2D Arrays

Index	0	1	2	3	4
0	65,340	12,483	138,189	902,960	633,877
1	5,246	424,642	650,380	821,254	866,122
2	89,678	236,781	601,691	329,274	913,534
3	103,902	4,567	733,611	263,010	85,550
4	2,778	658,305	128,788	978,155	620,702
5	45,024	55,058	705,586	89,672	384,605
6	780	47,538	523,784	556,801	617,107
7	32,667	350,890	834,753	638,108	85,188
8	56,083	145,582	775,040	548,322	756,587
9	41,123	543,542	537,738	513,048	418,482

Lab 2

- Due on Sunday
- Please see Blackboard

Processing Two-Dimensional Arrays

See the examples in the text.

1. **Initializing arrays with input values**
2. **Printing arrays**
3. **Summing all elements**
4. **Summing all elements by row**
5. **Summing all elements by column**
6. **Which row has the largest sum**
7. **Finding the index of the largest element**

Initializing arrays with input values

```
java.util.Scanner input = new Scanner(System.in);
System.out.println("Enter " + matrix.length + " rows and " +
  matrix[0].length + " columns: ");
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length; column++) {
    matrix[row][column] = input.nextInt();
  }
}
```

Initializing arrays with random values

```
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length; column++) {
    matrix[row][column] = (int)(Math.random() * 100);
  }
}
```

Printing arrays

```
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length; column++) {
    System.out.print(matrix[row][column] + " ");
  }

  System.out.println();
}
```

Summing all elements

```
int total = 0;
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length; column++) {
    total += matrix[row][column];
  }
}
System.out.println(total);
```

Summing elements by row

```
for (int row = 0; row < matrix.length; row++) {
  int rowSum = 0;
  for (int column = 0; column < matrix[row].length; column++) {
    rowSum += matrix[row][column];
  }
  System.out.println(rowSum);
}
```

Summing elements by column

```
for (int column = 0; column < matrix[0].length; column++) {
  int total = 0;
  for (int row = 0; row < matrix.length; row++)
    total += matrix[row][column];
  System.out.println("Sum for column " + column + " is "
    + total);
}
```

Summing elements by column for a ragged array

```
public static void sumByColumns2(int[][] R){
  int len = lengthOfLongest(R);
  int[] colSums = new int[len];

  for (int i=0; i<R.length;i++){
    for (int j=0; j<R[i].length;j++){
      colSums[j]=colSums[j]+R[i][j];
    }
  }

  for (int c=0; c<colSums.length; c++){
    System.out.println(colSums[c]);
  }

}

public static int lengthOfLongest(int[][] R){
  int longest = R[0].length;
  for (int i=1; i<R.length;i++){
    if (R[i].length>longest)
      longest = R[i].length;
  }
  return longest;
}
```

Summing elements by column for a ragged array (nicer way)

```
public static void sumByColumns(int[][] R){
    for (int c=0; ; c++){
      int colSum = 0;
      int boundaryCrossedCount =0;
      for (int r=0; r<R.length;r++){
        if (c<R[r].length){
          colSum+=R[r][c];
        }else
          boundaryCrossedCount++;
      }
      if(boundaryCrossedCount==R.length)
        break;
      System.out.println(colSum);
    }
}
```

Processing Two-Dimensional Arrays

See the examples in the text.

1. **Initializing arrays with input values**
2. **Printing arrays**
3. **Summing all elements**
4. **Summing all elements by row**
5. **Summing all elements by column**
6. **Which row has the largest sum**
7. **Finding the index of the largest element**

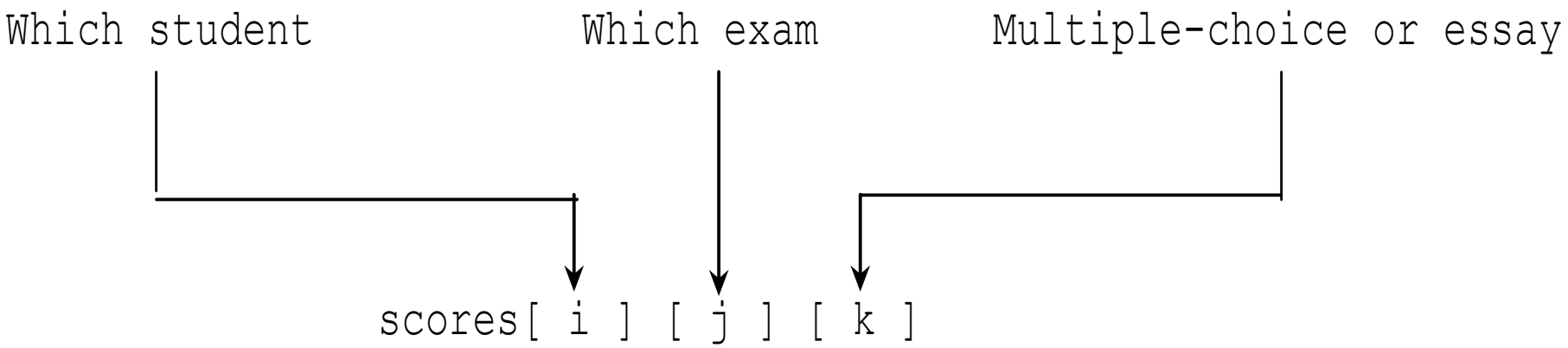
Multidimensional Arrays

Occasionally, you will need to represent n-dimensional data structures. In Java, you can create n-dimensional arrays for any integer n.

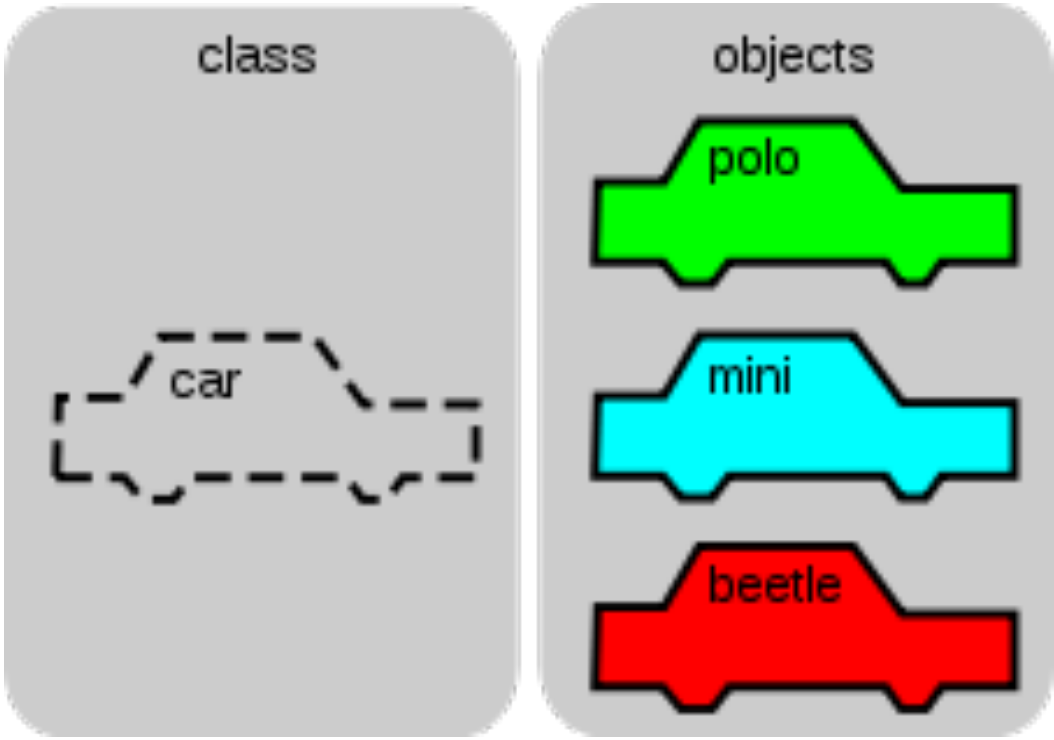
The way to declare two-dimensional array variables and create two-dimensional arrays can be generalized to declare n-dimensional array variables and create n-dimensional arrays for $n >= 3$.

Multidimensional Arrays

```
double[][][] scores = {
  {{7.5, 20.5}, {9.0, 22.5}, {15, 33.5}, {13, 21.5}, {15, 2.5}},
  {{4.5, 21.5}, {9.0, 22.5}, {15, 34.5}, {12, 20.5}, {14, 9.5}},
  {{6.5, 30.5}, {9.4, 10.5}, {11, 33.5}, {11, 23.5}, {10, 2.5}},
  {{6.5, 23.5}, {9.4, 32.5}, {13, 34.5}, {11, 20.5}, {16, 7.5}},
  {{8.5, 26.5}, {9.4, 52.5}, {13, 36.5}, {13, 24.5}, {16, 2.5}},
  {{9.5, 20.5}, {9.4, 42.5}, {13, 31.5}, {12, 20.5}, {16, 6.5}}};
```



Chapter 9: Objects and Classes



Object Oriented Programming Concepts

- **Object-oriented programming (OOP)** involves programming using objects.
- An ***object*** represents an entity in the real world that can be distinctly identified. For example, **a student, a desk, a circle, a button, and even a loan** can all be viewed as objects.
- An object has a unique identity, state, and behaviors. The *state* of an object consists of a set of *data fields* (also known as *properties*) with their current values. The *behavior* of an object is defined by a set of methods.

Thank You