Data Structures and Algorithms in Java

Procedural Programming: Arrays

Outline

An array is an ordered collection of values

One Dimensional (1D) Arrays · Creation

An array is an ordered collection of values

```
<type>[] <name> = {<expression>, <expression>, ...};

<type>[] <name>;
<name> = new <type>[] {<expression>, <expression>, ...};
```

One Dimensional (1D) Arrays · Creation

An array is an ordered collection of values

```
<type>[] <name> = {<expression>, <expression>, ...};

<type>[] <name>;
<name> = new <type>[] {<expression>, <expression>, ...};
```

Example

```
String[] suits = {"Clubs", "Diamonds", "Hearts", "Spades"};

double[] powersOfTwo;
powersOfTwo = new double[] {0.0625, 0.125, 0.25, 0.5, 1.0, 2.0, 4.0, 8.0, 16.0, 32};
```

One Dimensional (1D) Arrays · Creation

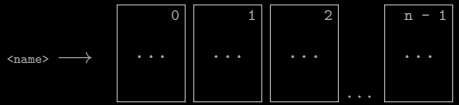
```
<type>[] <name> = new <type>[<size>];

<type>[] <name>;
<name> = new <type>[<size>];
```

One Dimensional (1D) Arrays · Creation

```
<type>[] <name> = new <type>[<size>];

<type>[] <name>;
<name> = new <type>[<size>];
```

Example

```
int[] x = new int[10];

boolean[] y;
y = new boolean[20]
```


<name>
0
1
2
n - 1
...
...
...
...
...

The number of elements (say `n`) in an array `x` is obtained as `x.length`

- The number of elements (say `n`) in an array `x` is obtained as `x.length`
- The `i`th element in an array `x` is accessed as `x[i]`, where `0 <= i < n`

The number of elements (say `n`) in an array `x` is obtained as `x.length`

The `i`th element in an array `x` is accessed as `x[i]`, where `0 <= i < n`

The `i`th element in an array `x` is assigned a value as

```
x[i] = <expression>;
```

One Dimensional (1D) Arrays · Example (Identity Permutation)

```
1  int[] perm;
2
3  perm = new int[5];
4
5  for (int i = 0; i < 5; i++) {
6      perm[i] = i;
7  }
```

Variable Trace

line #	perm	i

One Dimensional (1D) Arrays · Example (Identity Permutation)

```
1 int[] perm;
2
3 perm = new int[5];
4
5 for (int i = 0; i < 5; i++) {
6     perm[i] = i;
7 }
```

Variable Trace

line #	perm	i
1	null	

One Dimensional (1D) Arrays · Example (Identity Permutation)

```
1  int[] perm;
2
3  perm = new int[5];
4
5  for (int i = 0; i < 5; i++) {
6      perm[i] = i;
7  }
```

Variable Trace

line #	perm	i
3	[0, 0, 0, 0, 0]	

One Dimensional (1D) Arrays · Example (Identity Permutation)

```
1 int[] perm;
2
3 perm = new int[5];
4
5 for (int i = 0; i < 5; i++) {
6     perm[i] = i;
7 }
```

Variable Trace

line #	perm	i
5	[0, 0, 0, 0, 0]	0

One Dimensional (1D) Arrays · Example (Identity Permutation)

```
1 int[] perm;
2
3 perm = new int[5];
4
5 for (int i = 0; i < 5; i++) {
6     perm[i] = i;
7 }
```

Variable Trace

line #	perm	i
6	[0, 0, 0, 0, 0]	0

One Dimensional (1D) Arrays · Example (Identity Permutation)

```
1 int[] perm;
2
3 perm = new int[5];
4
5 for (int i = 0; i < 5; i++) {
6     perm[i] = i;
7 }
```

Variable Trace

line #	perm	i
5	[0, 0, 0, 0, 0]	1

One Dimensional (1D) Arrays · Example (Identity Permutation)

```
1 int[] perm;
2
3 perm = new int[5];
4
5 for (int i = 0; i < 5; i++) {
6     perm[i] = i;
7 }
```

Variable Trace

line #	perm	i
6	[0, 1, 0, 0, 0]	1

One Dimensional (1D) Arrays · Example (Identity Permutation)

```
1 int[] perm;
2
3 perm = new int[5];
4
5 for (int i = 0; i < 5; i++) {
6     perm[i] = i;
7 }
```

Variable Trace

line #	perm	i
5	[0, 1, 0, 0, 0]	2

One Dimensional (1D) Arrays · Example (Identity Permutation)

```
1 int[] perm;
2
3 perm = new int[5];
4
5 for (int i = 0; i < 5; i++) {
6     perm[i] = i;
7 }
```

Variable Trace

line #	perm	i
6	[0, 1, 2, 0, 0]	2

One Dimensional (1D) Arrays · Example (Identity Permutation)

```
1 int[] perm;
2
3 perm = new int[5];
4
5 for (int i = 0; i < 5; i++) {
6     perm[i] = i;
7 }
```

Variable Trace

line #	perm	i
5	[0, 1, 2, 0, 0]	3

One Dimensional (1D) Arrays · Example (Identity Permutation)

```
1 int[] perm;
2
3 perm = new int[5];
4
5 for (int i = 0; i < 5; i++) {
6     perm[i] = i;
7 }
```

Variable Trace

line #	perm	i
6	[0, 1, 2, 3, 0]	3

One Dimensional (1D) Arrays · Example (Identity Permutation)

```
1  int[] perm;
2
3  perm = new int[5];
4
5  for (int i = 0; i < 5; i++) {
6      perm[i] = i;
7  }
```

Variable Trace

line #	perm	i
5	[0, 1, 2, 3, 0]	4

One Dimensional (1D) Arrays · Example (Identity Permutation)

```
1  int[] perm;
2
3  perm = new int[5];
4
5  for (int i = 0; i < 5; i++) {
6      perm[i] = i;
7  }
```

Variable Trace

line #	perm	i
6	[0, 1, 2, 3, 4]	4

One Dimensional (1D) Arrays · Example (Identity Permutation)

```
1  int[] perm;
2
3  perm = new int[5];
4
5  for (int i = 0; i < 5; i++) {
6      perm[i] = i;
7  }
```

Variable Trace

line #	perm	i
5	[0, 1, 2, 3, 4]	

One Dimensional (1D) Arrays · Example (Identity Permutation)

```
1  int[] perm;
2
3  perm = new int[5];
4
5  for (int i = 0; i < 5; i++) {
6      perm[i] = i;
7  }
```

Variable Trace

line #	perm	i

One Dimensional (1D) Arrays · Example (In-place Reversal)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5 / 2; i++) {
4     int temp = perm[i];
5     perm[i] = perm[5 - i - 1];
6     perm[5 - i - 1] = temp;
7 }
```

Variable Trace

line #	perm	i	temp

One Dimensional (1D) Arrays · Example (In-place Reversal)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5 / 2; i++) {
4     int temp = perm[i];
5     perm[i] = perm[5 - i - 1];
6     perm[5 - i - 1] = temp;
7 }
```

Variable Trace

line #	perm	i	temp
1	[0, 1, 2, 3, 4]		

One Dimensional (1D) Arrays · Example (In-place Reversal)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5 / 2; i++) {
4     int temp = perm[i];
5     perm[i] = perm[5 - i - 1];
6     perm[5 - i - 1] = temp;
7 }
```

Variable Trace

line #	perm	i	temp
3	[0, 1, 2, 3, 4]	0	

One Dimensional (1D) Arrays · Example (In-place Reversal)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5 / 2; i++) {
4     int temp = perm[i];
5     perm[i] = perm[5 - i - 1];
6     perm[5 - i - 1] = temp;
7 }
```

Variable Trace

line #	perm	i	temp
4	[0, 1, 2, 3, 4]	0	0

One Dimensional (1D) Arrays · Example (In-place Reversal)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5 / 2; i++) {
4     int temp = perm[i];
5     perm[i] = perm[5 - i - 1];
6     perm[5 - i - 1] = temp;
7 }
```

Variable Trace

line #	perm	i	temp
5	[4, 1, 2, 3, 4]	0	0

One Dimensional (1D) Arrays · Example (In-place Reversal)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5 / 2; i++) {
4     int temp = perm[i];
5     perm[i] = perm[5 - i - 1];
6     perm[5 - i - 1] = temp;
7 }
```

Variable Trace

line #	perm	i	temp
6	[4, 1, 2, 3, 0]	0	0

One Dimensional (1D) Arrays · Example (In-place Reversal)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5 / 2; i++) {
4     int temp = perm[i];
5     perm[i] = perm[5 - i - 1];
6     perm[5 - i - 1] = temp;
7 }
```

Variable Trace

line #	perm	i	temp
3	[4, 1, 2, 3, 0]	1	

One Dimensional (1D) Arrays · Example (In-place Reversal)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5 / 2; i++) {
4     int temp = perm[i];
5     perm[i] = perm[5 - i - 1];
6     perm[5 - i - 1] = temp;
7 }
```

Variable Trace

line #	perm	i	temp
4	[4, 1, 2, 3, 0]	1	1

One Dimensional (1D) Arrays · Example (In-place Reversal)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5 / 2; i++) {
4     int temp = perm[i];
5     perm[i] = perm[5 - i - 1];
6     perm[5 - i - 1] = temp;
7 }
```

Variable Trace

line #	perm	i	temp
5	[4, 3, 2, 3, 0]	1	1

One Dimensional (1D) Arrays · Example (In-place Reversal)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5 / 2; i++) {
4     int temp = perm[i];
5     perm[i] = perm[5 - i - 1];
6     perm[5 - i - 1] = temp;
7 }
```

Variable Trace

line #	perm	i	temp
6	[4, 3, 2, 1, 0]	1	1

One Dimensional (1D) Arrays · Example (In-place Reversal)

```java
int[] perm = {0, 1, 2, 3, 4};

for (int i = 0; i < 5 / 2; i++) {
    int temp = perm[i];
    perm[i] = perm[5 - i - 1];
    perm[5 - i - 1] = temp;
}
```

Variable Trace

line #	perm	i	temp
3	[4, 3, 2, 1, 0]	2	

One Dimensional (1D) Arrays · Example (In-place Reversal)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5 / 2; i++) {
4     int temp = perm[i];
5     perm[i] = perm[5 - i - 1];
6     perm[5 - i - 1] = temp;
7 }
```

Variable Trace

line #	perm	i	temp

One Dimensional (1D) Arrays · Example (Knuth Shuffle)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5; i++) {
4     int r = StdRandom.uniform(i, 5);
5     int temp = perm[r];
6     perm[r] = perm[i];
7     perm[i] = temp;
8 }
```

Variable Trace

line #	perm	i	r	temp

One Dimensional (1D) Arrays · Example (Knuth Shuffle)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5; i++) {
4     int r = StdRandom.uniform(i, 5);
5     int temp = perm[r];
6     perm[r] = perm[i];
7     perm[i] = temp;
8 }
```

Variable Trace

line #	perm	i	r	temp
1	[0, 1, 2, 3, 4]			

One Dimensional (1D) Arrays · Example (Knuth Shuffle)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5; i++) {
4     int r = StdRandom.uniform(i, 5);
5     int temp = perm[r];
6     perm[r] = perm[i];
7     perm[i] = temp;
8 }
```

Variable Trace

line #	perm	i	r	temp
3	[0, 1, 2, 3, 4]	0		

One Dimensional (1D) Arrays · Example (Knuth Shuffle)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5; i++) {
4     int r = StdRandom.uniform(i, 5);
5     int temp = perm[r];
6     perm[r] = perm[i];
7     perm[i] = temp;
8 }
```

Variable Trace

line #	perm	i	r	temp
4	[0, 1, 2, 3, 4]	0	3	

One Dimensional (1D) Arrays · Example (Knuth Shuffle)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5; i++) {
4     int r = StdRandom.uniform(i, 5);
5     int temp = perm[r];
6     perm[r] = perm[i];
7     perm[i] = temp;
8 }
```

Variable Trace

line #	perm	i	r	temp
5	[0, 1, 2, 3, 4]	0	3	3

One Dimensional (1D) Arrays · Example (Knuth Shuffle)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5; i++) {
4     int r = StdRandom.uniform(i, 5);
5     int temp = perm[r];
6     perm[r] = perm[i];
7     perm[i] = temp;
8 }
```

Variable Trace

line #	perm	i	r	temp
6	[0, 1, 2, 0, 4]	0	3	3

One Dimensional (1D) Arrays · Example (Knuth Shuffle)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5; i++) {
4     int r = StdRandom.uniform(i, 5);
5     int temp = perm[r];
6     perm[r] = perm[i];
7     perm[i] = temp;
8 }
```

Variable Trace

line #	perm	i	r	temp
7	[3, 1, 2, 0, 4]	0	3	3

One Dimensional (1D) Arrays · Example (Knuth Shuffle)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5; i++) {
4     int r = StdRandom.uniform(i, 5);
5     int temp = perm[r];
6     perm[r] = perm[i];
7     perm[i] = temp;
8 }
```

Variable Trace

line #	perm	i	r	temp
3	[3, 1, 2, 0, 4]	1		

One Dimensional (1D) Arrays · Example (Knuth Shuffle)

```
1  int[] perm = {0, 1, 2, 3, 4};
2
3  for (int i = 0; i < 5; i++) {
4      int r = StdRandom.uniform(i, 5);
5      int temp = perm[r];
6      perm[r] = perm[i];
7      perm[i] = temp;
8  }
```

Variable Trace

line #	perm	i	r	temp
4	[3, 1, 2, 0, 4]	1	2	

One Dimensional (1D) Arrays · Example (Knuth Shuffle)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5; i++) {
4     int r = StdRandom.uniform(i, 5);
5     int temp = perm[r];
6     perm[r] = perm[i];
7     perm[i] = temp;
8 }
```

Variable Trace

line #	perm	i	r	temp
5	[3, 1, 2, 0, 4]	1	2	2

One Dimensional (1D) Arrays · Example (Knuth Shuffle)

```
1  int[] perm = {0, 1, 2, 3, 4};
2
3  for (int i = 0; i < 5; i++) {
4      int r = StdRandom.uniform(i, 5);
5      int temp = perm[r];
6      perm[r] = perm[i];
7      perm[i] = temp;
8  }
```

Variable Trace

line #	perm	i	r	temp
6	[3, 1, 1, 0, 4]	1	2	2

One Dimensional (1D) Arrays · Example (Knuth Shuffle)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5; i++) {
4     int r = StdRandom.uniform(i, 5);
5     int temp = perm[r];
6     perm[r] = perm[i];
7     perm[i] = temp;
8 }
```

Variable Trace

line #	perm	i	r	temp
7	[3, 2, 1, 0, 4]	1	2	2

One Dimensional (1D) Arrays · Example (Knuth Shuffle)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5; i++) {
4     int r = StdRandom.uniform(i, 5);
5     int temp = perm[r];
6     perm[r] = perm[i];
7     perm[i] = temp;
8 }
```

Variable Trace

line #	perm	i	r	temp
3	[3, 2, 1, 0, 4]	2		

One Dimensional (1D) Arrays · Example (Knuth Shuffle)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5; i++) {
4     int r = StdRandom.uniform(i, 5);
5     int temp = perm[r];
6     perm[r] = perm[i];
7     perm[i] = temp;
8 }
```

Variable Trace

line #	perm	i	r	temp
4	[3, 2, 1, 0, 4]	2	2	

One Dimensional (1D) Arrays · Example (Knuth Shuffle)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5; i++) {
4     int r = StdRandom.uniform(i, 5);
5     int temp = perm[r];
6     perm[r] = perm[i];
7     perm[i] = temp;
8 }
```

Variable Trace

line #	perm	i	r	temp
5	[3, 2, 1, 0, 4]	2	2	1

One Dimensional (1D) Arrays · Example (Knuth Shuffle)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5; i++) {
4     int r = StdRandom.uniform(i, 5);
5     int temp = perm[r];
6     perm[r] = perm[i];
7     perm[i] = temp;
8 }
```

Variable Trace

line #	perm	i	r	temp
6	[3, 2, 1, 0, 4]	2	2	1

One Dimensional (1D) Arrays · Example (Knuth Shuffle)

```
1  int[] perm = {0, 1, 2, 3, 4};
2
3  for (int i = 0; i < 5; i++) {
4      int r = StdRandom.uniform(i, 5);
5      int temp = perm[r];
6      perm[r] = perm[i];
7      perm[i] = temp;
8  }
```

Variable Trace

line #	perm	i	r	temp
7	[3, 2, 1, 0, 4]	2	2	1

One Dimensional (1D) Arrays · Example (Knuth Shuffle)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5; i++) {
4     int r = StdRandom.uniform(i, 5);
5     int temp = perm[r];
6     perm[r] = perm[i];
7     perm[i] = temp;
8 }
```

Variable Trace

line #	perm	i	r	temp
3	[3, 2, 1, 0, 4]	3		

One Dimensional (1D) Arrays · Example (Knuth Shuffle)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5; i++) {
4     int r = StdRandom.uniform(i, 5);
5     int temp = perm[r];
6     perm[r] = perm[i];
7     perm[i] = temp;
8 }
```

Variable Trace

line #	perm	i	r	temp
4	[3, 2, 1, 0, 4]	3	4	

One Dimensional (1D) Arrays · Example (Knuth Shuffle)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5; i++) {
4     int r = StdRandom.uniform(i, 5);
5     int temp = perm[r];
6     perm[r] = perm[i];
7     perm[i] = temp;
8 }
```

Variable Trace

line #	perm	i	r	temp
5	[3, 2, 1, 0, 4]	3	4	4

One Dimensional (1D) Arrays · Example (Knuth Shuffle)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5; i++) {
4     int r = StdRandom.uniform(i, 5);
5     int temp = perm[r];
6     perm[r] = perm[i];
7     perm[i] = temp;
8 }
```

Variable Trace

line #	perm	i	r	temp
6	[3, 2, 1, 0, 0]	3	4	4

One Dimensional (1D) Arrays · Example (Knuth Shuffle)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5; i++) {
4     int r = StdRandom.uniform(i, 5);
5     int temp = perm[r];
6     perm[r] = perm[i];
7     perm[i] = temp;
8 }
```

Variable Trace

line #	perm	i	r	temp
7	[3, 2, 1, 4, 0]	3	4	4

One Dimensional (1D) Arrays · Example (Knuth Shuffle)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5; i++) {
4     int r = StdRandom.uniform(i, 5);
5     int temp = perm[r];
6     perm[r] = perm[i];
7     perm[i] = temp;
8 }
```

Variable Trace

line #	perm	i	r	temp
3	[3, 2, 1, 4, 0]	4		

One Dimensional (1D) Arrays · Example (Knuth Shuffle)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5; i++) {
4     int r = StdRandom.uniform(i, 5);
5     int temp = perm[r];
6     perm[r] = perm[i];
7     perm[i] = temp;
8 }
```

Variable Trace

line #	perm	i	r	temp
4	[3, 2, 1, 4, 0]	4	4	

One Dimensional (1D) Arrays · Example (Knuth Shuffle)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5; i++) {
4     int r = StdRandom.uniform(i, 5);
5     int temp = perm[r];
6     perm[r] = perm[i];
7     perm[i] = temp;
8 }
```

Variable Trace

line #	perm	i	r	temp
5	[3, 2, 1, 4, 0]	4	4	4

One Dimensional (1D) Arrays · Example (Knuth Shuffle)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5; i++) {
4     int r = StdRandom.uniform(i, 5);
5     int temp = perm[r];
6     perm[r] = perm[i];
7     perm[i] = temp;
8 }
```

Variable Trace

line #	perm	i	r	temp
6	[3, 2, 1, 4, 0]	4	4	4

One Dimensional (1D) Arrays · Example (Knuth Shuffle)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5; i++) {
4     int r = StdRandom.uniform(i, 5);
5     int temp = perm[r];
6     perm[r] = perm[i];
7     perm[i] = temp;
8 }
```

Variable Trace

line #	perm	i	r	temp
7	[3, 2, 1, 4, 0]	4	4	4

One Dimensional (1D) Arrays · Example (Knuth Shuffle)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5; i++) {
4     int r = StdRandom.uniform(i, 5);
5     int temp = perm[r];
6     perm[r] = perm[i];
7     perm[i] = temp;
8 }
```

Variable Trace

line #	perm	i	r	temp
3	[3, 2, 1, 4, 0]	5		

One Dimensional (1D) Arrays · Example (Knuth Shuffle)

```
1 int[] perm = {0, 1, 2, 3, 4};
2
3 for (int i = 0; i < 5; i++) {
4     int r = StdRandom.uniform(i, 5);
5     int temp = perm[r];
6     perm[r] = perm[i];
7     perm[i] = temp;
8 }
```

Variable Trace

line #	perm	i	r	temp

Sample.java

Command-line input	m (int) and n (int)
Standard output	a random sample (without replacement) of m integers, each from the interval $[0, n)$

Sample.java

Command-line input	m (int) and n (int)
Standard output	a random sample (without replacement) of m integers, each from the interval $[0, n)$

```
>_ ~/workspace/dsaj/programs

$ _
```

Sample.java

Command-line input	m (int) and n (int)
Standard output	a random sample (without replacement) of m integers, each from the interval $[0, n)$

```
>_ ~/workspace/dsaj/programs
$ java Sample 6 16
```

Sample.java

Command-line input	m (int) and n (int)
Standard output	a random sample (without replacement) of m integers, each from the interval $[0, n)$

>_ ~/workspace/dsaj/programs

```
$ java Sample 6 16
11 10 12 13 6 8
$ _
```

Sample.java

Command-line input	m (int) and n (int)
Standard output	a random sample (without replacement) of m integers, each from the interval $[0, n)$

~/workspace/dsaj/programs

```
$ java Sample 6 16
11 10 12 13 6 8
$ java Sample 10 1000
```

Sample.java

Command-line input	m (int) and n (int)
Standard output	a random sample (without replacement) of m integers, each from the interval $[0, n)$

>_ ~/workspace/dsaj/programs

```
$ java Sample 6 16
11 10 12 13 6 8
$ java Sample 10 1000
21 432 270 287 166 484 437 675 78 213
$ _
```

Sample.java

Command-line input	m (int) and n (int)
Standard output	a random sample (without replacement) of m integers, each from the interval $[0, n)$

>_ ~/workspace/dsaj/programs

```
$ java Sample 6 16
11 10 12 13 6 8
$ java Sample 10 1000
21 432 270 287 166 484 437 675 78 213
$ java Sample 20 20
```

Sample.java

Command-line input	m (int) and n (int)
Standard output	a random sample (without replacement) of m integers, each from the interval $[0, n)$

>_ ~/workspace/dsaj/programs

```
$ java Sample 6 16
11 10 12 13 6 8
$ java Sample 10 1000
21 432 270 287 166 484 437 675 78 213
$ java Sample 20 20
9 0 15 13 4 8 11 17 3 18 16 5 7 19 14 12 2 1 10 6
$ _
```

i	0	1	2	3	4	5	6	7	8	9
perm[i]	0	0	0	0	0	0	0	0	0	0

m = 5, n = 10

i	0	1	2	3	4	5	6	7	8	9
perm[i]	0	1	2	3	4	5	6	7	8	9

m = 5, n = 10

i	0	1	2	3	4	5	6	7	8	9
perm[i]	3	7	2	9	1	5	6	4	8	0

m = 5, n = 10

Sample.java

```java
import stdlib.StdOut;
import stdlib.StdRandom;

public class Sample {
    public static void main(String[] args) {
        int m = Integer.parseInt(args[0]);
        int n = Integer.parseInt(args[1]);
        int[] perm = new int[n];
        for (int i = 0; i < n; i++) {
            perm[i] = i;
        }
        for (int i = 0; i < m; i++) {
            int r = StdRandom.uniform(i, n);
            int temp = perm[r];
            perm[r] = perm[i];
            perm[i] = temp;
        }
        for (int i = 0; i < m; i++) {
            StdOut.print(perm[i] + " ");
        }
        StdOut.println();
    }
}
```

`CouponCollector.java`

Command-line input	n (int)
Standard output	number of coupons one must collect before obtaining at least one of the n unique coupons

CouponCollector.java

Command-line input	n (int)
Standard output	number of coupons one must collect before obtaining at least one of the n unique coupons

```
>_ ~/workspace/dsaj/programs

$ _
```

CouponCollector.java

Command-line input	n (int)
Standard output	number of coupons one must collect before obtaining at least one of the n unique coupons

>_ ~/workspace/dsaj/programs

```
$ java CouponCollector 1000
```

CouponCollector.java

Command-line input	n (int)
Standard output	number of coupons one must collect before obtaining at least one of the n unique coupons

```
>_ ~/workspace/dsaj/programs

$ java CouponCollector 1000
8317
$ _
```

CouponCollector.java

Command-line input	n (int)
Standard output	number of coupons one must collect before obtaining at least one of the n unique coupons

```
>_ ~/workspace/dsaj/programs

$ java CouponCollector 1000
8317
$ java CouponCollector 1000
```

CouponCollector.java

Command-line input	n (int)
Standard output	number of coupons one must collect before obtaining at least one of the n unique coupons

```
>_ ~/workspace/dsaj/programs

$ java CouponCollector 1000
8317
$ java CouponCollector 1000
7867
$ _
```

CouponCollector.java

Command-line input	n (int)
Standard output	number of coupons one must collect before obtaining at least one of the n unique coupons

~/workspace/dsaj/programs

```
$ java CouponCollector 1000
8317
$ java CouponCollector 1000
7867
$ java CouponCollector 1000000
```

CouponCollector.java

Command-line input	n (int)
Standard output	number of coupons one must collect before obtaining at least one of the n unique coupons

```
>_ ~/workspace/dsaj/programs

$ java CouponCollector 1000
8317
$ java CouponCollector 1000
7867
$ java CouponCollector 1000000
15942756
$ _
```

value	count	collectedCount
	0	0

value	0	1	2
isCollected[value]	F	F	F

value	count	collectedCount
1	1	1

value	0	1	2
isCollected[value]	F	T	F

value	count	collectedCount
1	2	1

value	0	1	2
isCollected[value]	F	T	F

value	count	collectedCount
1	3	1

value	0	1	2
isCollected[value]	F	T	F

value	count	collectedCount
2	4	2

value	0	1	2
isCollected[value]	F	T	T

value	count	collectedCount
0	5	3

value	0	1	2
isCollected[value]	T	T	T

One Dimensional (1D) Arrays · Example (Coupon Collector Problem)

</> CouponCollector.java

```java
import stdlib.StdOut;
import stdlib.StdRandom;

public class CouponCollector {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        int count = 0;
        int collectedCount = 0;
        boolean[] isCollected = new boolean[n];
        while (collectedCount < n) {
            int value = StdRandom.uniform(0, n);
            count++;
            if (!isCollected[value]) {
                collectedCount++;
                isCollected[value] = true;
            }
        }
        StdOut.println(count);
    }
}
```

PrimeSieve.java

Command-line input	n (int)
Standard output	number of primes that are less than or equal to n

PrimeSieve.java

Command-line input	n (int)
Standard output	number of primes that are less than or equal to n

```
>_ ~/workspace/dsaj/programs

$ _
```

PrimeSieve.java

Command-line input	n (int)
Standard output	number of primes that are less than or equal to n

```
>_ ~/workspace/dsaj/programs

$ java PrimeSieve 10
```

PrimeSieve.java

Command-line input	n (int)
Standard output	number of primes that are less than or equal to n

```
>_ ~/workspace/dsaj/programs

$ java PrimeSieve 10
4
$ _
```

PrimeSieve.java

Command-line input	n (int)
Standard output	number of primes that are less than or equal to n

```
>_ ~/workspace/dsaj/programs

$ java PrimeSieve 10
4
$ java PrimeSieve 100
```

PrimeSieve.java

Command-line input	n (int)
Standard output	number of primes that are less than or equal to n

>_ ~/workspace/dsaj/programs

```
$ java PrimeSieve 10
4
$ java PrimeSieve 100
25
$ _
```

PrimeSieve.java

Command-line input	n (int)
Standard output	number of primes that are less than or equal to n

>_ ~/workspace/dsaj/programs

```
$ java PrimeSieve 10
4
$ java PrimeSieve 100
25
$ java PrimeSieve 1000
```

PrimeSieve.java

Command-line input	*n* (int)
Standard output	number of primes that are less than or equal to *n*

```
>_ ~/workspace/dsaj/programs

$ java PrimeSieve 10
4
$ java PrimeSieve 100
25
$ java PrimeSieve 1000
168
$ _
```

i	0	1	2	3	4	5	6	7	8	9	10
isPrime[i]	F	F	F	F	F	F	F	F	F	F	F

i	0	1	2	3	4	5	6	7	8	9	10
isPrime[i]	F	F	T	T	T	T	T	T	T	T	T

i	0	1	2	3	4	5	6	7	8	9	10
isPrime[i]	F	F	T	T	T	T	T	T	T	T	T

i	0	1	2	3	4	5	6	7	8	9	10
isPrime[i]	F	F	T	T	F	T	F	T	F	T	F

i	0	1	2	3	4	5	6	7	8	9	10
isPrime[i]	F	F	T	T	F	T	F	T	F	T	F

i	0	1	2	3	4	5	6	7	8	9	10
isPrime[i]	F	F	T	T	F	T	F	T	F	F	F

i	0	1	2	3	4	5	6	7	8	9	10
isPrime[i]	F	F	T	T	F	T	F	T	F	F	F

i	0	1	2	3	4	5	6	7	8	9	10
isPrime[i]	F	F	T	T	F	T	F	T	F	F	F

i	0	1	2	3	4	5	6	7	8	9	10
isPrime[i]	F	F	T	T	F	T	F	T	F	F	F

i	0	1	2	3	4	5	6	7	8	9	10
isPrime[i]	F	F	T	T	F	T	F	T	F	F	F

i	0	1	2	3	4	5	6	7	8	9	10
isPrime[i]	F	F	T	T	F	T	F	T	F	F	F

</> PrimeSieve.java

```java
import stdlib.StdOut;

public class PrimeSieve {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        boolean[] isPrime = new boolean[n + 1];
        for (int i = 2; i <= n; i++) {
            isPrime[i] = true;
        }
        for (int i = 2; i < n; i++) {
            if (isPrime[i]) {
                for (int j = 2; j <= n / i; j++) {
                    isPrime[i * j] = false;
                }
            }
        }
        int count = 0;
        for (int i = 2; i <= n; i++) {
            count += isPrime[i] ? 1 : 0;
        }
        StdOut.println(count);
    }
}
```

Aliasing refers to the situation where two variables refer to the same object

Aliasing refers to the situation where two variables refer to the same object

Example

```
int[] x = {1, 3, 7};
int[] y = x;
x[1] = 42;

StdOut.println(x[1]);
StdOut.println(y[1]);
```

writes

```
42
42
```

Example (creating a copy of an array `x` of ints)

```
int[] y = new int[x.length];

for (int i = 0; i < x.length; i++) {
    y[i] = x[i];
}
```

Two Dimensional (2D) Arrays · Creation

```
<type>[][] <name> = {{<expression>, <expression>, ...},
                     {<expression>, <expression>, ...},
                     ...
                     {<expression>, <expression>, ...}};

<type>[][] <name>;
<name> = new <type>[][] {{<expression>, <expression>, ...},
                         {<expression>, <expression>, ...},
                         ...
                         {<expression>, <expression>, ...}};
```

Two Dimensional (2D) Arrays · Creation

```
<type>[][] <name> = {{<expression>, <expression>, ...},
                     {<expression>, <expression>, ...},
                     ...
                     {<expression>, <expression>, ...}};

<type>[][] <name>;
<name> = new <type>[][] {{<expression>, <expression>, ...},
                         {<expression>, <expression>, ...},
                         ...
                         {<expression>, <expression>, ...}};
```

Example

```
int[][] identity = {{1, 0, 0},
                    0, 1, 0},
                    0, 0, 1}};

int[][] pascal;
pascal = new int[][] {{1, 0, 0, 0, 0},
                      {1, 1, 0, 0, 0},
                      {1, 2, 1, 0, 0},
                      {1, 3, 3, 1, 0},
                      {1, 4, 6, 4, 1}};
```

Two Dimensional (2D) Arrays · Creation

```
<type>[][] <name> = new <type>[<nrows>][<ncols>];

<type>[][] <name>;
<name> = new <type>[<nrows>][<ncols>];
```

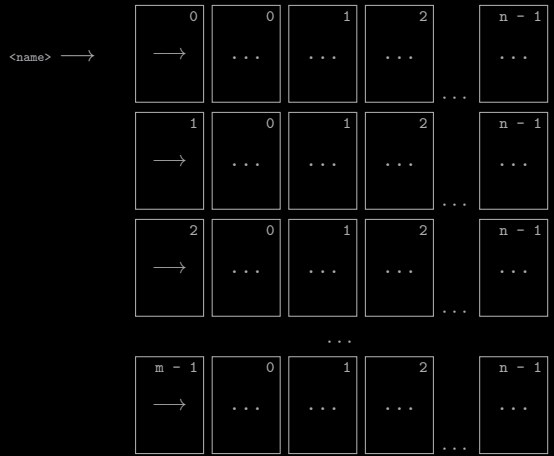
Two Dimensional (2D) Arrays · Creation

```
<type>[][] <name> = new <type>[<nrows>][<ncols>];

<type>[][] <name>;
<name> = new <type>[<nrows>][<ncols>];
```

Example

```
double[][] x = new double[10][10];

int[][] y;
y = new int[5][10];
```

Two Dimensional (2D) Arrays · Memory Model



The number of rows (say `m`) in an array `x` is obtained as `x.length`

- The number of rows (say `m`) in an array `x` is obtained as `x.length`
- The number of columns (say `n`) in an array `x` is obtained as `x[0].length`

- The number of rows (say `m`) in an array `x` is obtained as `x.length`
- The number of columns (say `n`) in an array `x` is obtained as `x[0].length`
- The element in row `i` and column `j` of an array `x` is accessed as `x[i][j]`, where `0 <= i < m` and `0 <= j < n`

Two Dimensional (2D) Arrays · Rows, Columns, and Array Elements

The number of rows (say `m`) in an array `x` is obtained as `x.length`

The number of columns (say `n`) in an array `x` is obtained as `x[0].length`

The element in row `i` and column `j` of an array `x` is accessed as `x[i][j]`, where `0 <= i < m` and `0 <= j < n`

The element in row `i` and column `j` of an array `x` is assigned a value as

```
x[i][j] = <expression>;
```

Two Dimensional (2D) Arrays · Example (Matrix Addition)

```
1 int[][] a = {{1, 2}, {3, 4}};
2 int[][] b = {{2, 3}, {4, 5}};
3
4 int[][] c = {{0, 0}, {0, 0}};
5 for (int i = 0; i < 2; i++) {
6     for (int j = 0; j < 2; j++) {
7         c[i][j] = a[i][j] + b[i][j];
8     }
9 }
```

Variable Trace

line #	c	i	j

Two Dimensional (2D) Arrays · Example (Matrix Addition)

```java
int[][] a = {{1, 2}, {3, 4}};
int[][] b = {{2, 3}, {4, 5}};

int[][] c = {{0, 0}, {0, 0}};
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
        c[i][j] = a[i][j] + b[i][j];
    }
}
```

Variable Trace

line #	c	i	j
1			

Two Dimensional (2D) Arrays · Example (Matrix Addition)

```java
int[][] a = {{1, 2}, {3, 4}};
int[][] b = {{2, 3}, {4, 5}};

int[][] c = {{0, 0}, {0, 0}};
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
        c[i][j] = a[i][j] + b[i][j];
    }
}
```

Variable Trace

line #	c	i	j
2			

Two Dimensional (2D) Arrays · Example (Matrix Addition)

```java
int[][] a = {{1, 2}, {3, 4}};
int[][] b = {{2, 3}, {4, 5}};

int[][] c = {{0, 0}, {0, 0}};
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
        c[i][j] = a[i][j] + b[i][j];
    }
}
```

Variable Trace

line #	c	i	j
4	[[0, 0], [0, 0]]		

Two Dimensional (2D) Arrays · Example (Matrix Addition)

```java
int[][] a = {{1, 2}, {3, 4}};
int[][] b = {{2, 3}, {4, 5}};

int[][] c = {{0, 0}, {0, 0}};
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
        c[i][j] = a[i][j] + b[i][j];
    }
}
```

Variable Trace

line #	c	i	j
5	[[0, 0], [0, 0]]	0	

Two Dimensional (2D) Arrays · Example (Matrix Addition)

```java
int[][] a = {{1, 2}, {3, 4}};
int[][] b = {{2, 3}, {4, 5}};

int[][] c = {{0, 0}, {0, 0}};
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
        c[i][j] = a[i][j] + b[i][j];
    }
}
```

Variable Trace

line #	c	i	j
6	[[0, 0], [0, 0]]	0	0

Two Dimensional (2D) Arrays · Example (Matrix Addition)

```java
int[][] a = {{1, 2}, {3, 4}};
int[][] b = {{2, 3}, {4, 5}};

int[][] c = {{0, 0}, {0, 0}};
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
        c[i][j] = a[i][j] + b[i][j];
    }
}
```

Variable Trace

line #	c	i	j
7	[[3, 0], [0, 0]]	0	0

Two Dimensional (2D) Arrays · Example (Matrix Addition)

```java
int[][] a = {{1, 2}, {3, 4}};
int[][] b = {{2, 3}, {4, 5}};

int[][] c = {{0, 0}, {0, 0}};
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
        c[i][j] = a[i][j] + b[i][j];
    }
}
```

Variable Trace

line #	c	i	j
6	[[3, 0], [0, 0]]	0	1

Two Dimensional (2D) Arrays · Example (Matrix Addition)

```java
int[][] a = {{1, 2}, {3, 4}};
int[][] b = {{2, 3}, {4, 5}};

int[][] c = {{0, 0}, {0, 0}};
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
        c[i][j] = a[i][j] + b[i][j];
    }
}
```

Variable Trace

line #	c	i	j
7	[[3, 5], [0, 0]]	0	1

Two Dimensional (2D) Arrays · Example (Matrix Addition)

```java
int[][] a = {{1, 2}, {3, 4}};
int[][] b = {{2, 3}, {4, 5}};

int[][] c = {{0, 0}, {0, 0}};
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
        c[i][j] = a[i][j] + b[i][j];
    }
}
```

Variable Trace

line #	c	i	j
6	[[3, 5], [0, 0]]	0	

Two Dimensional (2D) Arrays · Example (Matrix Addition)

```java
int[][] a = {{1, 2}, {3, 4}};
int[][] b = {{2, 3}, {4, 5}};

int[][] c = {{0, 0}, {0, 0}};
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
        c[i][j] = a[i][j] + b[i][j];
    }
}
```

Variable Trace

line #	c	i	j
5	[[3, 5], [0, 0]]	1	

Two Dimensional (2D) Arrays · Example (Matrix Addition)

```java
int[][] a = {{1, 2}, {3, 4}};
int[][] b = {{2, 3}, {4, 5}};

int[][] c = {{0, 0}, {0, 0}};
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
        c[i][j] = a[i][j] + b[i][j];
    }
}
```

Variable Trace

line #	c	i	j
6	[[3, 5], [0, 0]]	1	0

Two Dimensional (2D) Arrays · Example (Matrix Addition)

```java
int[][] a = {{1, 2}, {3, 4}};
int[][] b = {{2, 3}, {4, 5}};

int[][] c = {{0, 0}, {0, 0}};
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
        c[i][j] = a[i][j] + b[i][j];
    }
}
```

Variable Trace

line #	c	i	j
7	[[3, 5], [7, 0]]	1	0

Two Dimensional (2D) Arrays · Example (Matrix Addition)

```java
int[][] a = {{1, 2}, {3, 4}};
int[][] b = {{2, 3}, {4, 5}};

int[][] c = {{0, 0}, {0, 0}};
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
        c[i][j] = a[i][j] + b[i][j];
    }
}
```

Variable Trace

line #	c	i	j
6	[[3, 5], [7, 0]]	1	1

Two Dimensional (2D) Arrays · Example (Matrix Addition)

```java
int[][] a = {{1, 2}, {3, 4}};
int[][] b = {{2, 3}, {4, 5}};

int[][] c = {{0, 0}, {0, 0}};
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
        c[i][j] = a[i][j] + b[i][j];
    }
}
```

Variable Trace

line #	c	i	j
7	[[3, 5], [7, 9]]	1	1

Two Dimensional (2D) Arrays · Example (Matrix Addition)

```java
int[][] a = {{1, 2}, {3, 4}};
int[][] b = {{2, 3}, {4, 5}};

int[][] c = {{0, 0}, {0, 0}};
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
        c[i][j] = a[i][j] + b[i][j];
    }
}
```

Variable Trace

line #	c	i	j
6	[[3, 5], [7, 9]]	1	

Two Dimensional (2D) Arrays · Example (Matrix Addition)

```java
int[][] a = {{1, 2}, {3, 4}};
int[][] b = {{2, 3}, {4, 5}};

int[][] c = {{0, 0}, {0, 0}};
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
        c[i][j] = a[i][j] + b[i][j];
    }
}
```

Variable Trace

line #	c	i	j
5	[[3, 5], [7, 9]]		

Two Dimensional (2D) Arrays · Example (Matrix Addition)

```
1 int[][] a = {{1, 2}, {3, 4}};
2 int[][] b = {{2, 3}, {4, 5}};
3
4 int[][] c = {{0, 0}, {0, 0}};
5 for (int i = 0; i < 2; i++) {
6     for (int j = 0; j < 2; j++) {
7         c[i][j] = a[i][j] + b[i][j];
8     }
9 }
```

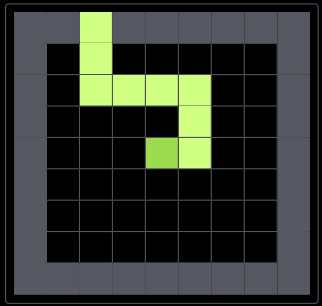
Variable Trace

line #	c	i	j

SelfAvoid.java

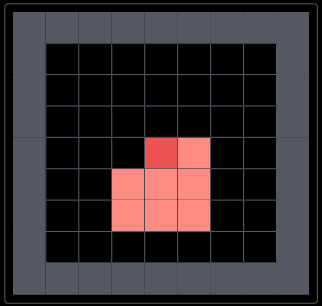
Command-line input	*n* (int) and *trials* (int)
Standard output	percentage of dead ends encountered in *trials* self-avoiding random walks on an $n \times n$ lattice

Escape



→ ↑ ↑ ← ← ← ↑ ↑

Dead End



→ ↓ ↓ ← ← ↑ →

```
>_ ~/workspace/dsaj/programs
$ _
```

```
>_ ~/workspace/dsaj/programs
$ java SelfAvoid 20 1000
```

```
>_ ~/workspace/dsaj/programs

$ java SelfAvoid 20 1000
32% dead ends
$ _
```

```
>_ ~/workspace/dsaj/programs

$ java SelfAvoid 20 1000
32% dead ends
$ java SelfAvoid 40 1000
```

```
>_ ~/workspace/dsaj/programs

$ java SelfAvoid 20 1000
32% dead ends
$ java SelfAvoid 40 1000
75% dead ends
$ _
```

```
>_ ~/workspace/dsaj/programs

$ java SelfAvoid 20 1000
32% dead ends
$ java SelfAvoid 40 1000
75% dead ends
$ java SelfAvoid 80 1000
```

```
>_ ~/workspace/dsaj/programs

$ java SelfAvoid 20 1000
32% dead ends
$ java SelfAvoid 40 1000
75% dead ends
$ java SelfAvoid 80 1000
98% dead ends
$ _
```

Two Dimensional (2D) Arrays · Example (Self-avoiding Random Walk)

</> SelfAvoid.java

```
import stdlib.StdOut;
import stdlib.StdRandom;

public class SelfAvoid {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        int trials = Integer.parseInt(args[1]);
        int deadEnds = 0;
        for (int t = 0; t < trials; t++) {
            boolean[][] a = new boolean[n][n];
            int x = n / 2;
            int y = n / 2;
            while (x > 0 && x < n - 1 && y > 0 && y < n - 1) {
                a[x][y] = true;
                if (a[x - 1][y] && a[x + 1][y] && a[x][y - 1] && a[x][y + 1]) {
                    deadEnds++;
                    break;
                }
                int r = StdRandom.uniform(1, 5);
                if (r == 1 && !a[x + 1][y]) {
                    x++;
                } else if (r == 2 && !a[x - 1][y]) {
                    x--;
                } else if (r == 3 && !a[x][y + 1]) {
                    y++;
                } else if (r == 4 && !a[x][y - 1]) {
                    y--;
                }
            }
        }
        StdOut.println(100 * deadEnds / trials + "% dead ends");
    }
}
```

Converting 2D Arrays to 1D Arrays and Vice Versa

Converting 2D Arrays to 1D Arrays and Vice Versa

Converting an $m \times n$ array X into a 1D array Y

- The element $X(i, j)$ maps to the element $Y(k)$, where $k = n \cdot i + j$

Converting 2D Arrays to 1D Arrays and Vice Versa

Converting an $m \times n$ array X into a 1D array Y

- The element $X(i, j)$ maps to the element $Y(k)$, where $k = n \cdot i + j$

Converting a 1D array Y of size l into an $m \times n$ array X

- The element $Y(k)$ maps to the element $X(i, j)$, where $i = \left\lfloor \frac{k}{n} \right\rfloor$, $j = k \bmod n$, and $m = \frac{l}{n}$

Converting 2D Arrays to 1D Arrays and Vice Versa

Converting an $m \times n$ array X into a 1D array Y

- The element $X(i, j)$ maps to the element $Y(k)$, where $k = n \cdot i + j$

Converting a 1D array Y of size l into an $m \times n$ array X

- The element $Y(k)$ maps to the element $X(i, j)$, where $i = \left\lfloor \frac{k}{n} \right\rfloor$, $j = k \bmod n$, and $m = \frac{l}{n}$

Example

$$X = \begin{array}{c} \begin{matrix} 0 & 1 & 2 & 3 & 4 \end{matrix} \\ \begin{bmatrix} A & B & C & D & E \\ F & G & H & I & J \\ K & L & M & N & O \end{bmatrix} \end{array} \begin{array}{c} \\ 0 \\ 1 \\ 2 \end{array}$$

$$Y = \begin{array}{c} \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 \end{matrix} \\ \begin{bmatrix} A & B & C & D & E & F & G & H & I & J & K & L & M & N & O \end{bmatrix} \end{array}$$

Ragged Arrays

Ragged Arrays

A ragged array is a 2D array in which the rows have unequal number of columns

Ragged Arrays

A ragged array is a 2D array in which the rows have unequal number of columns

The number of columns in the `i`th row of a ragged array `x` is obtained as `x[i].length`

Ragged Arrays

A ragged array is a 2D array in which the rows have unequal number of columns

The number of columns in the `i`th row of a ragged array `x` is obtained as `x[i].length`

Example (printing a ragged array)

```
int[][] pascal = {{1},
                  {1, 1},
                  {1, 2, 1},
                  {1, 3, 3, 1},
                  {1, 4, 6, 4, 1}};

for (int i = 0; i < pascal.length; i++) {
    for (int j = 0; j < pascal[i].length; j++) {
        StdOut.print(pascal[i][j] + " ");
    }
    StdOut.println();
}
```

writes

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```