Sag, wie's ist!

**Die Befragungen zum CHE Ranking stehen an.
In diesem Jahr in den Fächern**

- **BWL**
- **Rechtswissenschaft**
- **Soziale Arbeit**
- **VWL**
- **Wirtschaftsinformatik**
- **Wirtschaftsingenieurwesen**
- **Wirtschaftspsychologie**
- **Wirtschaftsrecht**
- **Wirtschaftswissenschaften**

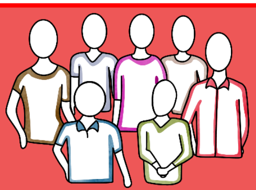
Ergebnisse und Infos werden unter **www.zeit.de/che-ranking** und im **ZEIT Studienführer 2020/21** veröffentlicht.

CHE
Ranking

Einladungen zu den Befragungen schickt die Hochschule direkt an Euch.

CHE Ranking — Die Studierendenbefragung

auf einen Blick



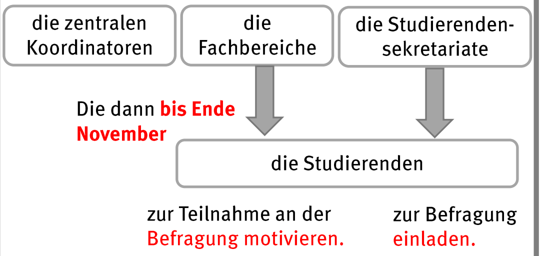
Befragt werden Studierende ab dem zweiten Studienjahr u.a. zu

- Lehrangebot
- Betreuung
- Forschungsbezug
- Praxisorientierung
- Ausstattung

In ihrem jeweiligen Studiengang.

Der Ablauf der Befragung

Das CHE informiert über den **Befragungsstart ab Oktober** und stellt **Info- und Werbematerial** zur Verfügung für



die zentralen Koordinatoren | die Fachbereiche | die Studierenden-sekretariate

Die dann **bis Ende November**

die Studierenden

zur Teilnahme an der Befragung motivieren.

zur Befragung einladen.

Das CHE bietet über ein individuell nutzbares Infoportal Informationen zum Stand der Erhebungen an. Online darauf zugreifen können die **zentralen Koordinatoren** und die **Fachbereiche**

Service für Hochschulen

Zugang zum Infoportal des CHE Hochschulrankings mit

- FAQs
- Muster-Fragebögen
- Stand der Erhebungen

Erklärvideos zum Ablauf der Befragungen

Werbematerial

Zugang zum Hochschulranking bei ZEIT online

Hochschulspezifische Auswertungen der Ergebnisse

Persönliche Ansprechpartner beim CHE für Rückfragen

Das aktuelle CHE Hochschulranking beinhaltet die Urteile von rund **150.000 Studierenden**.

Sag, wie's ist!

Die Befragungen zum U-Multirank und CHE Ranking stehen an. In diesem Jahr in den Fächern

- Naturwissenschaften
- Mathematik
- Informatik
- Sport/Sportwissenschaft
- Medizin/Pflege
- Politikwissenschaft
- Psychologie

Ergebnisse werden unter www.umultirank.org www.zeit.de/che-ranking und im ZEIT Studienführer 2018/19 veröffentlicht.

Mehr Infos zu U-Multirank

Mehr Infos zum CHE Ranking

Einladungen zu den Befragungen schickt die Hochschule direkt an Euch.

CHE Ranking

multirank
Universities compared. Your way.

Werbematerial mit Platz fürs eigene Hochschul-Logo

Die Ergebnisse werden ab einer Beteiligung von 15 Studierenden oder eine Quote von 10 % veröffentlicht.

100 % anonym

Das CHE besitzt keine E-Mailadressen der Studierenden.

Five-Minute Review

1. What elements may a class definition contain?
2. Which perspectives on class design do we distinguish?
3. What is *overloading*? *Overriding*?
4. What is *layered abstraction*?
5. What is **`this`**? **`null`**?

Five-Minute Review

1. What are *local/instance/class variables*? What about *constants*?
2. What is an *array*?
3. How do we locally declare an array of 5 integers?
4. What coding advice for float's do you know? What is the rationale for it?
5. What is an *immutable* class?

Programming – Lecture 7

Arrays and ArrayLists (**Chapter 11**)

- Array concepts + vocabulary
- Internal array representation
- Arrays for tabulation, cryptograms
- Array initialization
- Image processing, bit operators
- **`ArrayList`** class
- Primitive types vs. objects, wrapper classes, boxing/unboxing (**Chapter 8**)

Arrays

- *Array* ("*Feld*"): ordered collection of homogeneous (i.e., same type) data
- *Array declaration*: *type* `[]` *name* `= new` *type*`[`*n*`];`
- Example: `int[] intArray = new int[10];`
- *Elements*: individual data in array
- *Element types*: type of elements
- *Array length*: number of elements
- *Element index*: position of element in array
- Automatic initialization to *default value* (0, `false`, `null`)
- *Array selection*: *name*`[`*index*`]`

Cycling through Array Elements

Pattern:

```
for (int i = 0; i < array.length; i++) {
     Operations involving the $i^{th}$ element of the array
}
```

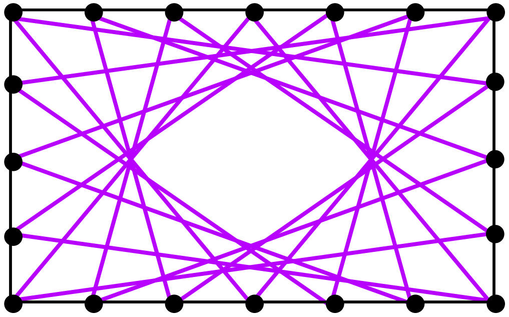
Reset every element in **`intArray`** to zero:

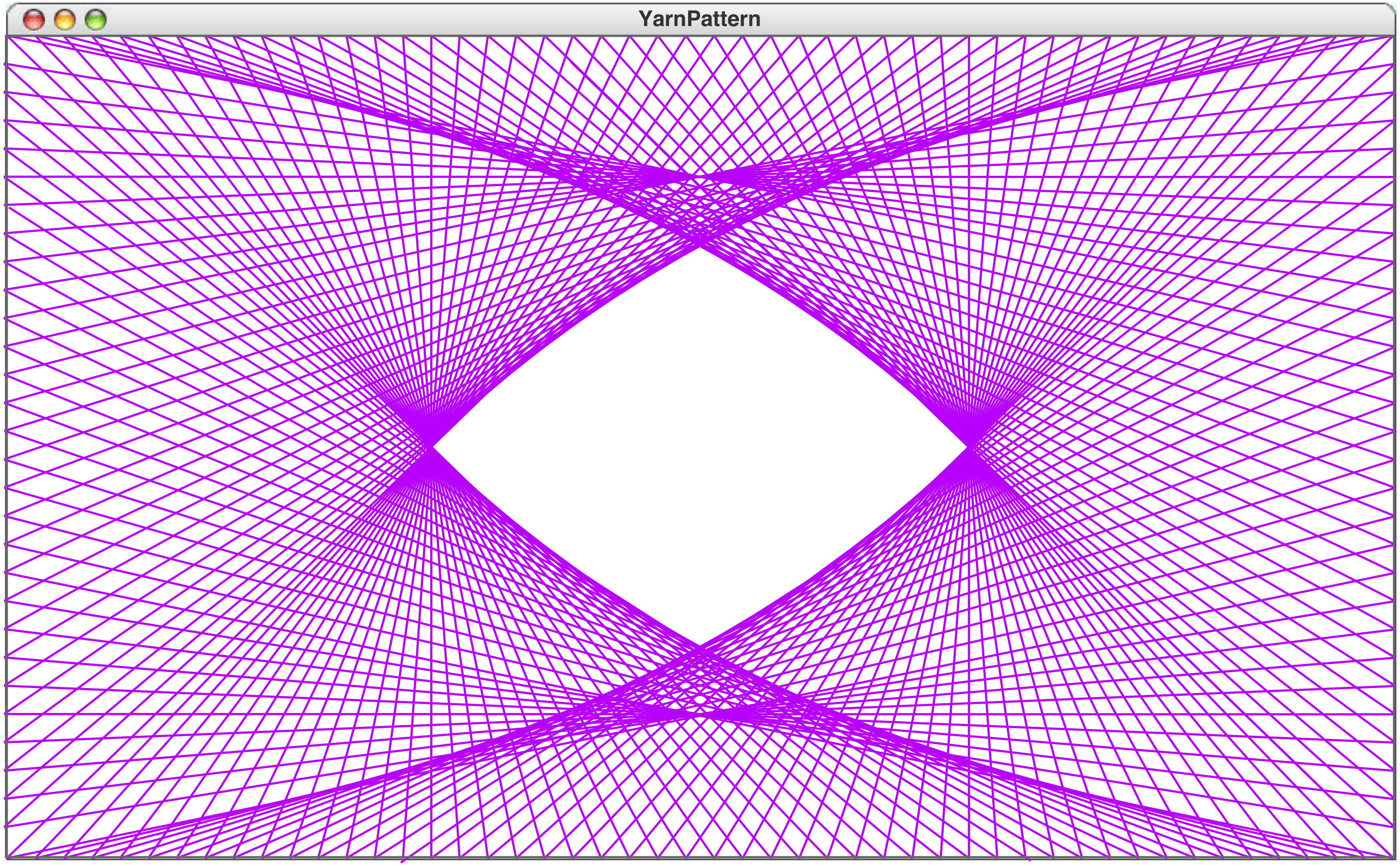
```
for (int i = 0; i < intArray.length;
      i++) {
   intArray[i] = 0;
}
```

Summing an Array

```
/**
 * Calculates the sum of an integer array.
 * @param array An array of integers
 * @return sum of values in the array
 */
private int sumArray(int[] array) {
   int sum = 0;
   for (int i = 0; i < array.length;
        i++) {
      sum += array[i];
   }
   return sum;
}
```

YarnPattern




YarnPattern

```
import acm.graphics.*;
import acm.program.*;
import java.awt.*;

/**
 * This program creates a pattern that simulates the process of
 * winding a piece of colored yarn around an array of pegs along
 * the edges of the canvas.
 */
public class YarnPattern extends GraphicsProgram {

   public void run() {
      initPegArray();
      int thisPeg = 0;
      int nextPeg;
      do {
         nextPeg = (thisPeg + DELTA) % N_PEGS;
         GPoint p0 = pegs[thisPeg];
         GPoint p1 = pegs[nextPeg];
         GLine line = new GLine(p0.getX(), p0.getY(), p1.getX(), p1.getY());
         line.setColor(Color.MAGENTA);
         add(line);
         thisPeg = nextPeg;
      } while (thisPeg != 0);
   }
```

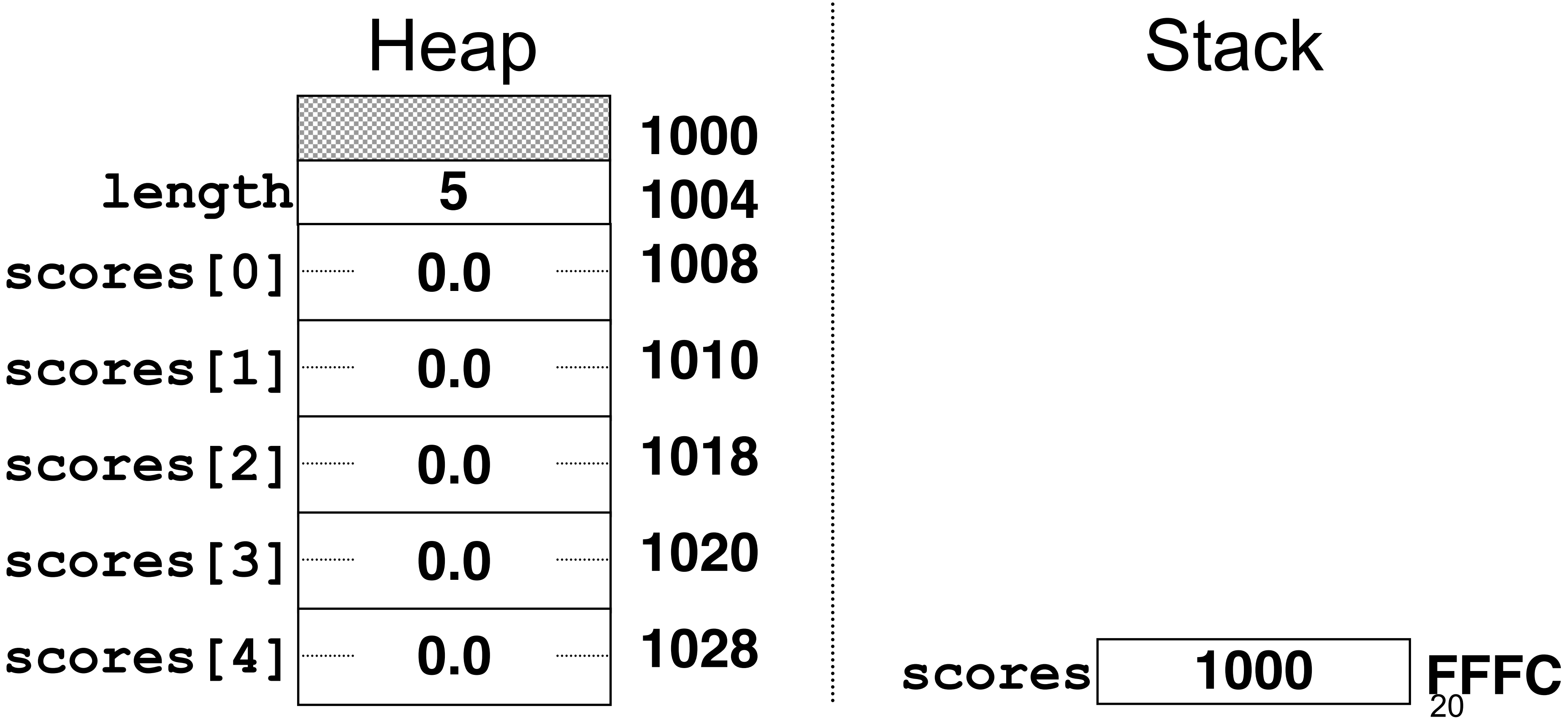
page 1 of 2

skip code

```
/* Initializes the array of pegs */
   private void initPegArray() {
      int pegIndex = 0;
      for (int i = 0; i < N_ACROSS; i++) {
         pegs[pegIndex++] = new GPoint(i * PEG_SEP, 0);
      }
      for (int i = 0; i < N_DOWN; i++) {
         pegs[pegIndex++] = new GPoint(N_ACROSS * PEG_SEP, i * PEG_SEP);
      }
      for (int i = N_ACROSS; i > 0; i--) {
         pegs[pegIndex++] = new GPoint(i * PEG_SEP, N_DOWN * PEG_SEP);
      }
      for (int i = N_DOWN; i > 0; i--) {
         pegs[pegIndex++] = new GPoint(0, i * PEG_SEP);
      }
   }

/* Private constants */
   private static final int DELTA = 67;    /* How many pegs to advance     */
   private static final int PEG_SEP = 10;  /* Pixels separating each peg   */
   private static final int N_ACROSS = 50; /* Pegs across (minus a corner) */
   private static final int N_DOWN = 30;   /* Pegs down (minus a corner)   */
   private static final int N_PEGS = 2 * N_ACROSS + 2 * N_DOWN;

/* Private instance variables */
   private GPoint[] pegs = new GPoint[N_PEGS];

}
```

Internal Representation of Arrays

```
double[] scores = new double[5];
```



Heap

length 5 1004

1000

scores[0] 0.0 1008

scores[1] 0.0 1010

scores[2] 0.0 1018

scores[3] 0.0 1020

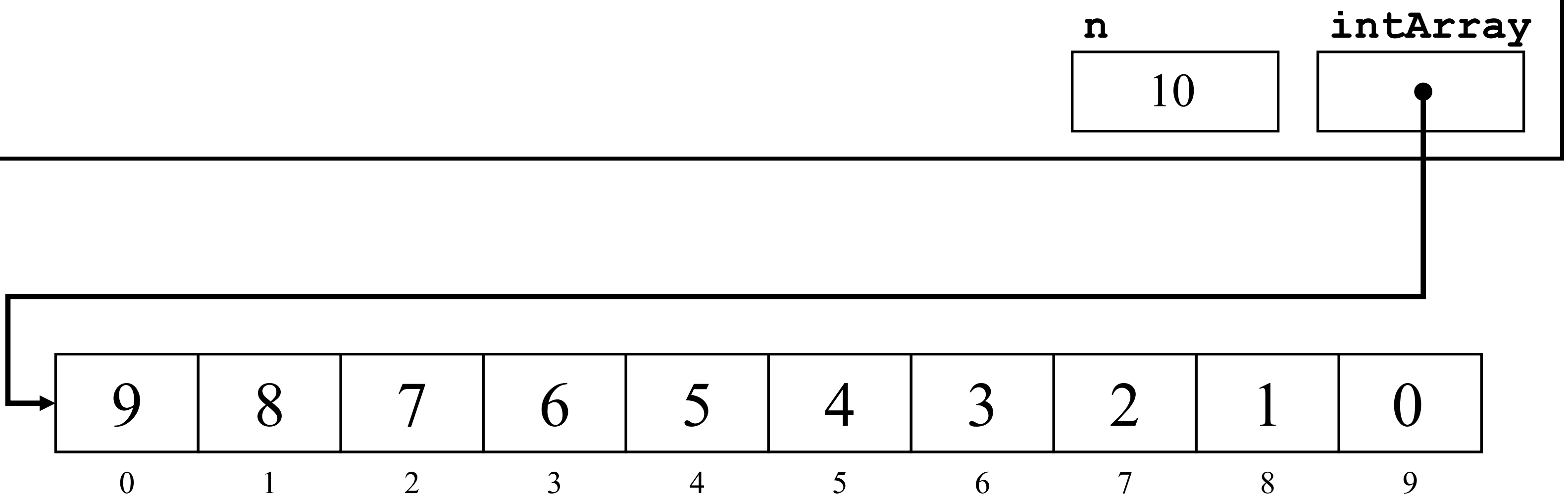
scores[4] 0.0 1028

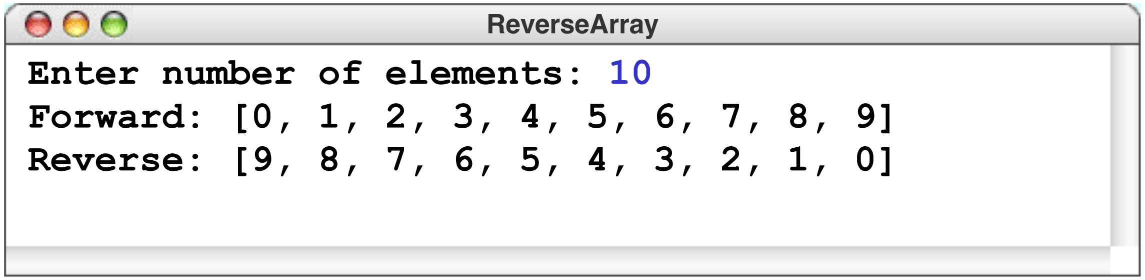
Stack

scores 1000 FFFC

ReverseArray

```
public void run() {
   int n = readInt("Enter number of elements: ");
   int[] intArray = createIndexArray(n);
   println("Forward: " + arrayToString(intArray));
   reverseArray(intArray);
   println("Reverse: " + arrayToString(intArray));
}
```

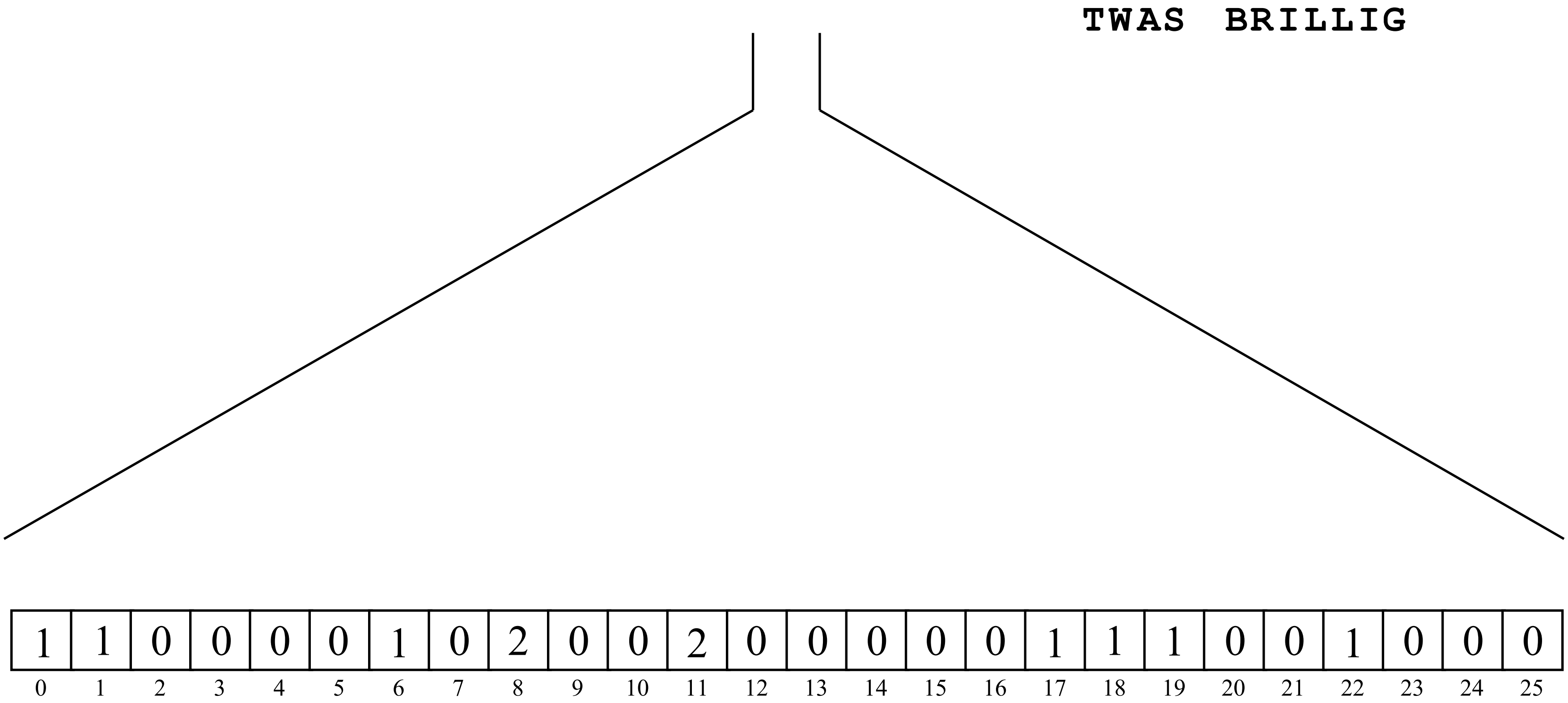




```
Enter number of elements: 10
Forward: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Reverse: [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

skip simulation

Arrays for Tabulation – Cryptograms



TWAS BRILLIG

1	1	0	0	0	0	1	0	2	0	0	2	0	0	0	0	0	1	1	1	0	0	1	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

CountLetterFrequencies

```
import acm.program.*;

/**
 * This program creates a table of the letter frequencies in a
 * paragraph of input text terminated by a blank line.
 */
public class CountLetterFrequencies extends ConsoleProgram {

   /* Private instance variables */
   private int[] frequencyTable = new int[26];

   public void run() {
      println("This program counts letter frequencies.");
      println("Enter a blank line to indicate the end of the text.");
      while (true) {
         String line = readLine();
         if (line.length() == 0) break;
         countLetterFrequencies(line);
      }
      printFrequencyTable();
   }
```

page 1 of 2

skip code

```
   /* Counts the letter frequencies in a line of text */
   private void countLetterFrequencies(String line) {
      for (int i = 0; i < line.length(); i++) {
         char ch = line.charAt(i);
         if (Character.isLetter(ch)) {
            int index = Character.toUpperCase(ch) - 'A';
            frequencyTable[index]++;
         }
      }
   }

   /* Displays the frequency table */
   private void printFrequencyTable() {
      for (char ch = 'A'; ch <= 'Z'; ch++) {
         int index = ch - 'A';
         println(ch + ": " + frequencyTable[index]);
      }
   }

}
```

skip code

Initializing Arrays

In Java, unlike in C/C++, arrays are automatically initialized: 0 for int’s, false for Booleans, null for references.

For other initializations:

type[] *name* = { *elements* };

```
int[] powersOfTen =
   { 1, 10, 100, 1000, 10000 };
```

A Method for Month Names

```
private String nameForMonth(int month) {
    switch (month) {
    case 1: return "January";
    case 2: return "February";
    case 3: return "March";
    case 4: return "April";
    case 5: return "May";
    //...
    default: return null;
}
```

A Method for Month Names

```
private String nameForMonth(int month) {
    switch (month) {
    case 1: return "January";
    case 2: return "February";
    case 3: return "March";
    case 4: return "April";
    case 5: return "May";
    //...
    default:
        throw new IllegalArgumentException(
            "month must be between 1 and 12.");
}
```

Constant Lookup Tables

```
private static final String[]
MONTH_NAMES = {
   null /* Included because there
           is no month #0 */,
   "January", "February", "March",
   "April", "May", "June",
   "July", "August", "September",
   "October", "November", "December"
 };
```

Note: `final` only ensures that `MONTH_NAMES` cannot be changed. Its elements can still change.

Multidimensional Arrays

```
int[][] m = new int[2][3];
```

`m[0][0]`	`m[0][1]`	`m[0][2]`
`m[1][0]`	`m[1][1]`	`m[1][2]`

Initialization:

```
int[][] m = { {  0,  1,  2 },
              { 10, 11, 12 } };
```

Multidimensional Arrays

… are arrays of arrays (unlike in C!)

```
int[][] m = { {  0,  1,  2 },
              { 10, 11, 12 } };
println(m[0][0] + " " + m[1][0]);



m[0] = m[1];  // Not possible in C!
println(m[0][0] + " " + m[1][0]);



m[0][0] = 42; m[1][0] = 43;
println(m[0][0] + " " + m[1][0]);
```

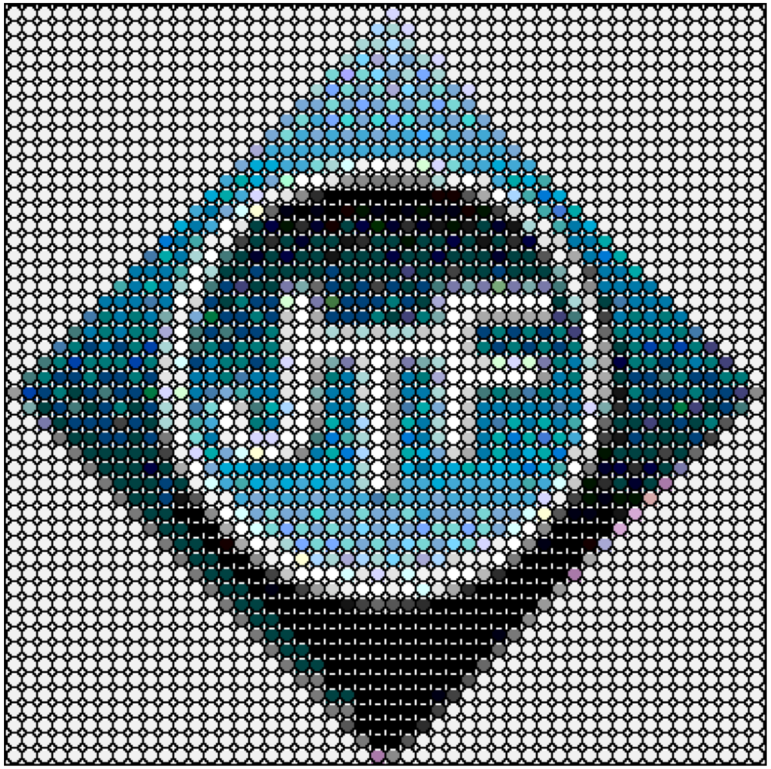
Arrays of Objects

```
GOval[] ovals;
```

```
GOval[] ovals = new GOval[5];
```

```
{ null, null, null, null, null }
```

Image Processing



Pixel Arrays

```
GImage logo = new GImage("JTFLogo.gif");
int[][] pixels = logo.getPixelArray();
```

`pixels` is an array of rows,
each row is an array of pixels.

`pixels[y][x]` retrieves pixel in row y, column x

Image height:

`pixels.length`

Image width:

`pixels[0].length`

Pixel Values

$$\underbrace{1\,1\,1\,1\,1\,1\,1\,1}_{\textit{transparency } (\alpha)}\underbrace{1\,0\,0\,1\,1\,0\,0\,1}_{\textit{red}}\underbrace{0\,1\,1\,0\,0\,1\,1\,0}_{\textit{green}}\underbrace{0\,0\,1\,1\,0\,0\,1\,1}_{\textit{blue}}$$

Transparency: **`0`** = transparent, **`255`** = opaque

RGB value **`0xFF996633`**:

FlipVertical

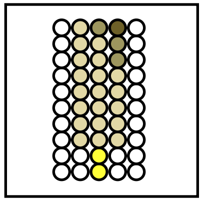
```
public void run() {
```

```
private GImage flipVertical(GImage image) {
   int[][] array = image.getPixelArray();
   int height = array.length;
   for (int p1 = 0; p1 < height / 2; p1++) {
      int p2 = height - p1 - 1;
      int[] temp = array[p1];
      array[p1] = array[p2];
      array[p2] = temp;
   }
   return new GImage(array);
}
```

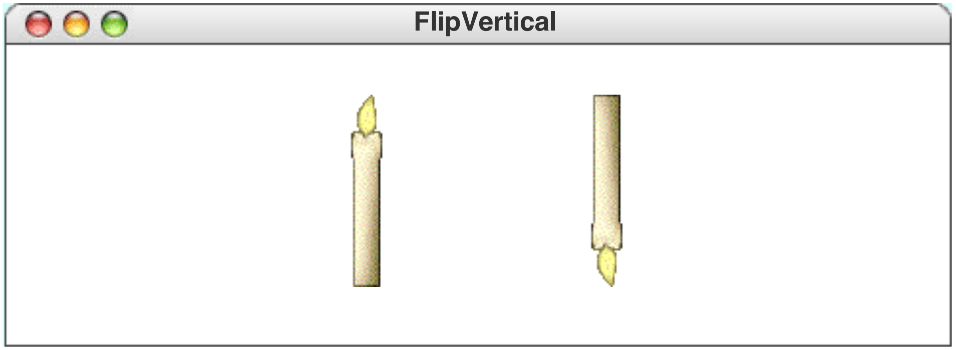
height

100

array



image



skip simulation

Bitwise Operators

x **&** *y*	**Bitwise AND.** The result has a 1 bit wherever both *x* and *y* have 1s.
x **|** *y*	**Bitwise OR.** The result has a 1 bit wherever either *x* or *y* have 1s.
x **^** *y*	**Exclusive OR.** The result has a 1 bit wherever *x* and *y* differ.
~*x*	**Bitwise NOT.** The result has a 1 bit wherever *x* has a 0.
x **<<** *n*	**Left shift.** Shift the bits in *x* left *n* positions, shifting in 0s.
x **>>** *n*	**Right shift (arithmetic).** Shift *x* right *n* bits, replicating the sign bit (leftmost bit).
x **>>>** *n*	**Right shift (logical).** Shift *x* right *n* bits, shifting in 0s.

Bitwise AND

&

	0	1
0	0	0
1	0	1

Primary use: **masking**

Example: select blue component of pixel value

	1	1	1	1	1	1	1	1	1	0	0	1	1	0	0	1	0	1	1	0	0	1	1	0	0	0	1	1	0	0	1	1
&	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1

Bitwise OR

|

	0	1
0	0	1
1	1	1

Primary use: assemble single integer value

Example: Convert an RGB value into an opaque pixel value

	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	0	1	0	1	1	1	1
|	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	0	1	0	1	1	1	1

Exercise: Shift Operators

Suppose **`pixel`** contains this bit pattern:

1	1	1	1	1	1	1	1	1	0	0	1	1	0	0	1	0	1	1	0	0	1	1	0	0	0	1	1	0	0	1	1

1. What is the value of **`pixel << 2`**?

1	1	1	1	1	1	1	1	1	0	0	1	1	0	0	1	0	1	1	0	0	1	1	0	0	0	1	1	0	0	1	1

2. What is the value of **`pixel >> 8`**?

1	1	1	1	1	1	1	1	1	0	0	1	1	0	0	1	0	1	1	0	0	1	1	0	0	0	1	1	0	0	1	1

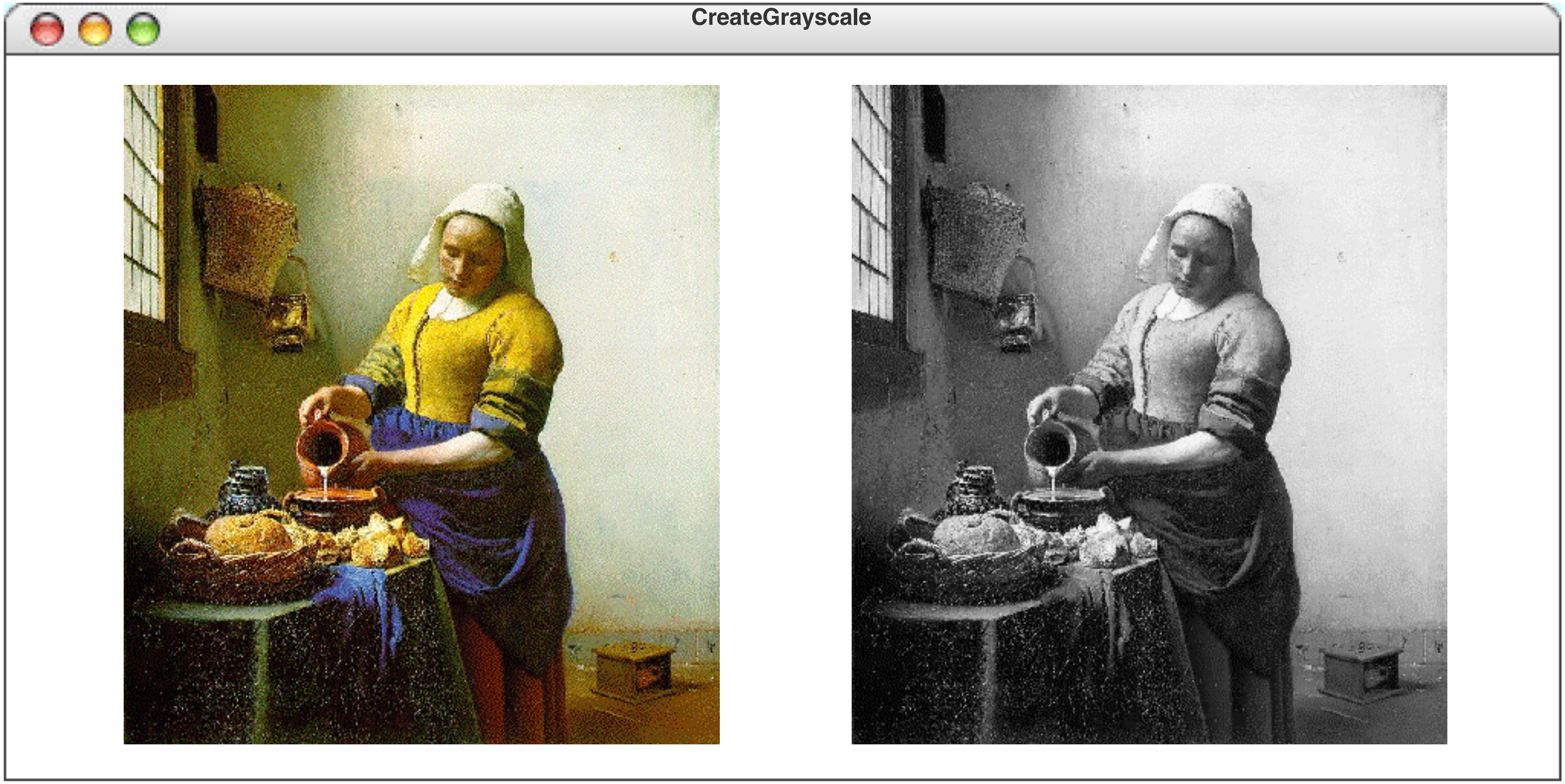
3. What is the value of **`pixel >>> 24`**?

1	1	1	1	1	1	1	1	1	0	0	1	1	0	0	1	0	1	1	0	0	1	1	0	0	0	1	1	0	0	1	1

Please visit http://pingo.upb.de/643250

Vera hat den Verstand verloren.
Ich bin das Einhorn auf dem Eis!!

Creating a Grayscale Image

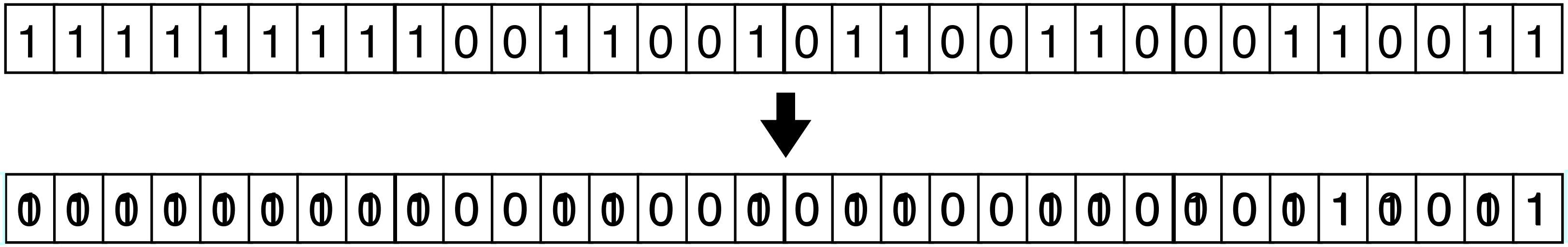


```java
/* Creates a grayscale version of the original image */
private GImage createGrayscaleImage(GImage image) {
   int[][] array = image.getPixelArray();
   int height = array.length;
   int width = array[0].length;
   for (int i = 0; i < height; i++) {
      for (int j = 0; j < width; j++) {
         int pixel = array[i][j];
         int r = GImage.getRed(pixel);
         int g = GImage.getGreen(pixel);
         int b = GImage.getBlue(pixel);
         int xx = computeLuminosity(r, g, b);
         array[i][j] = GImage.createRGBPixel(xx, xx, xx);
      }
   }
   return new GImage(array);
}

/* Calculates pixel luminosity using the NTSC formula */
private int computeLuminosity(int r, int g, int b) {
   return GMath.round(0.299 * r + 0.587 * g + 0.114 * b);
}
```

Manipulating Pixel Values

Isolate red:

`(pixel >> 16) & 0xFF`



1 1 1 1 1 1 1 1 1 0 0 1 1 0 0 1 0 1 1 0 0 1 1 0 0 0 1 1 0 0 1 1

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1

Given RGB values **`r`**, **`g`**, and **`b`**, compute pixel value for corresponding opaque color:

```
(0xFF << 24) |
(r << 16) | (g << 8) | b
```

Static Methods in `GImage`

```
/** Returns alpha component from RGB value. */
   public static int getAlpha(int pixel) {
      return (pixel >> 24) & 0xFF; }

/** Returns red component from RGB value. */
   public static int getRed(int pixel) {
      return (pixel >> 16) & 0xFF; }

/** Returns green component from RGB value. */
   public static int getGreen(int pixel) {
      return (pixel >> 8) & 0xFF; }

/** Returns blue component from RGB value. */
   public static int getBlue(int pixel) {
      return pixel & 0xFF; }
```

Static Methods in `GImage`

```
/** Creates opaque pixel value
    from color components */
   public static int createRGBPixel(int r,
       int g, int b) {
     return createRGBPixel(r, g, b, 0xFF);
   }

/** Creates pixel value from color components,
    including alpha */
   public static int createRGBPixel(int r,
       int g, int b, int alpha) {
     return (alpha << 24) | (r << 16) |
            (g << 8) | b;
   }
```

`ArrayList` Class

Alternative to Arrays: **`ArrayList`** Class

- A class, not special language form
- **Pro:** more flexible, allows add/remove
- **Con:** typically not as fast as Arrays

`ArrayList` is *generic* (*class*):

- class parameterized over types
- a.k.a. *template*, *parameterized class*

ArrayList Class

Syntax: **ArrayList<*type*>**

```
import java.util.*;
ArrayList<String> strList =
    new ArrayList<>();
String str = "hello";
strList.add(str);
strList.add("there");
str = strList.get(1);
```

`ArrayList` Class Methods

`boolean add(T element)`

Adds a new element to the end of the **`ArrayList`**; the return value is always **`true`**.

`void add(int index, T element)`

Inserts a new element into the **`ArrayList`** before the position specified by **`index`**.

`T remove(int index)`

Removes the element at the specified position and returns that value.

`boolean remove(T element)`

Removes the first instance of **`element`**, if it appears; returns **`true`** if a match is found.

`void clear()`

Removes all elements from the **`ArrayList`**.

`int size()`

Returns the number of elements in the **`ArrayList`**.

`T get(int index)`

Returns the object at the specified index.

`T set(int index, T value)`

Sets the element at the specified index to the new value and returns the old value.

`int indexOf(T value)`

Returns the index of the first occurrence of the specified value, or −1 if it does not appear.

`boolean contains(T value)`

Returns **`true`** if the **`ArrayList`** contains the specified value.

`boolean isEmpty()`

Returns **`true`** if the **`ArrayList`** contains no elements.

Reversing an **ArrayList**

```
import acm.program.*;
import java.util.*;

/**
 * This program reads in a list of integers and then displays that list in
 * reverse order. This version uses an ArrayList<Integer> to hold the values.
 */
public class ReverseArrayList extends ConsoleProgram {

   /* Private constants */
   private static final int SENTINEL = 0;

   public void run() {
      println("This program reverses the elements in an ArrayList.");
      println("Use " + SENTINEL + " to signal the end of the list.");
      ArrayList<Integer> list = readArrayList();
      reverseArrayList(list);
      printArrayList(list);
   }

   /* Reads the data into the list */
   private ArrayList<Integer> readArrayList() {
      ArrayList<Integer> list = new ArrayList<>();
      while (true) {
         int value = readInt(" ? ");
         if (value == SENTINEL) break;
         list.add(value);
      }
      return list;
   }
```

page 1 of 2

skip code

```
   /* Prints the data from the list, one element per line */
   private void printArrayList(ArrayList<Integer> list) {
      for (int i = 0; i < list.size(); i++) {
         int value = list.get(i);
         println(value);
      }
   }

   /* Reverses the data in an ArrayList */
   private void reverseArrayList(ArrayList<Integer> list) {
      for (int i = 0; i < list.size() / 2; i++) {
         swapElements(list, i, list.size() - i - 1);
      }
   }

   /* Exchanges two elements in an ArrayList */
   private void swapElements(ArrayList<Integer> list, int p1, int p2) {
      int temp = list.get(p1);
      list.set(p1, list.get(p2));
      list.set(p2, temp);
   }
}
```

Wrapper Classes

Type of array list elements must be class type (i.e., not primitive)

```
ArrayList<Integer> list =
    new ArrayList<>();
list.add(42);
int answer = list.get(0);
```

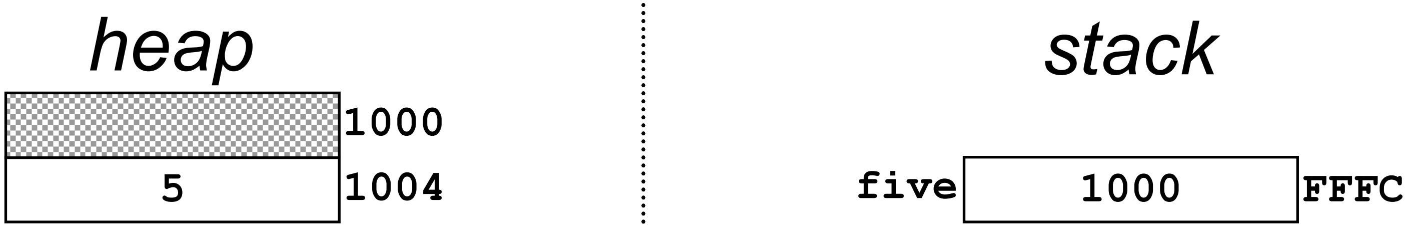
Wrapper Classes

- Convenient when object types (instead of primitive types) are required, as in **`ArrayList`**
- Are *immutable* – state cannot be changed anymore
- Other immutable classes: **`String`**, **`Rational`**

Wrapper Classes

boolean	↔	Boolean	float	↔	Float
byte	↔	Byte	int	↔	Integer
char	↔	Character	long	↔	Long
double	↔	Double	short	↔	Short

```
Integer five = new Integer(5);
```



heap

stack

heap	address
▨	1000
5	1004

five | 1000 | FFFC

```
int six = five.intValue() + 1;
```

(Auto) Boxing and Unboxing

```
Integer five = new Integer(5);
int six = five.intValue() + 1;
```

vs.

```
Integer five = 5;     // Boxing 5
int six = five + 1;   // Unboxing five
```

Enhanced for Statement

```
ArrayList<String> nameList;

for (int i = 0; i < nameList.size(); i++)
{
  String name = nameList.get(i);
  println(name);
}
```

can be abbreviated to *enhanced for statement*:

```
for (String name : nameList) {
  println(name);
}
```

Enhanced for Statement

This works for classes that implement the **`Iterable`** interface (e.g., **`ArrayList`**) and arrays:

```
int[] ints = ...;

for (int i = 0; i < ints.length; i++) {
  int val = ints[i];
  println(val);
}
```

can be abbreviated to:

```
for (int val : ints) {
  println(val);
}
```

Summary

- Arrays are *ordered* collections of *homogeneous* element type
- Array elements are selected with an *index*, starting at 0
- In Java, arrays are implemented as objects, stored on the *heap*; an *array variable*, typically stored on the *stack*, is a *reference* to the array
- Arrays may be *initialized* as part of the declaration
- Images can be represented as two-dimensional arrays, with 32-bit pixel values
- The **`ArrayList`** class is a *generic class* that also allows adding and deleting elements
- The *enhanced for* statement is convenient to iterate through arrays and **`Iterable`** classes