1) getPixel and setPixel (=putPixel) methods may be preferred instead of a board matrix.

```
public static void floodFill(int x, int y, int fillColor)
 {
        if (board[y][x] == 0)
        {
                board[y][x] = 2;
                floodFill(x+1,y,fillColor);
                floodFill(x,y+1,fillColor);
                floodFill(x-1,y,fillColor);
                floodFill(x,y-1,fillColor);
        }
 }
```

2) Optional : Generic Vector , Customer and Checkout Classes may be defined (not needed).

1

```
class MQueue
{
 Vector mQ;
 private int customers, TotalWaitingTime;
 public double AvgWaitingTime;

 public MQueue()
 { mQ = new Vector();  customers = 0;
   TotalWaitingTime=0; AvgWaitingTime=0; }
…
}
```

2

```
public void enque()
 {  if (mQ.isEmpty())
   { Vector cQ = new Vector(); cQ.add(1); mQ.add(cQ); }
     else {
       for(int i=0;i<mQ.size();++i) {
       Vector cQ = (Vector)mQ.elementAt(i);
       if(cQ.size()<3)  { cQ.add(cQ.size()+1); return; };
     };
    Vector cQ = new Vector(); cQ.add(1); mQ.add(cQ);
   }
 }
```

3

```
public int deque()
{
  int cNumber = (int)(Math.random()*mQ.size());
  Vector cQ = (Vector)mQ.elementAt(cNumber);
  int WaitingTime = (Integer)(cQ.remove(0));
  customers++;  TotalWaitingTime += WaitingTime;
  AvgWaitingTime = TotalWaitingTime/(double)customers;
  if(cQ.size()==0) mQ.remove(cNumber);
  return WaitingTime;
}
```

3) Alternative Methods and Solutions exist.

```
public void count(TreeNode localRoot)
{  if(localRoot!=null)
    {   level++;
        count(localRoot.leftChild);
        if (level>maxDepth) maxDepth = level; nodes++;
        count(localRoot.rightChild);
        level--;
    }
}
```

```
public boolean isFull()
{
  level=0; nodes=0; maxDepth=0;
  count(root);
  if(nodes==Math.pow(2,maxDepth)-1)
    return true;
  return false;
}
```