DATA STRUCTURES

WENYE LI
CUHK-SZ

OUTLINE

- **Basics in Computer Memory**
    - Partially go beyond the current scope.
    - Will come back to detailed discussion after a few weeks.
- Arrays
- Implementation
- Examples

STRUCTURE OF MEMORY

- The basic unit of memory is called a **bit**, either 0 or 1.
- In most modern architectures, the smallest unit on which the hardware operates is a sequence of eight consecutive bits called **byte**.
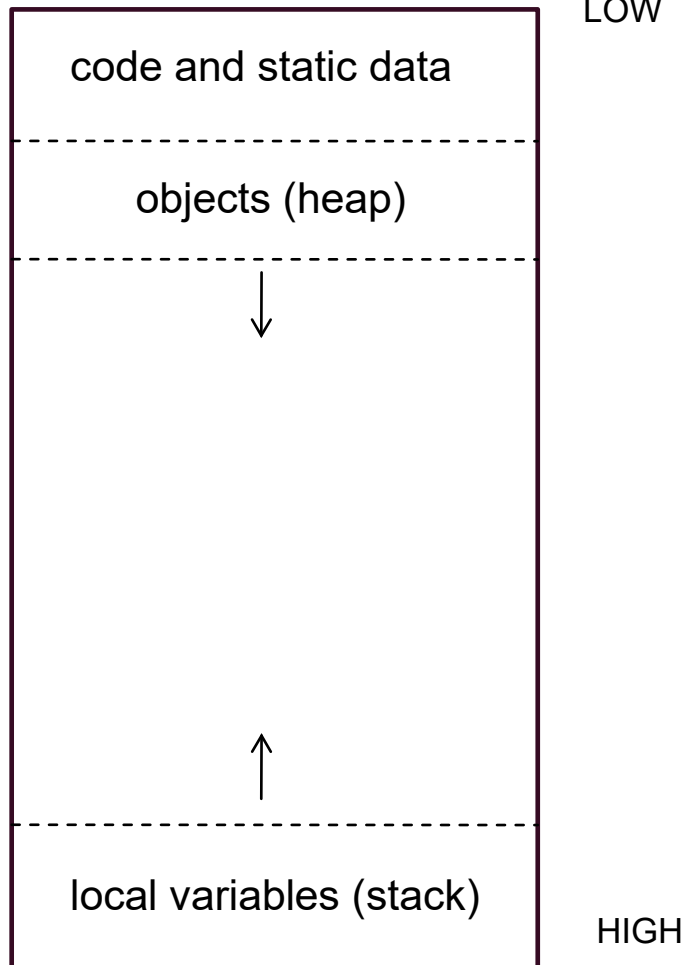
a binary (executable) file

0   1   2   3

01011001100001001001111011000011...

- Numbers and instructions are stored in still larger units, mostly common a **word**. Because machines have different architectures, the number of bytes and the order of bytes in a word vary from machine to machine.
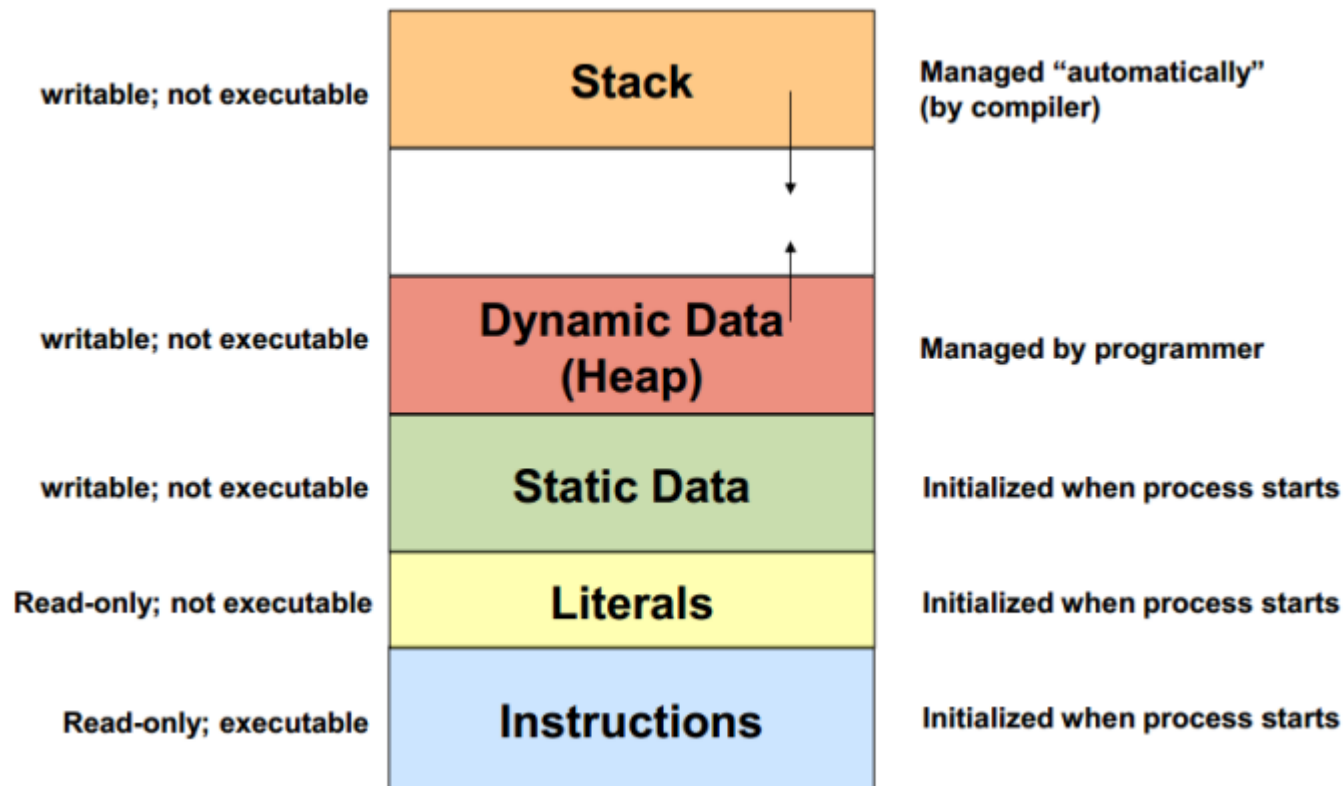
NUMBERS, BASES, AND CONVERSION

- $(21)_{10} = (10101)_2$
- $(0.65625)_{10} = (0.10101)_2$

- Octal (0,1,2,3,4,5,6,7)
  $(10101)_2 = (010101)_2 = (25)_8$
  $(0.10101)_2 = (0.101010)_2 = (0.52)_8$

- Hexadecimal (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F)
  $(10101)_2 = (00010101)_2 = (15)_{16}$
  $(0.10101)_2 = (0.10101000)_2 = (0.A8)_{16}$

- Useful numbers
  $(10000000000)_2 = (1024)_{10}$ (about 1K)

LOW

code and static data
objects (heap)
↓
↑
local variables (stack)

HIGH

MEMORY ALLOCATION

If you prefer the other direction

writable; not executable	**Stack**	Managed "automatically" (by compiler)
writable; not executable	**Dynamic Data (Heap)**	Managed by programmer
writable; not executable	**Static Data**	Initialized when process starts
Read-only; not executable	**Literals**	Initialized when process starts
Read-only; executable	**Instructions**	Initialized when process starts

Parameter	STACK	HEAP
Basic	Memory is allocated in a contiguous block.	Memory is allocated in any random order.
Allocation and De-allocation	Automatic by compiler instructions.	Manual by the programmer.
Cost	Less	More
Implementation	Easy	Hard
Access time	Faster	Slower
Main Issue	Shortage of memory	Memory fragmentation
Locality of reference	Excellent	Adequate
Safety	Thread safe, data stored can only be accessed by owner	Not Thread safe, data stored visible to all threads
Flexibility	Fixed-size	Resizing is possible
Data type structure	Linear	Hierarchical

STACK VS HEAP

Will discuss with more details later.

MEMORY ALLOCATION TO VARIABLES

- One region of memory is reserved for **static data**.

  - never created or destroyed as program runs, such as named constants.

- When a new object is created, Java allocates space from **heap**.

- When a method is called, Java allocates a new block of memory called a stack frame to hold its local variables.

- When a method returns, its stack frame is erased. Stack frames come from **stack**.

- Java identifies an object by its address in memory. That address is called a reference.

- Eg., when Java executes

  Rational a = new Rational(1, 2);

  it allocates heap space for the new Rational object. For this example, imagine that the object is allocated at address 1000.

- The local variable a is allocated in the current stack frame and is assigned the value (address), which identifies the object.
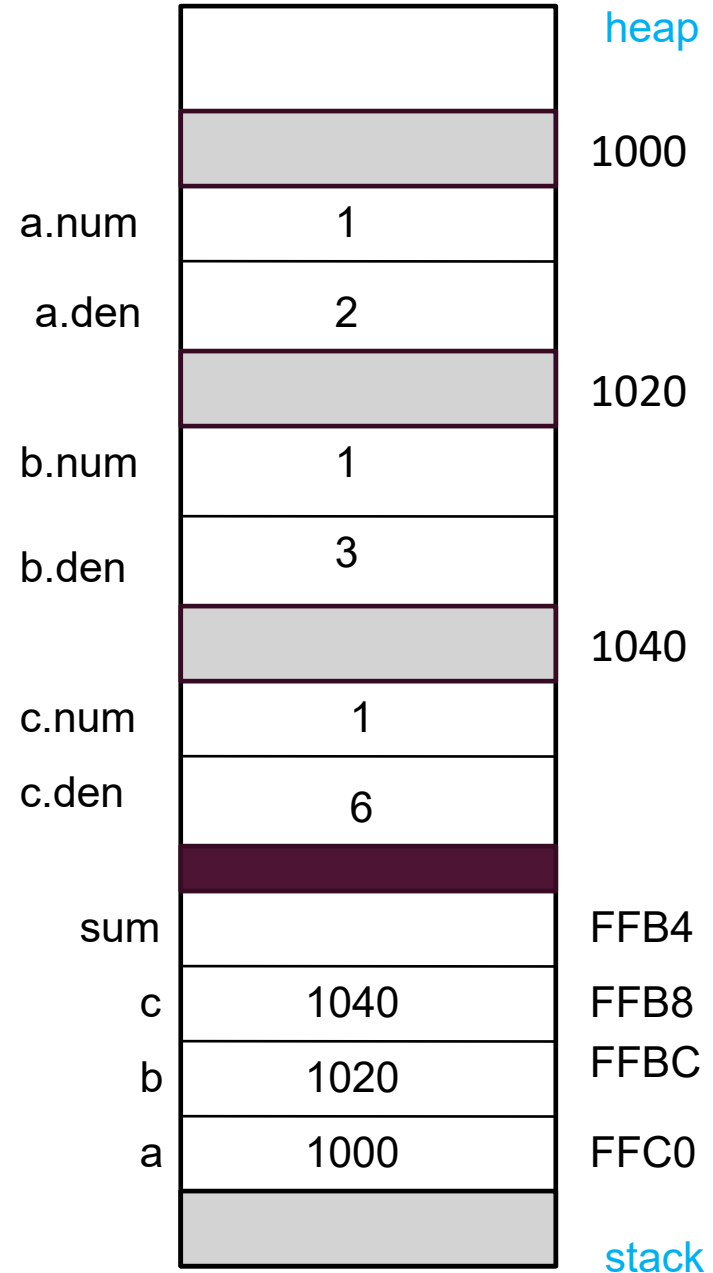
OBJECT REFERENCES

```java
public class Rational
{
    /** Creates a new Rational initialized to zero */
    public Rational() {
        this(0);
    }
    /**
     * Creates a new Rational from the integer argument.
     * @param n The initial value
     */
    public Rational(int n) {
        this(n, 1);
    }
    /**
     * Creates a new Rational with the value x / y.
     * @param x The numerator of the rational number
     * @param y The denominator of the rational number
     */
    public Rational(int x, int y) {
        int g = gcd(Math.abs(x), Math.abs(y));
        num = x / g;
        den = Math.abs(y) / g;
        if (y < 0) num = -num;
    }
    /**
     * Adds the rational number r to this one and returns the sum.
     * @param r The rational number to be added
     * @return The sum of the current number and r
     */
    public Rational add(Rational r) {
        return new Rational(this.num * r.den + r.num * this.den,
                            this.den * r.den);
    }
    /**
     * Subtracts the rational number r from this one and returns
     * the difference.
     * @param r The rational number to be subtracted
     * @return The result of subtracting r from the current number
     */
    public Rational subtract(Rational r) {
        return new Rational(this.num * r.den - r.num * this.den,
                            this.den * r.den);
    }

    /**
     * Multiplies this number by the rational number r.
     * @param r The rational number used as a multiplier
     * @return The result of multiplying the current number by r
     */
    public Rational multiply(Rational r) {
        return new Rational(this.num * r.num, this.den * r.den);
    }
    /**
     * Divides this number by the nonzero rational number r.
     * @param r The nonzero rational number used as a divisor
     * @return The result of dividing the current number by r
     */
    public Rational divide(Rational r) {
        return new Rational(this.num * r.den, this.den * r.num);
    }
    /**
     * Creates a string representation of this rational number.
     * @return The string representation of this rational number
     */
    public String toString() {
        if (den == 1) {
            return "" + num;
        } else {
            return num + "/" + den;
        }
    }
    /**
     * Calculates the greatest common divisor using Euclid's algorithm.
     * @param First integer
     * @param Second integer
     * @return The greatest common divisor of x and y
     */
    private int gcd(int x, int y) {
        int r = x % y;
        while (r != 0) {
            x = y;
            y = r;
            r = x % y;
        }
        return y;
    }
/** Private instance variables */
    private int num;        /* The numerator of this Rational */
    private int den;        /* The denominator of this Rational */
} /* class Rational */
```
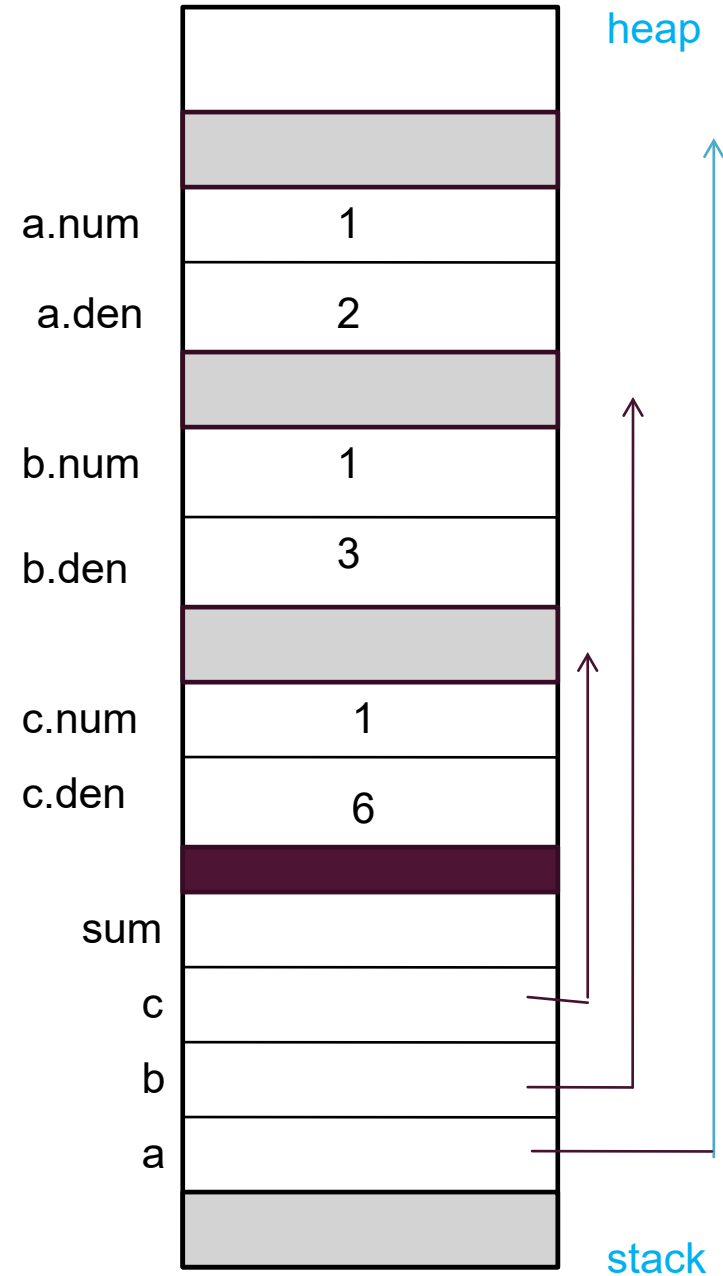
Address Model

```
public void run() {
    Rational  a = new Rational(1, 2);
    Rational  b = new Rational(1, 3);
    Rational  c = new Rational(1, 6);
    Rational  sum = (a.add(b)).add(c);
}
```

		heap
		1000
a.num	1	
a.den	2	
		1020
b.num	1	
b.den	3	
		1040
c.num	1	
c.den	6	
sum		FFB4
c	1040	FFB8
b	1020	FFBC
a	1000	FFC0
		stack

Pointer Model

```
public void run() {
    Rational  a = new Rational(1, 2);
    Rational  b = new Rational(1, 3);
    Rational  c = new Rational(1, 6);
    Rational  sum = (a.add(b)).add(c);
}
```

heap

a.num	1
a.den	2
b.num	1
b.den	3
c.num	1
c.den	6
sum	
c	
b	
a	

stack

```java
public void run() {
    Rational  a = new Rational(1, 2);
    Rational  b = new Rational(1, 3);
    Rational  c = new Rational(1, 6);
    Rational  sum = (a.add(b)).add(c);
}

public  Rational  add(Rational  r) {
    return new Rational(this.num*r.den
                    + r.num*this.den,
                this.den*r.den);
}
```

heap	
	1000
a.num	1
a.den	2
	1020
b.num	1
b.den	3
	1040
c.num	1
c.den	6

r	1020	FFA8
this	1000	FFAC
sum		FFB4
c	1040	FFB8
b	1020	FFBC
a	1000	FFC0
		stack

```
public void run() {
    Rational  a = new Rational(1, 2);
    Rational  b = new Rational(1, 3);
    Rational  c = new Rational(1, 6);
    Rational  sum = (a.add(b)).add(c);
}

public  Rational  add(Rational  r) {
    return new Rational(this.num*r.den
            + r.num*this.den,
        this.den*r.den);
}
```

heap

	1000
a.num	1
a.den	2
	1020
b.num	1
b.den	3
	1040
c.num	1
c.den	6
	1060
(a.add(b)).num	5
(a.add(b)).den	6

sum		FFB4
c	1040	FFB8
b	1020	FFBC
a	1000	FFC0
		stack

```
public void run() {
    Rational  a = new Rational(1, 2);
    Rational  b = new Rational(1, 3);
    Rational  c = new Rational(1, 6);
    Rational  sum = (a.add(b)).add(c);
}

public  Rational  add(Rational  r) {
    return new Rational(this.num*r.den
             + r.num*this.den,
           this.den*r.den);
}
```

heap

| a.num | 1 |
| a.den | 2 |

1000

| b.num | 1 |
| b.den | 3 |

1020

| c.num | 1 |
| c.den | 6 |

1040

| (a.add(b)).num | 5 |
| (a.add(b)).den | 6 |

1060

| (a.add(b)).add(c).num | 1 |
| (a.add(b)).add(c).den | 1 |

1080

r	1040	FFA8
this	1060	FFAC
sum		FFB4
c	1040	FFB8
b	1020	FFBC
a	1000	FFC0
		stack

```
public void run() {
    Rational  a = new Rational(1, 2);
    Rational  b = new Rational(1, 3);
    Rational  c = new Rational(1, 6);
    Rational  sum = (a.add(b)).add(c);
}

public  Rational  add(Rational  r) {
    return new Rational(this.num*r.den
                + r.num*this.den,
             this.den*r.den);
}
```

heap

	1000
a.num	1
a.den	2
	1020
b.num	1
b.den	3
	1040
c.num	1
c.den	6
	1060
(a.add(b)).num	5
(a.add(b)).den	6
	1080
(a.add(b)).add(c).num	1
(a.add(b)).add(c).den	1

sum	1080	FFB4
c	1040	FFB8
b	1020	FFBC
a	1000	FFC0
		stack

```java
public void run() {
    Rational  a = new Rational(1, 2);
    Rational  b = new Rational(1, 3);
    Rational  c = new Rational(1, 6);
    Rational  sum = (a.add(b)).add(c);
}

public  Rational  add(Rational  r) {
    return new Rational(this.num*r.den
                + r.num*this.den,
              this.den*r.den);
}
```

heap

	1000
a.num	1
a.den	2
	1020
b.num	1
b.den	3
	1040
c.num	1
c.den	6
	1060
(a.add(b)).num	5
(a.add(b)).den	6
	1080
(a.add(b)).add(c).num	1
(a.add(b)).add(c).den	1

sum	1080	FFB4
c	1040	FFB8
b	1020	FFBC
a	1000	FFC0
		stack

GARBAGE COLLECTION

- In the example, the object a.add(b) was created in the intermediate step but not referenced by the final stack. It is now **garbage**.

- When memory is running short, Java does garbage collection
  - Mark the objects referenced by variables on stack or in static storage.
  - Sweep all objects in the heap, reclaim unmarked objects (garbage).

- This process is called **garbage collection**.

EXERCISE: STACK-HEAP DIAGRAM

```
public class Point {                    public class Line {
    public Point(int x, int y) {            public Line(Point p1, Point p2) {
        cx = x;                                 start = p1;
        cy = y;                                 finish = p2;
    }                                       }

    private int cx;                         private Point start;
    private int cy;                         private Point finish;
}                                       }

        public void run() {
            Point p1 = new Point(0, 0);
            Point p2 = new Point(200, 200);
            Line line = new Line(p1, p2);
        }
```

Draw a heap-stack diagram (pointer model) showing the state of memory just
before the run() method returns.

PRIMITIVE TYPE VERSUS OBJECTS

- Primitive type

```
public void run() {
    int  x = 17;
    increment(x);
    println("x = " + x);
}

private void increment(int  n)  {
    n++;
    println("n = " + n);
}

Output
n = 18
n = 17
```

When you pass an argument of a primitive type to a method, Java copies the value of the argument into the parameter variable. As a result, changes to the parameter variable have no effect on the argument.

Passing x of primitive type int, a value

increment(x);
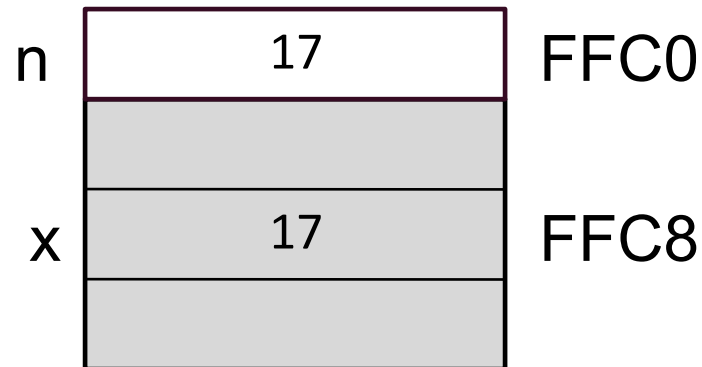
x (a value) is copied into n

EMBEDDEDINTEGER CLASS

```java
public class EnbeddedInteger {

    public EmbeddedInteger(int n) {
        value = n;
    }

    public void setValue(int n) {
        value = n;
    }

    public  int  getValue() {
        return value;
    }

    public  String  toString() {
        return "" + value;
    }

    private int value;
}
```

- Object

```
public  void  run() {
      EmbeddedInteger  x = new EmbeddedInteger(17);
      increment(x);
      println("x = " + x);
}

private void increment(EmbeddedInteger n) {
    n.setValue(n.getValue()  + 1);
    println("n = " + n);
}
```

Output
n = 18
n = 18

PRIMITIVE TYPES VS OBJECTS

- When you pass an object as an argument, there seems to be some form of sharing going on. However, any changes that you make to the instance variables *inside* an object have a permanent effect on the object.

- Stack-heap diagrams make the reason for this seeming asymmetry clear. When you pass an object to a method, Java copies the *reference*, not the object itself.
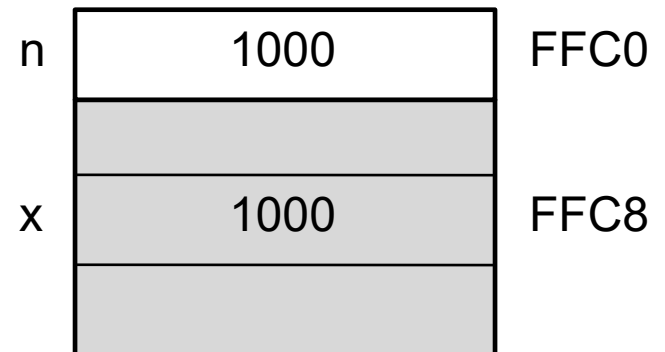
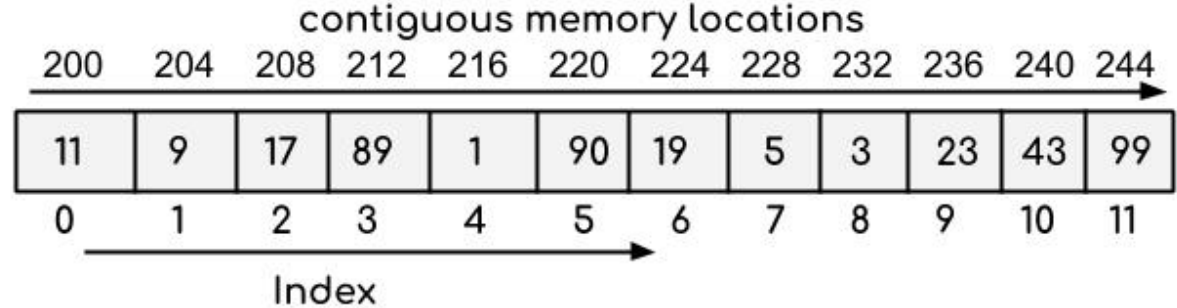# Passing object x, a reference (address)

increment(x)

heap

stack

n	1000	FFC0
x	1000	FFC8

1000

x.value	17
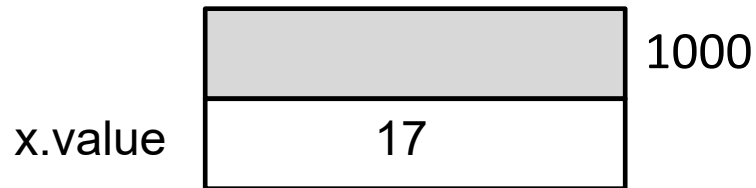
x (a reference to an object) is copied into n
x and n share the same object

OUTLINE

- Basics in Computer Memory

- Arrays

- Implementation

- Examples

contiguous memory locations

200	204	208	212	216	220	224	228	232	236	240	244
11	9	17	89	1	90	19	5	3	23	43	99
0	1	2	3	4	5	6	7	8	9	10	11

Index

ARRAY

- Array: An ordered collection of values
  - Ordered and fixed length
  - Homogeneous: Each value in the array is of the same type

- The individual values in an array are called **elements**.

- The number of elements is called the **length** of the array

- Each element is identified by its position in the array, which is called **index**.
  - In Java, the index numbers begin with 0.

ILLUSTRATION FROM WIKIPEDIA

- Array: a data structure consisting of a collection of elements (values or variables)

  - Each element is identified by at least one array index or key.

  - The memory position of each element can be computed from its index tuple.

  - The simplest type of data structure is a linear array, also called one-dimensional array.

- Example: an array of 10 32-bit (4-byte) integer variables, with indices 0 through 9,

  - May be stored as 10 words at memory addresses 2000, 2004, 2008, ..., 2036, (in hexadecimal: 0x7D0, 0x7D4, 0x7D8, ..., 0x7F4)

  - The element with index i has the address 2000 + (i × 4).

ARRAY DECLARATION

- An array is characterized by
  - Element type
  - Length

    type[ ] identifier = new type[length];

- Default values in initialization
  - numerics   0
  - boolean   false
  - objects   null

AN ARRAY OF OBJECTS

Elements of an array can be objects of any Java class.

Example: An array of 5 instances of the student class

Student [ ]

topStudents = new Student[5];

DEFINING LENGTH

- Use named constant to declare the length of an array.

private static final int N_JUDGES = 5;

double[ ] scores = new double[N_JUDGES];

- Or read the length of an array from the user.

SELECTING ELEMENTS

- Identifying an element

array[index]

- Index can be an expression

- Cycling through array elements

```
for (int i = 0; i < array.length; i++) {
    operations involving the ith element;
}
```

What will happen in stack & heap: int[ ] numbers = new int[10];

HUMAN-READABLE INDEX VALUES

- Starting index numbering at 0 can be confusing.
    - Sometimes, it makes sense to work with index that begins with 1.

- Two standard ways:
    - Use Java's index number internally and then add one when presenting to the user.
    - Use index values beginning at 1 and ignore the first (0) element in each array.

```java
/**
 * The student class is basic class.
 */
public class Student
{
    /**
     * @param name The student's name
     * @param id student's id
     */
    public Student(String name, int id) {
        studentName = name;
        studentId = id;
    }
    /**
     * Gets name of student
     * @return the name of student
     */
    public String getName() {
        return studentName;
    }
    /**
     * Gets id of student
     * @return the id of student
     */
    public int getId() {
        return studentId;
    }
    /**
     * sets the number of credits enarned.
     * @param credits The new number of credits earned
     */
    public void setCredits(double credits) {
        creditsEarned = credits;
    }
    /**
     * Gets the number of credits earned.
     * @return The number of credits this student has earned
     */
    public double getCredits() {
        return creditsEarned;
    }
    /**
     * Sets whether the student is paid up.
     * @param flag The value true or false indicating paid-up status
     */
    public void setPaidUp(boolean flag) {
        paidUp = flag;
    }
    /**
     * Returns whether the sutdent is paid up.
     * @return Whether the student is paid up.
     */
    public boolean isPaidUp() {
        return paidUp;
    }
    /**
     * Creates a string identifying this student.
     * @return The string used to display this student.
     */
    public String toString() {
        return studentName + " (#" + studentId + ")";
    }

    /* public constants */
    public static final double CREDITS_TO_GRADUATE = 32.0;
    /* Private instance variables */
    private String studentName;      /* The student's name */
    private int studentId;           /* The student's id */
    private double creditsEarned;    /* The number of credits earned */
    private boolean paidUp;          /* Whether student is paid up */
}
```

INTERNAL REPRESENTATION OF ARRAYS

Student[ ] topStudents = new Student[2];

topStudents[0] = new Student("Abcd", 314159);

		1000
		1004
length	2	1008
topStudents[0]	null	100C
topStudents[1]	null	1010

heap

topStudents	1000	FFB8
		FFBC
		FFC0

stack
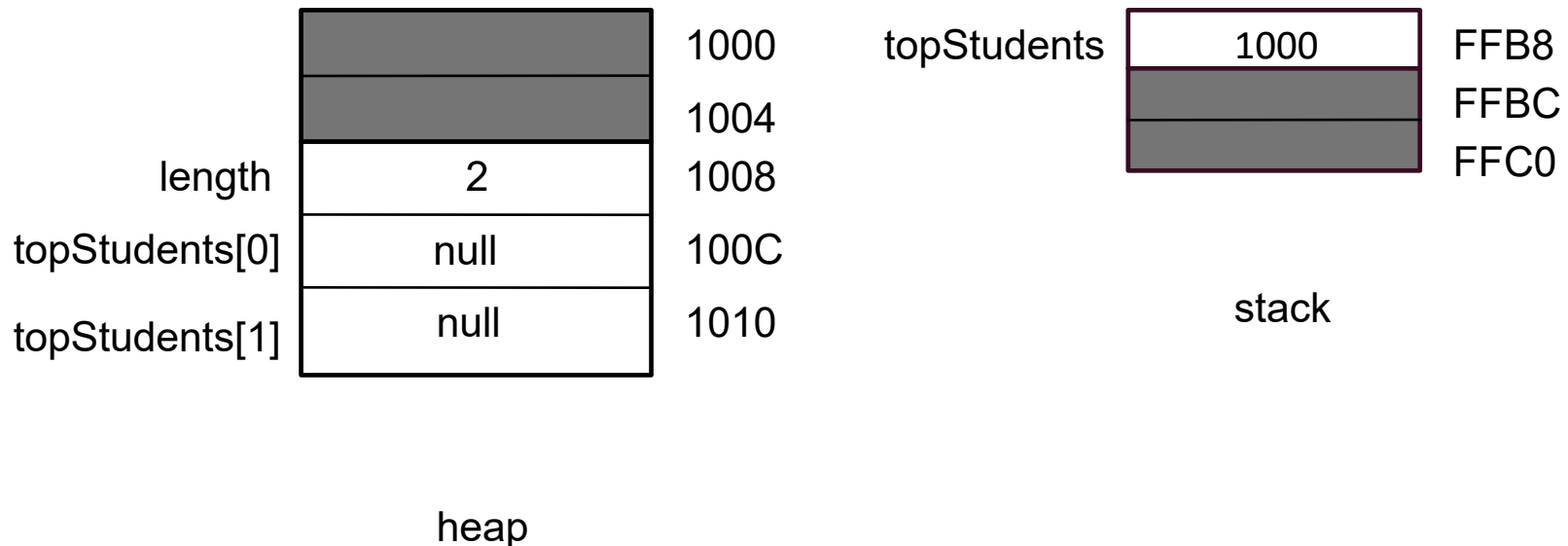
		1000
		1004
length	2	1008
topStudents[0]	1028	100C
topStudents[1]	null	1010
		1014
		1018
length	4	101C
	A | b	1020
	c | d	1024
		1028
		102C
studentName	1014	1030
studentID	314159	1034
creditsEarned	0.0	1038
		103C
paidUp	false	1040

Student[] topStudents = new Student[2];

topStudents[0] = new Student("Abcd", 314159);

topStudents	1000	FFB8
		FFBC
		FFC0

PASSING ARRAYS AS PARAMETERS

- Recall: Passing objects (references) versus primitive type (values) as parameters.

- Java defines all arrays as objects, implying that the elements of an array are shared between the callee and the caller.

  swapElements(array[i], array[n – i – 1]) (wrong)

  swapElements(array, i, n – i – 1)

```java
private void swapElements(int[] array, int p1, int p2) {
        int  tmp = array[p1];
        array[p1] = array[p2];
        array[p2] = tmp;
}
```

- Every array in Java has a length field.

```java
private void reverseArray(int[] array)  {
        for (int  i = 0; i < array.length / 2; i++) {
                swapElements(array, i, array.length – i – 1);
        }
}
```

USING ARRAYS

- Example: Letter frequency table

Array: letterCounts[ ]

index: distance from 'A'

index = Character.toUpperCase(ch) – 'A'

letterCounts[0] is the count for 'A' or 'a'

A convenient way of initializing an array:

```
int[ ] digits = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

private static final String[ ] US_CITIES_OVER_ONE_MILLION = {
    "New York",
    "Los Angeles",
    "Chicago",
    "Huston",
    "Philadelphia",
    "Phoenix",
    "San Diego",
    "San Antonio",
    "Dallas",
}
```

TWO-DIMENSIONAL ARRAYS

- Each element of an array is an array (of the same dimension)

  int[][] A = new int[3][2];

- An array of three arrays of dimension two

  A[0][0]   A[0][1]

  A[1][0]   A[1][1]

  A[2][0]   A[2][0]

Memory allocation (row orientation)

A[0][0]

A[0][1]

A[1][0]

A[1][1]

A[2][0]

A[2][1]

INITIALIZING A TWO-DIMENSIONAL ARRAY

```
static int A[3][2] = {
    {1, 4},
    {2, 5},
    {3, 6}
};
```

A 3-by-2 matrix

THE `ARRAYLIST` CLASS

- The `java.util` package includes a class called `ArrayList`
  - Provide standard array behaviors along with other useful operations.

- `ArrayList` is a Java class rather than a special form in the language. All operations on `ArrayList`s are indicated using method calls.
  - Create a new `ArrayList` by calling the `ArrayList` constructor.
  - Get the number of elements by calling the `size` method.
  - Use the `get` and `set` methods to select individual elements.

https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

OUTLINE

- Basics in Computer Memory

- Arrays

- Implementation

- Examples

IMPLEMENTATIONS

- Compile and Run the following programs in server

  - Project 2.1   maxArray.java

  - Project 2.2   remaxArray.java

  - Project 2.3   sortArray.java

  - Project 2.4   binArray.java

  - Project 2.5   mergeArrays.java

  - Project 2.6   noDups.java

OUTLINE

- Basics in Computer Memory

- Arrays

- Implementation

- Examples

NUMBER ARRAY

- Each element of the array is a number: int, float, double, …

```java
public class Main
{
    public static void main(String[] args)
    {

        // create an array
        int[] age = {12, 4, 5, 2, 5};

        // access each array elements
        System.out.println("Accessing Elements of Array:");
        System.out.println("First Element: " + age[0]);
        System.out.println("Second Element: " + age[1]);
        System.out.println("Third Element: " + age[2]);
        System.out.println("Fourth Element: " + age[3]);
        System.out.println("Fifth Element: " + age[4]);
    }
}
```

Output

```
Accessing Elements of Array:
First Element: 12
Second Element: 4
Third Element: 5
Fourth Element: 2
Fifth Element: 5
```

```java
public class Main
{
    public static void main(String[] args)
    {

        // create an array
        int[] age = {12, 4, 5};

        // loop through the array
        // using for loop
        System.out.println("Using for Loop:");
        for(int i = 0; i < age.length; i++)
        {
            System.out.println(age[i]);
        }
    }
}
```

Output

```
Using for Loop:
12
4
5
```

```java
public class Main
{
    public static void main(String[] args)
    {
        // create an array
        int[] age = {12, 4, 5};

        // loop through the array
        // using for loop
        System.out.println("Using for-each Loop:");
        for(int a : age)
        {
            System.out.println(a);
        }
    }
}
```

Output

```
Using for-each Loop:
12
4
5
```

```java
public class Main
{
    public static void main(String[] args)
    {

        int[] numbers = {2, -9, 0, 5, 12, -25, 22, 9, 8, 12};
        int sum = 0;
        Double average;

        // access all elements using for each loop
        // add each element in sum
        for (int number : numbers)
        {
            sum += number;
        }

        // get the total number of elements
        int arrayLength = numbers.length;

        // calculate the average
        // convert the average from int to double
        average =  ((double)sum / (double)arrayLength);

        System.out.println("Sum = " + sum);
        System.out.println("Average = " + average);
    }
}
```

Output:

```
Sum = 36
Average = 3.6
```

```java
public class MultidimensionalArray
{
    public static void main(String[] args)
    {

        // create a 2d array
        int[][] a =
        {
            {1, 2, 3},
            {4, 5, 6, 9},
            {7},
        };

        // calculate the length of each row
        System.out.println("Length of row 1: " + a[0].length);
        System.out.println("Length of row 2: " + a[1].length);
        System.out.println("Length of row 3: " + a[2].length);
    }
}
```

Output:

```
Length of row 1: 3
Length of row 2: 4
Length of row 3: 1
```

```java
public class MultidimensionalArray {
    public static void main(String[] args)
    {

        int[][] a =
        {
            {1, -2, 3},
            {-4, -5, 6, 9},
            {7},
        };

        for (int i = 0; i < a.length; ++i)
        {
            for(int j = 0; j < a[i].length; ++j)
            {
                System.out.println(a[i][j]);
            }
        }
    }
}
```

Output:

```
1
-2
3
-4
-5
6
9
7
```

```java
public class MultidimensionalArray
{
    public static void main(String[] args)
    {

        // create a 2d array
        int[][] a =
        {
            {1, -2, 3},
            {-4, -5, 6, 9},
            {7},
        };

        // first for...each loop access the individual array
        // inside the 2d array
        for (int[] innerArray : a)
        {
            // second for...each loop access each element inside the row
            for(int data : innerArray)
            {
                System.out.println(data);
            }
        }
    }
}
```

Output:

```
1
-2
3
-4
-5
6
9
7
```

```java
public class ThreeArray
{
    public static void main(String[] args)
    {

        // create a 3d array
        int[][][] test =
        {
            {
                {1, -2, 3},
                {2, 3, 4}
            },
            {
                {-4, -5, 6, 9},
                {1},
                {2, 3}
            }
        };

        // for..each loop to iterate through elements of 3d array
        for (int[][] array2D : test)
        {
            for (int[] array1D : array2D)
            {
                for(int item : array1D)
                {
                    System.out.println(item);
                }
            }
        }
    }
}
```

Output:

```
1
-2
3
2
3
4
-4
-5
6
9
1
2
3
```

```java
public class ConvertDoubleArrayToFloatArray
{
    public static void main(String args[])
    {
        double[] doubleArray = {3.5, 5.0, 7.5, 11.55};
        float[] floatArray = new float[doubleArray.length];
        for (int i = 0 ; i < doubleArray.length; i++)
        {
            floatArray[i] = (float) doubleArray[i];
        }
        for(int i = 0; i < floatArray.length; i++)
        {
            System.out.println("Element at Index " + i + " is : " + floatArray[i]);
        }
    }
}
```

What is the output?

```java
public class ConvertFloatArrayToDoubleArray
{
    public static void main(String args[])
    {
        float[] floatArray = {2.0f, 1.5f, 8.45f, 116.77f};
        double[] doubleArray = new double[floatArray.length];
        for (int i = 0 ; i < floatArray.length; i++)
        {
            doubleArray[i] = (double) floatArray[i];
        }
        for(int i = 0; i < doubleArray.length; i++)
        {
            System.out.println("Element at Index " + i + " is : " + doubleArray[i]);
        }
    }
}
```

What is the output?

STRING ARRAY

- String Array is an array holding a fixed number of strings or string values.
    - One structure commonly used in Java.
    - Even the argument of the 'main' function in Java is a String Array.

- String array is an array of objects.
    - String is an object.

```java
public class Main
{
    public static void main(String[] args)
    {

        String[] myarray;          //declaration of string array without size
        String[] strArray = new String[5];  //declaration with size

        //System.out.println(myarray[0]);   //variable myarray might not have been initialized
        //display elements of second array
        System.out.print(strArray[0] + " " + strArray[1] + " " + strArray[2] + " " +
                        strArray[3] + " " + strArray[4]);
    }
}
```

```
null  null  null  null  null
```

```java
public class Main
{
    public static void main(String[] args)
    {
        //declare and initialize a string array
        String[] numArray = {"one", "two", "three", "four", "five"};
        int len = numArray.length;   //get the length of array
        //display the length
        System.out.println("Length of numArray{\"one\",\"two\", \"three\", \"four\", \"five\"}:" + len);
    }
}
```

```
Length of numArray{"one","two", "three", "four", "five"}:5
```

```java
1  public class Main
2  {
3      public static void main(String[] args)
4      {
5          //declare and initialize a string array
6          String[] numArray = {"one", "two", "three", "four", "five"};
7          System.out.println("String Array elements displayed using for loop:");
8          // for loop to iterate over the string array
9          for(int i = 0; i < numArray.length; i++)
10             System.out.print(numArray[i] + " ");
11
12         System.out.println("\n");
13
14         System.out.println("String Array elements displayed using enhanced for loop:");
15         //enhanced for loop to iterate over the string array
16         for(String val : numArray)
17             System.out.print(val + " ");
18     }
19 }
```

```
String Array elements displayed using for loop:

one two three four five



String Array elements displayed using enhanced for loop:

one two three four five
```

```java
import java.util.*;
public class Main
{
    public static void main(String[] args)
    {
        String[] colorsArray = new String[5];
        // initial array values
        colorsArray[0] = "Red";
        colorsArray[1] = "Green";
        colorsArray[2] = "Blue";
        System.out.println("Original Array:" + Arrays.toString(colorsArray));
        intnumberOfItems = 3;

        // try to add new value at the end of the array
        String newItem = "Yellow";
        colorsArray[numberOfItems++] = newItem;
        System.out.println("Array after adding one element:" +
                           Arrays.toString(colorsArray));
    }
}
```

```
Original Array:[Red, Green, Blue, null, null]

Array after adding one element:[Red, Green, Blue, Yellow, null]
```

```java
import java.util.*;

public class Main
{
    public static void main(String[] args)
    {
        //original array
        String[] colorsArray = {"Red", "Green", "Blue" };
        System.out.println("Original Array: " + Arrays.toString(colorsArray));

        //length of original array
        int orig_length = colorsArray.length;
        //new element to be added to string array
        String newElement = "Orange";
        //define new array with length more than the original array
        String[] newArray = new String[ orig_length + 1 ];
        //add all elements of original array to new array
        for (int i = 0; i < colorsArray.length; i++)
        {
            newArray[i] = colorsArray [i];
        }
        //add new element to the end of new array
        newArray[newArray.length - 1] = newElement;
        //make new array as original array and print it
        colorsArray = newArray;
        System.out.println("Array after adding new item: " + Arrays.toString(colorsArray));
    }
}
```

```
Original Array: [Red, Green, Blue]

Array after adding new item: [Red, Green, Blue, Orange]
```

```java
import java.util.*;

class Main
{

    public static void main(String[] args)
    {

        String[] colors = {"red", "green", "blue", "white", "orange"};
        System.out.println("Original array: " + Arrays.toString(colors));
        Arrays.sort(colors);
        System.out.println("Sorted array: " + Arrays.toString(colors));
    }
}
```

```
Original array: [red, green, blue, white, orange]

Sorted array: [blue, green, orange, red, white]
```

```java
import java.util.*;
public class Main
{
    public static void main(String[] args)
    {
        String[] strArray = { "Book", "Pencil", "Eraser", "Color", "Pen" };
        boolean found = false;
        int index = 0;
        String searchStr = "Pen";
        for (int i = 0; i < strArray.length; i++)
        {
            if(searchStr.equals(strArray[i]))
            {
                index = i;
                found = true;
                break;
            }
        }
        if(found)
            System.out.println(searchStr + " found at the index " + index);
        else
            System.out.println(searchStr + " not found in the array");

    }
}
```

```
Pen found at the index 4
```

```java
import java.util.*;
public class Main
{
    public static void main(String args[])
    {
        //string array
        String[] str_Array = {"This", "is", "Software", "Testing", "Help"};
        System.out.print("Original string array:");
        //print string array
        for(String val : str_Array)
            System.out.print(val + " ");

        System.out.println("\n");
        //construct a stringbuilder object from given string array
        StringBuilder stringBuilder = new StringBuilder();
        for (int i = 0; i < str_Array.length; i++)
        {
            stringBuilder.append(str_Array[i] + " ");
        }
        //print the string
        System.out.println("String obtained from string array:" + stringBuilder.toString());
    }
}
```

```
Original  string  array:This  is Software  Testing  Help



String  obtained  from  string  array:This  is Software  Testing  Help
```

```java
1    import java.util.*;
2
3    public class Main
4    {
5        public static void main( String[] args )
6        {
7            //string arrya declaration
8            String [] str_Array = {"10", "20", "30", "40", "50"};
9            //print the string array
10           System.out.println("Original String Array:");
11           for(String val : str_Array)
12               System.out.print(val + " ");
13
14           System.out.println("\nThe integer array obtained from string array:");
15           //declare an int array
16           int [] int_Array = new int [str_Array.length];
17           //assign string array values to int array
18           for(int i = 0; i < str_Array.length; i++)
19           {
20               int_Array[i] = Integer.parseInt(str_Array[i]);
21           }
22           //display the int array
23           System.out.println(Arrays.toString(int_Array));
24       }
25   }
```

```
Original String Array:

10 20 30 40 50

The integer array obtained from string array:

[10, 20, 30, 40, 50]
```

```java
1  public class ByteArraySize
2  {
3      public static void main(String args[])
4      {
5          String str = "byte array size example";
6          byte array[] = str.getBytes();
7          System.out.println("Size of byte Array : " + array.length);
8      }
9  }
```

What is the output?

```java
1   import java.util.Arrays;
2   public class StringToByteArray
3   {
4       public static void main(String[] args)
5       {
6           String str = " convert String to byte Array in Java ";
7           byte[] bytearray = str.getBytes();
8           System.out.println(Arrays.toString(bytearray));
9       }
10  }
```

What is the output?

BOOLEAN ARRAY

- Each element of the array is a Boolean value (true, false).

```java
1   import java.util.Arrays;
2   public class BooleanArrayTest
3   {
4       public static void main(String[] args)
5     {
6           Boolean[] boolArray = new Boolean[5]; // initialize a boolean array
7           for(int i = 0; i < boolArray.length; i++)
8         {
9               System.out.println(boolArray[i]);
10
11        }
12          Arrays.fill(boolArray, Boolean.FALSE);
13          // all the values will be false
14          for(int i = 0; i < boolArray.length; i++)
15        {
16              System.out.println(boolArray[i]);
17
18        }
19          Arrays.fill(boolArray, Boolean.TRUE);
20          // all the values will be true
21          for (int i = 0; i < boolArray.length; i++)
22        {
23              System.out.println(boolArray[i]);
24
25        }
26
27    }
28  }
```

What is the output?

SUMMARY

- Understand memory, understand everything!
    - The behavior of arrays is reflected in the change of stacks and queues.

- What's the difference between an array of objects and an array of primitive types?