Array

• Data structure for representing a table.
• May have any non-negative number \( N \) of entries.
• Indexed by integers in the range: \([0…N-1]\).
• Given an index, we can look up the corresponding table entry.
• Given an index and a value, we can change the corresponding table entry.
Random Menu Program

• Declare and initialize array of weekdays.
• Declare and initialize array of meals.
• Iterate over weekdays:
  – Choose a random index into the meal array.
  – Print out the meal for that day.
package randommenu;
import acm.program.*;
import java.util.Random;
public class RandomMenu extends ConsoleProgram {

    private static final String[] week = {
        "Monday", "Tuesday",
        "Wednesday", "Thursday",
        "Friday", "Saturday", "Sunday"};

    private static final String[] menu = {
        "Spagetti", "Steak", "Salmon", "Burgers",
        "Chicken", "Salad", "Swordfish",
        "Pizza", "Chinese Food", "Eat Out"};

    public void run() {
        Random randomizer = new Random();
        for (int day = 0; day < week.length; day++) {
            int choice = Math.abs(randomizer.nextInt()) % menu.length;
            println(week[day] + ": " + menu[choice]);
        }
    }

    public static void main(String[] args) {
        new RandomMenu().start();
    }
}
Layout of Arrays in Memory

week:

0 → “Monday”
1 → “Tuesday”
2 → “Wednesday”
3 → “Thursday”
4 → “Friday”
5 → “Saturday”
6 → “Sunday”

menu:

0 → “Spagetti”
1 → “Steak”
2 → “Salmon”
3 → “Burgers”
4 → “Chicken”
5 → “Salad”
6 → “Swordfish”
7 → “Pizza”
8 → “Chinese Food”
9 → “Eat Out”
Array Declarations

<Data Type>  <Variable Name> [ ] ;

String week[];

<Data Type> [ ]  <Variable Name>;

String[ ] menu;

Variables are declared, but no space for the array has been allocated.
Array Declarations with Allocation

<Data Type> <Variable Name> [ ] = new <Data Type>[<Length>] ;

String week[] = new String[7];

<Data Type>[ ] <Variable Name> = new <Data Type>[<Length>] ;

String[ ] menu = new String[10];

Variables are declared, space for the array is allocated, but no values have been placed in the array.
Array Declarations with Allocation and Initialization

<Data Type> <Variable Name> [ ] = { <Value>, . . . , <Value>} ;

String week[] = {“Monday”, . . . , “Sunday”};

<Data Type>[ ] <Variable Name> = { <Value>, . . . , <Value}> ;

String[ ] menu = {“Spagetti”, . . . , “Eat Out”};

Variables are declared, space for the array is allocated, and values have been placed in the array.
Declaration:

```
String week[];
```

Declaration & Allocation:

```
String week[] = new String[7];
```

Declaration, Allocation & Initialization:

```
String[ ] menu = {"Spagetti", . . . , "Eat Out"};
```
Declaration

week: ____________

Declaration & Allocation

week: ____________

0
1
2
3
4
5
6

Declaration, Allocation & Initialization

week: ____________

0 → “Monday”
1 → “Tuesday”
2 → “Wednesday”
3 → “Thursday”
4 → “Friday”
5 → “Saturday”
6 → “Sunday”
Contest Program

• Read in the number of students in class.
• Declare and allocate an array of the required length.
• Read in each student’s name and store it in the array.
• Select three random array indexes for 1st, 2nd, and 3rd prize.
package contest;
import acm.program.*;
import java.util.Random;
public class Contest extends ConsoleProgram {

    public void run() {
        print("How many students are in class today? ");
        int attendance = readInt();
        String[] student = new String[attendance];
        for (int i = 0; i < attendance; i++)
            student[i] = readLine("Please enter the " + i + "th student's name: ");
        Random randomizer = new Random();
        int choice = Math.abs(randomizer.nextInt())%attendance;
        println("First prize goes to: " + student[choice]);
        choice = Math.abs(randomizer.nextInt())%attendance;
        println("Second prize goes to: " + student[choice]);
        choice = Math.abs(randomizer.nextInt())%attendance;
        println("Third prize goes to: " + student[choice]);
    }

    public static void main(String[] args) {
        new Contest().start();
    }
}
Array Indexes

String student[] = {"Larry", "Curly", "Moe", "Laurel", "Hardy");

student:

0 → "Larry"
1 → "Curly"
2 → "Moe"
3 → "Laurel"
4 → "Hardy"

student.length = 5

Nothing Here!
Assigning Array Elements

```c
student[2] = "Tom";
```

Before Assignment:

```
student:

0   -> "Larry"
1   -> "Curly"
2   -> "Moe"
3   -> "Laurel"
4   -> "Hardy"
```

After Assignment:

```
student:

0   -> "Larry"
1   -> "Curly"
2   -> "Tom"
3   -> "Laurel"
4   -> "Hardy"
```
Accessing Array Elements

```java
System.out.println(student[2]);
```

Program Output: **Tom**

```java
student:

0  ->  "Larry"
1  ->  "Curly"
2  ->  "Tom"
3  ->  "Laurel"
4  ->  "Hardy"
```
Array Bounds Errors

```java
for (int i = 0; i <= attendance; i++)
{
  myConsole.out.print("Please enter the " + i + "th student's name: ");
  myConsole.out.flush();
  student[i] = myConsole.input.readString();
}
```

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException
  at Contest.main(Contest.java:19)

The `for` loop terminates later than it should, so it attempts to assign a value to `student[attendance]` which is past the end of the array. Any attempt to assign to or access non-existent array element will raise this exception.
Accessing Array Data in Memory

```
\text{Address of myTable[0] in memory}
\text{a}
\text{Address of myTable[i] in memory}
\text{a + i*s}
```
Random Permutation of an Array

• Construct a random permutation of a given array.
• For each element in the result array, randomly choose an (unchosen) element of the given array.
• Use array of booleans to keep track of which elements have already been chosen.
public static char[] randomPermutation(char[] stuff) {
    Random randomInts = new Random();
    int size = stuff.length;
    boolean[] chosen = new boolean[size];
    for (int i=0; i<size; i++) { chosen[i] = false; }
    char[] result = new char[size];
    for (int i=0; i<size; i++) {
        boolean done = false;
        do {
            int choice = randomInts.nextInt(size);
            if (!chosen[choice]) {
                result[i] = stuff[choice];
                chosen[choice] = true;
                done = true;
            }
        } while (!done);
    }
    return result;
}
Tile Puzzle

Repeatedly move a numbered tile onto the blank space in order to restore the original ordered configuration.
Sample Tile Puzzle Initial Configuration
Multi-Dimensional Arrays

A two-dimensional array is implemented as an array of arrays.
package tilepuzzle;
import acm.program.*;
import acm.graphics.*;
import java.awt.event.*;
import java.awt.Color;
import java.awt.Font;
import java.util.Random;

public class TilePuzzle extends GraphicsProgram {

    public static final int PUZZLE_SIZE = 4;
    public static final int NUMBER_OF_TILES = PUZZLE_SIZE*PUZZLE_SIZE;
    public static final int CELL_WIDTH = 128;
    public static final int CELL_HEIGHT = 128;
    public static final Font FONT = new Font("Serif", Font.BOLD, 40);
    private static final int DELAY = 125;
    private static Tile[] tile;
    private static Tile[][] state;
    private static Tile blank;

    // ... Omitted ...
}

Declaring a One Dimensional Array of Tiles

Declaring a Two Dimensional Array of Tiles
Two Dimensional Array Declarations

<Data Type> <Variable Name> [ ] [ ];

Tile state[][];

<Data Type> [ ] [ ] <Variable Name>;

Tile[][] state;

Variables are declared, but no space for the array has been allocated.
Two Dimensional Array
Declarations with Allocation

<Data Type>   <Variable Name> [ ] [ ] = new <Data Type> [<Length>] [<Length>] ;

Tile state[][] = new Tile[4][4];

<Data Type>[ ][ ]   <Variable Name> = new <Data Type>[<Length>] [<Length>] ;

Tile[][] state = new Tile[4][4];

Variables are declared, space for the array is allocated, but no values have been placed in the array.
Two Dimensional Array Declarations with Allocation and Initialization

<Data Type> <Variable Name> [ ] [ ] = { {<Value>,...,<Value>},
  ..., 
  {<Value>,...,<Value>} }

Tile state[][] = {{...},..., {...}}

<Data Type>[ ][ ] <Variable Name> = { {<Value>,...,<Value>},
  ..., 
  {<Value>,...,<Value>} }

Tile[][] state = {{...},..., {...}}

Variables are declared, space for the array is allocated, and values have been placed in the array.
public void run() {
    initialize();
    randomize();
    addMouseListeners();
    while (!solved()) pause(DELAY);
    getDialog().println("You Solved It!");
}

public void mousePressed(MouseEvent e) {
    Tile choice = selectTileAt(e.getX(), e.getY());
    if (valid(choice)) moveTile(choice);
}

After we execute `addMouseListeners`, the `mousePressed` ("callback") method is invoked whenever the user clicks in the program window. If the user clicks on a valid tile, the callback function executes a move that swaps the selected tile with the blank space. Meanwhile ... the `run` method periodically checks whether the puzzle is solved, and displays a message when the solution has been found.
Allocate space for a one dimensional array to hold all the tiles.

```java
private void initialize() {
    GRect background = new GRect((PUZZLE_SIZE+2)*CELL_WIDTH,
                                (PUZZLE_SIZE+2)*CELL_HEIGHT);
    background.setFilled(true);
    background.setColor(Color.GRAY);
    add(background);

    tile = new Tile[NUMBER_OF_TILES];

    for (int i = 0; i < NUMBER_OF_TILES; i++) {
        Tile newTile = new Tile(i);
        tile[i] = newTile;
        add(newTile);
    }

    blank = tile[NUMBER_OF_TILES-1];
}
```

Put the *ith* tile in the *ith* array location. Also add it to the program window.
private void randomize() {
    Random randomizer = new Random();
    state = new Tile[PUZZLE_SIZE][PUZZLE_SIZE];
    for (int r = 0; r < PUZZLE_SIZE; r++)
        for (int c = 0; c < PUZZLE_SIZE; c++)
            {
                int number;
                boolean done = false;
                do
                {
                    number = Math.abs(randomizer.nextInt())%NUMBER_OF_TILES;
                    if (!tile[number].placed()) done = true;
                }
                while (!done);
                tile[number].place(r,c);
                state[r][c] = tile[number];
            }
}

Iterate over all the cells in the two-dimensional state array. For each cell, randomly choose a tile to place in that cell. The chosen tile must not have been previously placed elsewhere.

Allocating, but not initializing the two dimensional state array.

Initializing each element of the two dimensional state array.
private boolean solved() {
    boolean answer = true;
    for (int r = 0; r < PUZZLE_SIZE; r++) {
        for (int c = 0; c < PUZZLE_SIZE; c++) {
            int correctNumber = r*PUZZLE_SIZE + c;
            if (state[r][c].number() != correctNumber) {
                answer = false;
                break;
            }
        }
    }
    if (!answer) break;
    return answer;
}

The puzzle is solved if each cell holds tile number r*PUZZLE_SIZE + c where r is the row and c is the column of the cell. Notice that the for loop uses a break statement to terminate as soon as a violation is found.
Computing Goal Tile Number from Row and Column

\[ t = r \times 4 + c \]
private Tile selectTileAt(int x, int y) {
    GPoint mousePoint = new GPoint(x, y);
    for (int r = 0; r < PUZZLE_SIZE; r++) {
        for (int c = 0; c < PUZZLE_SIZE; c++) {
            Tile selection = state[r][c];
            if (selection.contains(mousePoint))
                return selection;
        }
    }
    return null;
}

To find the tile the user has selected (if any) at mouse coordinates \((x, y)\), iterate over all the cells in the two-dimensional \texttt{state} array. Return a tile that contains the mouse point, if one exists. Otherwise, return \texttt{null}. Notice that we use \texttt{return} to end both \texttt{for}-loops and return immediately from the method, when we find a valid selection.
private boolean valid(Tile choice) {
    if (choice == null) return false;
    int rDiff = Math.abs(choice.row() - blank.row());
    int cDiff = Math.abs(choice.column() - blank.column());
    if (((cDiff==1) && (rDiff==0)) || ((cDiff==0) && (rDiff==1)) )
        return true;
    else return false;
}

A tile choice is valid if the tile is next to the blank. First find the
difference in rows between the tile position and the blank
position. Then find the difference in columns. One difference
must be zero. The other difference must be one. (Or visa versa.)

What is the style flaw in the method shown above?
private void moveTile(Tile choice) {
    int choiceRow = choice.row();
    int choiceColumn = choice.column();
    int blankRow = blank.row();
    int blankColumn = blank.column();
    state[blankRow][blankColumn] = choice;
    state[choiceRow][choiceColumn] = blank;
    choice.place(blankRow,blankColumn);
    blank.place(choiceRow,choiceColumn);
}

Move the selected tile by interchanging the tile’s position and the blank’s position. When a tile is moved, the two dimensional state array is updated. In addition, the Tile class place method is used to record the tile’s new position in the tile itself.
Computing Goal Row and Column from Tile Number

\[ r = \text{Row} \]
\[ c = \text{Column} \]

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
0 & 0 & 1 & 2 & 3 \\
1 & 4 & 5 & 6 & 7 \\
2 & 8 & 9 & 10 & 11 \\
3 & 12 & 13 & 14 \\
\end{array}
\]

\[ r = \frac{t}{4} \]
\[ c = t \mod 4 \]
Checkerboard Pattern

color = ( ((r%2)==0)^((c%2)==0) ) ? Color.red : Color.white

If r is divisible by 2 or c is divisible by 2 (but not both) then the color is red, otherwise the color is white.