Kapil Gowru
Mathew Kurian

Homework 2

Question 1

Rename.java (portion)

```
public Result rename() {

   boolean search = true;
   int down = 0, right = 0, id = -1;

   while (search) {

      Splitter splitter = null;

      try {
         splitter = m_splitters[down][right];
      } catch (Exception e) {
         return null;
      }
```

```
        if (splitter == null) {
            return null;
        }

        Direction direction = splitter.getDirection(Thread.currentThread()
                .getId());

        switch (direction) {
        case DOWN: {
            down++;
            break;
        }
        case RIGHT: {
            right++;
            break;
        }
        case STOP: {
            id = getId(down, right, m_range); // Use the function below to
calculate id from the returned Result
            search = false;

            break;
        }
        }
    }

    return new Result(down, right, id);
}

public static int getId(int down, int right, int range) {

    int y = down * (down + 1) / 2 + 1;
    int x = 0;

    // Arithmetic series
    if (right > 0) {
        int dx = down + 1;
        int ex = dx + right;
        int sx = dx * (dx + 1) / 2;
        x = ex * (ex + 1) / 2 - sx;
    }

    // Based on grid position not the previous id
    return x + y;
}
```

Splitter.java (portion)

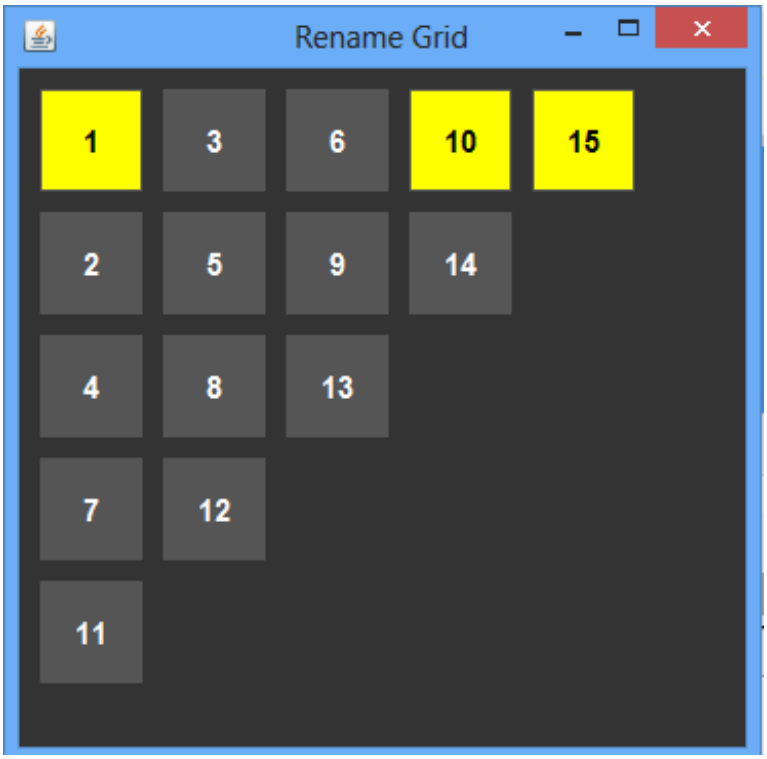
```
public Direction getDirection(long pid) {

   m_pid.set(pid);

   if (m_stopped.get()) {
      return Direction.RIGHT;
   } else {
      m_stopped.set(true);
      if (m_pid.get() == pid) {
         return Direction.STOP;
      } else {
         return Direction.DOWN;
      }
   }
}

public void release() {
   m_stopped.set(false);
}
```

Visual



Question 2

If you have an atomic read and non-atomic write, then the values returned during any duration of the write can yield an unexpected value. This behavior can cause the Bakery algorithm to fail. Bakery will fail in this case because it won't follow fairness anymore.

Thread A	**Thread B**	**Labels**
Enter CS	Waiting in while-loop	label[A] = 1; label[B] = 2;
Exits CS	OS halts	label[A] = 0; label[B] = 2;
Lock called	…	label[A] = 0; label[B] = 2;
Write label[A] = 3	Reads wrong label. Enters CS.	label[A] = -1; (Thread B reads Thread A wrong)
Enters CS	OS pauses	label[A] = 3; label[B] = 2;

Thread A is able to go around and enter, while thread B has to wait; hence, fairness was not upheld

Question 3

Graph