer is full, the image object is discarded
 public void insert(Image img) {
  ...
 }

 // remove the image object in the first slot (the oldest image) in the
buffer
 // if buffer is empty, a null value is returned
 public Image remove() {
  Image img=null;
  ...
  return img;
 }

 public void run() {
  ...
  while(true) {
   try {
    Thread.sleep(5000);
    insert(new Image());
   } catch(InterruptedException e) {
    ...
   }
  }
 }
}

class ImageProcessor implements Runnable {
 private ImageSensor sensor;

 ImageProcessor (ImageSensor sensor) {
  this.sensor = sensor;
 }

 private void process() {
  ...
 }

 public void run() {
  Image img=null;

  while(true) {
   try {
    img = sensor.remove();
    if(img != null)
     process();
   } catch(InterruptedException e) {
    ...
   }
  }
 }
}

class ImagingSystem {
 public static void main(String [] args) {
  ...
 }
}
```

(a) Implement the `insert()` and `remove()` methods in `ImageSensor` class in a thread-safe manner. [06]

(b) Considering the implementation of `run()` methods in `ImageSensor` class and [04]

`ImageProcessor` class, briefly discuss one positive aspect and one negative aspect of this implementation.

(c) Instead of the solution given above, provide an alternative solution using observer-observable design pattern. Also implement the `main()` method in `ImagingSystem` class for this new execution scenario. NOTE: Do NOT use guarded blocks. [15]