Computer Programming Fundamentals

CS 152
Professor: Leah Buechley
TAs: Melody Horn, Noah Garcia, Andrew Geyko, Juan Ormaza
Time: MWF 10:00-10:50am
https://handandmachine.cs.unm.edu/classes/CS152_Fall2021/

ASSIGNMENT 5

Last day to hand in with late days:
Thursday

QUIZ 4 FRIDAY

NO MAKE UP QUIZZES

ASSIGNMENT 6
DUE FRIDAY 11/19

questions?

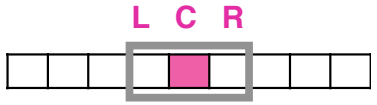
1D CELLULAR AUTOMATA

COMPUTE RULE FOR ONE CELL

OUR RULE

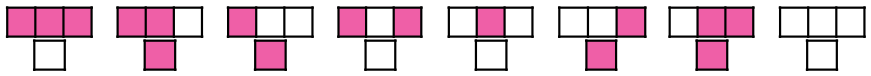
```
//given a neighborhood, return next value of cell
int rule (int left, int center, int right) {

}
```



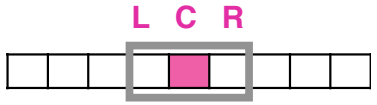
Neighborhood

Rule:



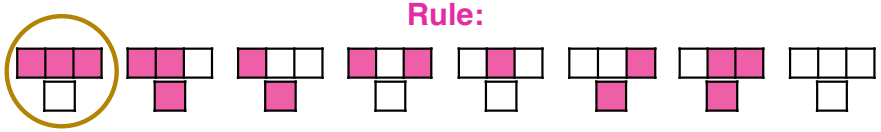
OUR RULE

```
//given a neighborhood, return next value of cell
int rule (int left, int center, int right) {
    if (left == ALIVE && center == ALIVE && right == ALIVE)
        return DEAD;
}
```



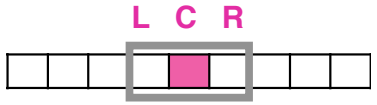
Neighborhood

Rule:



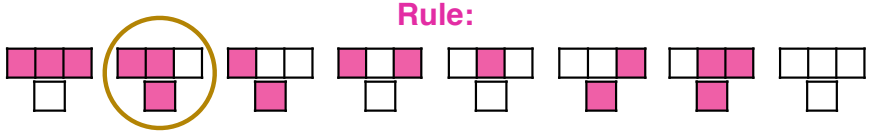
OUR RULE

```
//given a neighborhood, return next value of cell
int rule (int left, int center, int right) {
    if (left == ALIVE && center == ALIVE && right == ALIVE)
        return DEAD;
    if (left == ALIVE && center == ALIVE && right == DEAD)
        return ALIVE;
}
```



Neighborhood

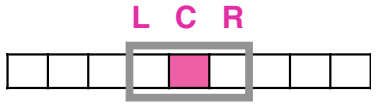
Rule:



why don’t we need an else?
the return statement will get us
out of the method

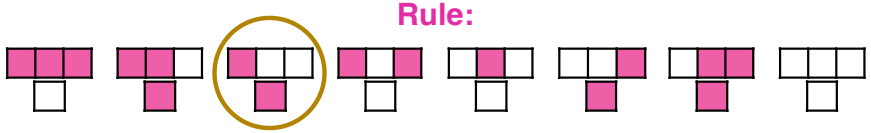
OUR RULE

```
//given a neighborhood, return next value of cell
int rule (int left, int center, int right) {
    if (left == ALIVE && center == ALIVE && right == ALIVE)
        return DEAD;
    if (left == ALIVE && center == ALIVE && right == DEAD)
        return ALIVE;
    if (left == ALIVE && center == DEAD && right == DEAD)
        return ALIVE;
}
```



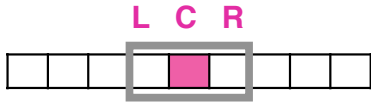
Neighborhood

Rule:

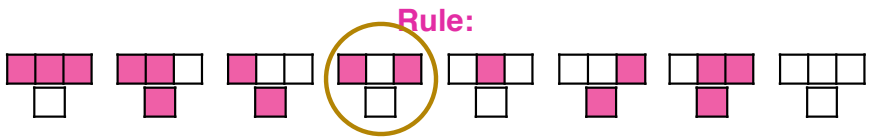


OUR RULE

```
//given a neighborhood, return next value of cell
int rule (int left, int center, int right) {
    if (left == ALIVE && center == ALIVE && right == ALIVE)
        return DEAD;
    if (left == ALIVE && center == ALIVE && right == DEAD)
        return ALIVE;
    if (left == ALIVE && center == DEAD && right == DEAD)
        return ALIVE;
    if (left == ALIVE && center == DEAD && right == ALIVE)
        return DEAD;
}
```

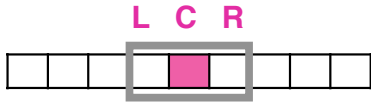


Neighborhood



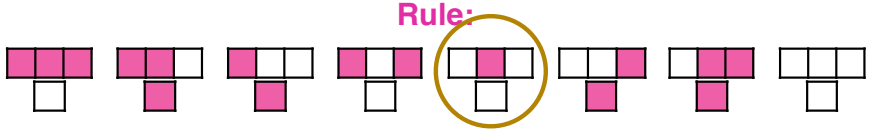
OUR RULE

```
//given a neighborhood, return next value of cell
int rule (int left, int center, int right) {
    if (left == ALIVE && center == ALIVE && right == ALIVE)
        return DEAD;
    if (left == ALIVE && center == ALIVE && right == DEAD)
        return ALIVE;
    if (left == ALIVE && center == DEAD && right == DEAD)
        return ALIVE;
    if (left == ALIVE && center == DEAD && right == ALIVE)
        return DEAD;
    if (left == DEAD && center == ALIVE && right == DEAD)
        return DEAD;
}
```



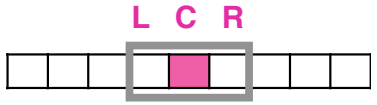
Neighborhood

Rule:



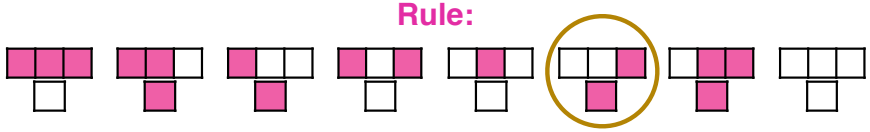
OUR RULE

```
//given a neighborhood, return next value of cell
int rule (int left, int center, int right) {
    if (left == ALIVE && center == ALIVE && right == ALIVE)
        return DEAD;
    if (left == ALIVE && center == ALIVE && right == DEAD)
        return ALIVE;
    if (left == ALIVE && center == DEAD && right == DEAD)
        return ALIVE;
    if (left == ALIVE && center == DEAD && right == ALIVE)
        return DEAD;
    if (left == DEAD && center == ALIVE && right == DEAD)
        return DEAD;
    if (left == DEAD && center == DEAD && right == ALIVE)
        return ALIVE;
}
```



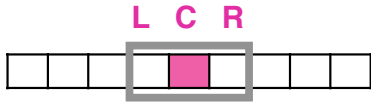
Neighborhood

Rule:



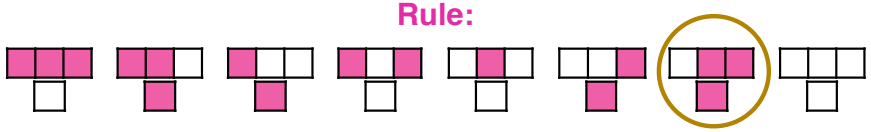
OUR RULE

```
//given a neighborhood, return next value of cell
int rule (int left, int center, int right) {
    if (left == ALIVE && center == ALIVE && right == ALIVE)
        return DEAD;
    if (left == ALIVE && center == ALIVE && right == DEAD)
        return ALIVE;
    if (left == ALIVE && center == DEAD && right == DEAD)
        return ALIVE;
    if (left == ALIVE && center == DEAD && right == ALIVE)
        return DEAD;
    if (left == DEAD && center == ALIVE && right == DEAD)
        return DEAD;
    if (left == DEAD && center == DEAD && right == ALIVE)
        return ALIVE;
    if (left == DEAD && center == ALIVE && right == ALIVE)
        return ALIVE;
}
```



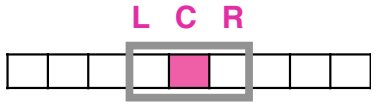
Neighborhood

Rule:



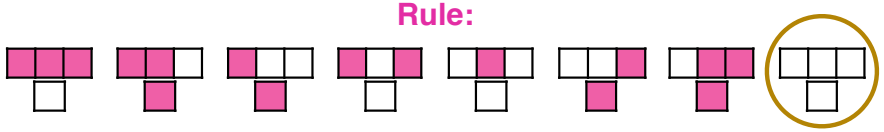
OUR RULE

```
//given a neighborhood, return next value of cell
int rule (int left, int center, int right) {
    if (left == ALIVE && center == ALIVE && right == ALIVE)
        return DEAD;
    if (left == ALIVE && center == ALIVE && right == DEAD)
        return ALIVE;
    if (left == ALIVE && center == DEAD && right == DEAD)
        return ALIVE;
    if (left == ALIVE && center == DEAD && right == ALIVE)
        return DEAD;
    if (left == DEAD && center == ALIVE && right == DEAD)
        return DEAD;
    if (left == DEAD && center == DEAD && right == ALIVE)
        return ALIVE;
    if (left == DEAD && center == ALIVE && right == ALIVE)
        return ALIVE;
    if (left == DEAD && center == DEAD && right == DEAD)
        return DEAD;
}
```



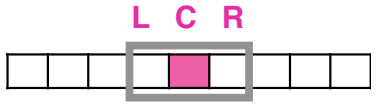
Neighborhood

Rule:



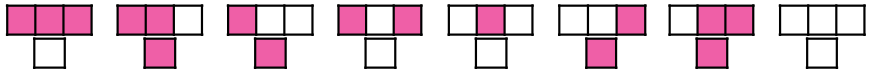
OUR RULE

```
//given a neighborhood, return next value of cell
int rule (int left, int center, int right) {
    if (left == ALIVE && center == ALIVE && right == ALIVE)
        return DEAD;
    if (left == ALIVE && center == ALIVE && right == DEAD)
        return ALIVE;
    if (left == ALIVE && center == DEAD && right == DEAD)
        return ALIVE;
    if (left == ALIVE && center == DEAD && right == ALIVE)
        return DEAD;
    if (left == DEAD && center == ALIVE && right == DEAD)
        return DEAD;
    if (left == DEAD && center == DEAD && right == ALIVE)
        return ALIVE;
    if (left == DEAD && center == ALIVE && right == ALIVE)
        return ALIVE;
    if (left == DEAD && center == DEAD && right == DEAD)
        return DEAD;
    return -1;
}
```



Neighborhood

Rule:



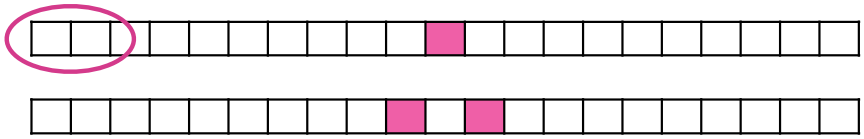
questions?

INEFFICIENT BUT CLEAR CODE

COMPUTE ONE ITERATION OF ENTIRE CA

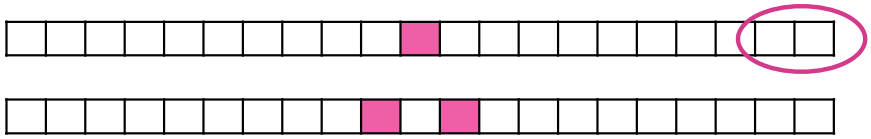
ONE ITERATION

```
int iterate() {
    displayCurrentStates();
    //compute next state for each element in currentStates
    for (int i=1;i<size-1;i++) {
        nextStates[i] = rule(currentStates[i-1], currentStates[i], currentStates[i+1]);
    }
}
```



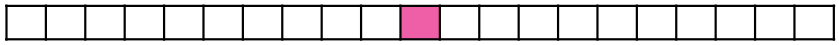
ONE ITERATION

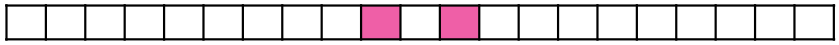
```
int iterate() {
    displayCurrentStates();
    //compute next state for each element in currentStates
    for (int i=1;i<size-1;i++) {
        nextStates[i] = rule(currentStates[i-1], currentStates[i], currentStates[i+1]);
    }
}
```



ONE ITERATION

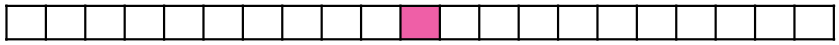
```
int iterate() {
    displayCurrentStates();
    //compute next state for each element in currentStates
    for (int i=1;i<size-1;i++) {
        nextStates[i] = rule(currentStates[i-1], currentStates[i], currentStates[i+1]);
    }
    //set currentStates to be nextStates
    currentStates = nextStates;
}
```

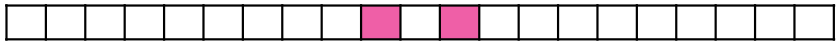




A PROBLEM

```
int iterate() {
    displayCurrentStates();
    //compute next state for each element in currentStates
    for (int i=1;i<size-1;i++) {
        nextStates[i] = rule(currentStates[i-1], currentStates[i], currentStates[i+1]);
    }
    //set currentStates to be nextStates
    currentStates = nextStates;
}
```



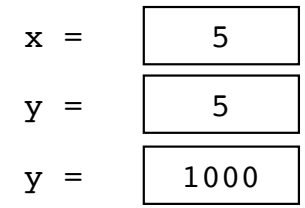


WHY?

PRIMITIVE TYPE VARIABLES

```
int x;
x = 5;
int y = x;
y = 1000;
```

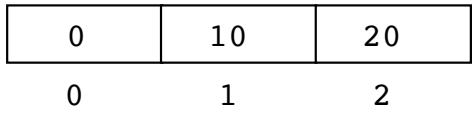
in the computer's memory somewhere



ARRAYS VS PRIMITIVE TYPES

```
int[] myArray;
myArray = new int[3];
for (int i = 0; i < myArray.length; i++)
    myArray[i] = i * 10;
```

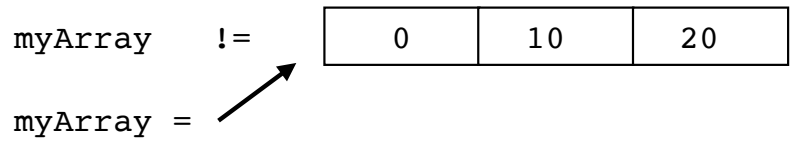
in the computer's memory somewhere



ARRAYS and REFERENCES

```
int[] myArray;
myArray = new int[3];
for (int i = 0; i < myArray.length; i++)
    myArray[i] = i * 10;
```

in the computer's
memory somewhere

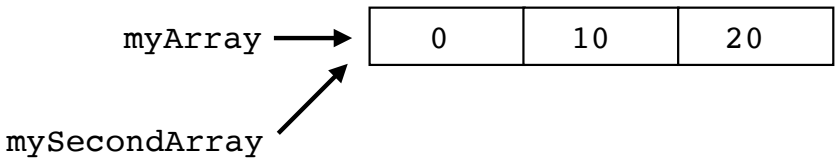


when you create an array variable, the variable stores a “reference” to the array, not the actual array

ARRAYS and REFERENCES

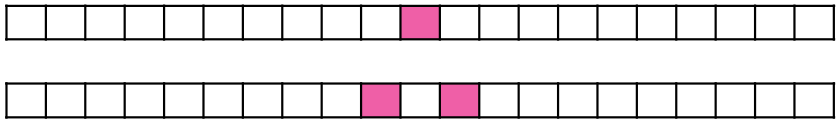
```
int[] myArray;
int[] mySecondArray;
myArray = new int[3];
for (int i = 0; i < myArray.length; i++)
    myArray[i] = i * 10;
mySecondArray = myArray;
```

in the computer's memory somewhere



BACK TO OUR PROBLEM

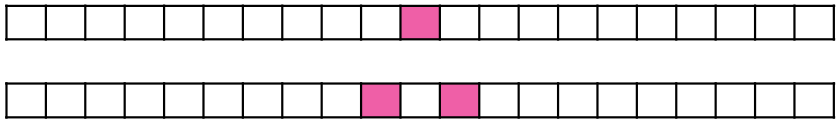
```
int iterate() {
    displayCurrentStates();
    //compute next state for each element in currentStates
    for (int i=1;i<size-1;i++) {
        nextStates[i] = rule(currentStates[i-1], currentStates[i], currentStates[i+1]);
    }
    //set currentStates to be nextStates
    currentStates = nextStates;
}
```



we want a true copy,
not a reference

BACK TO OUR PROBLEM

```
int iterate() {
    displayCurrentStates();
    //compute next state for each element in currentStates
    for (int i=1;i<size-1;i++) {
        nextStates[i] = rule(currentStates[i-1], currentStates[i], currentStates[i+1]);
    }
    //set currentStates to be nextStates
    currentStates = nextStates;
}
```



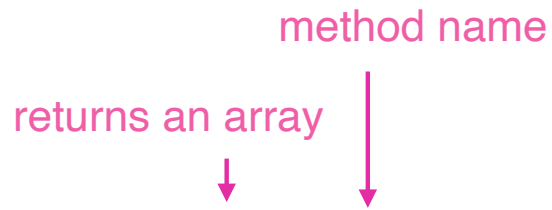
if nextStates and currentStates are referencing the same array, this computation won't work

we'll be changing the array during our rule computation

questions?

SOLUTION: MAKE A TRUE COPY

JAVA clone METHOD FOR ARRAYS




```
protected Array clone()
```

Creates and returns a copy of this object.

Parameters:
`none`

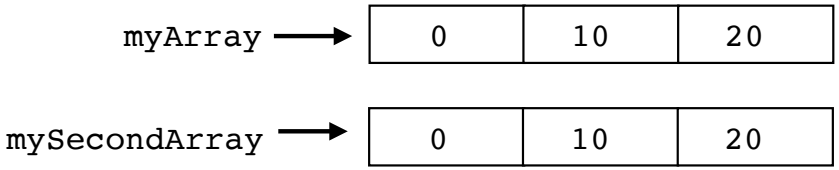
Returns:
The resulting `Array`

https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/lang/Object.html#clone()

CLONE

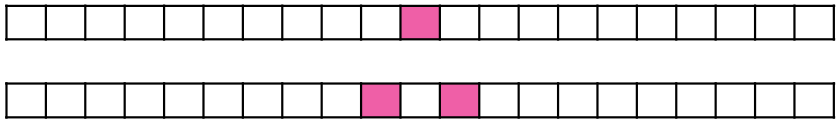
```
int[] myArray;
int[] mySecondArray;
myArray = new int[3];
for (int i = 0; i < myArray.length; i++)
    myArray[i] = i * 10;
mySecondArray = myArray.clone();
```

in the computer's memory somewhere



SOLUTION TO OUR PROBLEM

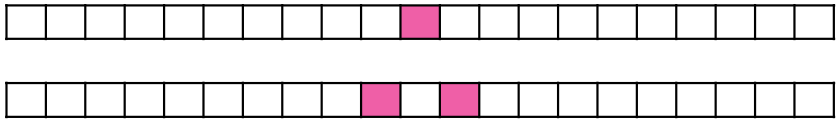
```
int iterate() {
    displayCurrentStates();
    //compute next state for each element in currentStates
    for (int i=1;i<size-1;i++) {
        nextStates[i] = rule(currentStates[i-1], currentStates[i], currentStates[i+1]);
    }
    //set currentStates to be nextStates
    currentStates = nextStates.clone();
}
```



questions?

MORE ITERATIONS

```
int iterate(int iterations) {
    displayCurrentStates();
    for (int j=0;j<iterations;j++) {
        //compute next state for each element in currentStates
        for (int i = 1; i < size - 1; i++) {
            nextStates[i] = rule(currentStates[i - 1], currentStates[i], currentStates[i + 1]);
        }
        //set currentStates to be nextStates
        currentStates = nextStates.clone();
        displayCurrentStates();
    }
}
```



questions?

PUTING IT ALL TOGETHER

IN main

```
public static void main(String[] args) {
    CellularAutomata1D CA = new CellularAutomata1D();
    CA.iterate(3);
}
```

```
  *
 **
*  *
****
```