Pregunta 1

precondición: none(tautologia,siempre aplicable)

postcondicioń: return value is CELL EMPTY, CELL CROSS,or CELL, NAUGHT
if row >= 0 && row < TOTAL.ROWS && column >= 0 && column < TOTAL_COLUMN
otherwise null

Pregunta 2

Precondición: none

Postcondición: return CELL EMPTY, CELL CROSS or CELL.NAUGHT
if <= row < TOTAL.ROWS and 0 <= column < TOTAL.COLUMNS
otherwise throw new OutOfBoardException

max >= list[i](3 >= 5)

Pregunta 3

uch that max=list[j]
Precondición: list.length>0
Postcondición: max>= list[i] for each i
(0 ≤ i< list.length), and there exists j
(0 ≤ j< list.length) such that max=list[j]

```
int max(int[] list){
      int result=list[0];
      for (int i=0; i<list.length-1; i++){
           if (result<list[i])
            result=list[i];
      }
      return result;
}
```

list= [3, 2, 5] satiface la precondición list.length > 0, max(list) = 3
max >= list[i](3 >= 5) para i= 2(falso)
Por lo tanto la implementación anterior no es correcta

Pregunta 4

Precondición: p.length > 0
Poscondición (v2): q.length = p.length y q[i] <= q[i +1] para cualquier i, 0 <= i < q.length −1 y para cualquier j (0 <= j < p.length), existe k (0 <= k < q.length) tal que p[j] = p[k] y para cualquier j (0 <= j < q.length) existe k (0 <= k < p.length) tal que q[j] = p[k].

```
public int [ ] sort (int p [ ]){
   int[ ] q = new int [p.length];
   for (int i = 0; i < p.length; i ++)
       q[i] = 1;
   return q;
}
```

p = [2, 1, 2, 3] y q = [1, 2, 3, 3] cumple la postcondicioń pero q y p no tienen la misma cantidad de cada elemento.
Por lo tanto la implementación anterior no es correcta

Pregunta 5

```
public boolean purchase(String drink){
        if (drink.equalsIgnoreCase(COFFEE)){
            if (coffee.getCount()>0&&deposit>=coffee.getPrice()){
                coffee.sell();
                calculateChange(coffee.getPrice());
                return true;
            }
        }
   ...
}
```

En el código anterior falta actualizar la cantidad de cafés, dar vuelto y volver el deposito a cero. Con esos cambios el código quedaría:

```
public boolean purchase(String drink){
        if (drink.equalsIgnoreCase(COFFEE)){
            if (coffee.getCount()>0&&deposit>=coffee.getPrice()){
                coffee.sell();
                coffe.setCount(coffe.getCount() - 1);
                calculateChange(coffee.getPrice());
                giveChange();
                setDeposit(0);
                return true;
            }
        }
   ...
}
```

Pregunta 6

```
public class PrePostcondiciones {

  /**
   * @param list precondición: list.length > 0
   * @return q postcondición: q[i] = list[list.length - 1 - i] para cada i en [0, list.length - 1)
   */
  int[] reverse(int[] list) {
    int[] q = new int[list.length];
    for (int i = 0; i < list.length; i++) {
      q[i] = list[list.length - 1 - i];
    }
    return q;
  }

  /**
   * @param list precondición: list.length > 0
   * @return ind postcondición: list[ind] = key si key está en list caso contratio ind = -1
   */
  int linearSearch(int[] list, int key) {
    for (int i = 0; i < list.length; i++) {
      if (list[i] == key) {
        return i;
      }
    }
    return -1;
  }

  /**
   * @param letra precondición: letra es un caracter
   * @return q postcondición: q es true si letra es vocal en otro caso es false
   */
  boolean isVowel(char letra) {
    switch (letra) {
      case 'a':
      case 'e':
      case 'i':
      case 'o':
      case 'u':
      case 'A':
      case 'E':
      case 'I':
      case 'O':
      case 'U':
        return true;
      default:
        return false;
    }
  }

  /**
   * @param year precondición: year > 0
   * @return q postcondición: q = True si year es bisiesto y q = False caso contrario
   */
  boolean isLeapYear(int year) {
    if (year % 4 == 0) {
      if (year % 100 == 0) {
        if (year % 400 == 0) {
          return true;
        } else {
          return false;
        }
      } else {
        return true;
      }
    } else {
      return false;
    }
  }
```

```java
  /**
   * @param a precondición: a > 0
   * @param b precondición: b > 0
   * @param c precondición: c > 0
   * @return t postcondición t = SCALENE si el triángulo de lados a, b, c es escaleno. t = ISOSCELES
   * si el triángulo de lados a, b, c es isósceles. t = EQUILATERAL si el triángulo de lados a, b, c
   * es equilátero.
   */
  TriangleType reportTriangleType(int a, int b, int c) {
    if (a == b || b == c || a == c) {
      if (a == b && b == c) {
        return TriangleType.EQUILATERAL;
      } else {
        return TriangleType.ISOSCELES;
      }
    }
    return TriangleType.SCALENE;
  }

  enum TriangleType {
    SCALENE, ISOSCELES, EQUILATERAL
  }
}
```