Chapter 3

Working with Data

Primitive data types & Strings

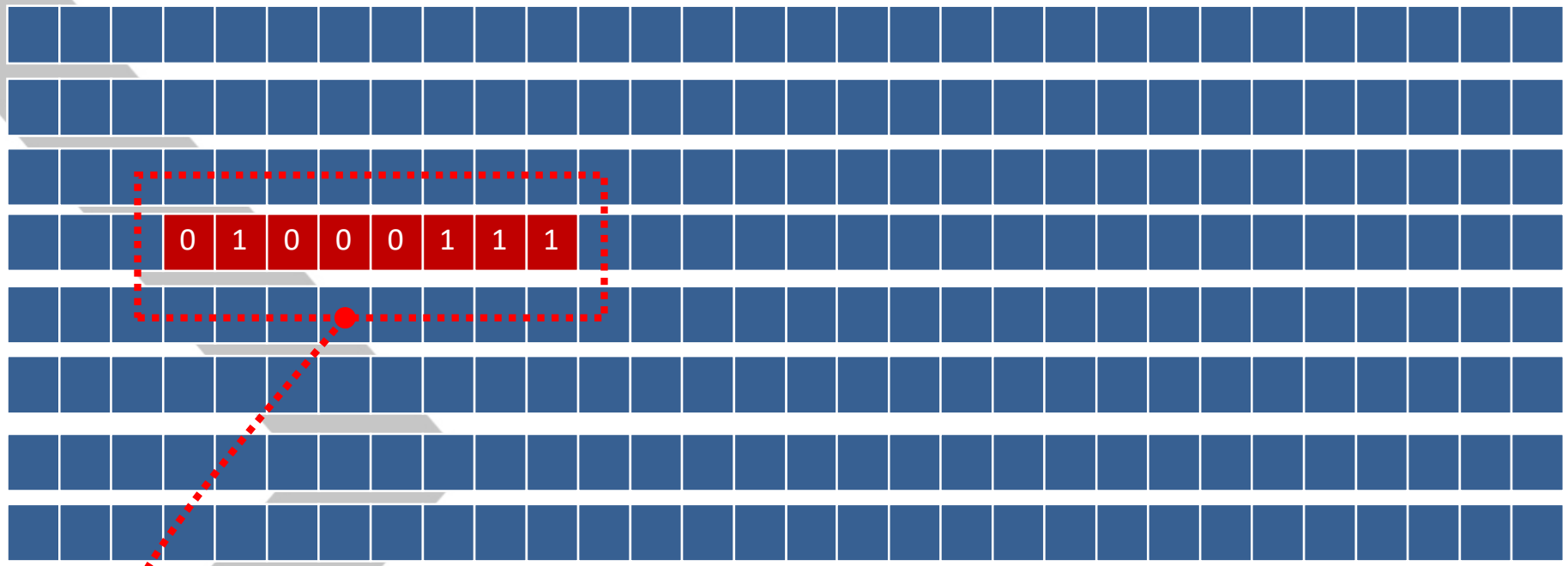
Objectives

Chapter 3

Students should:

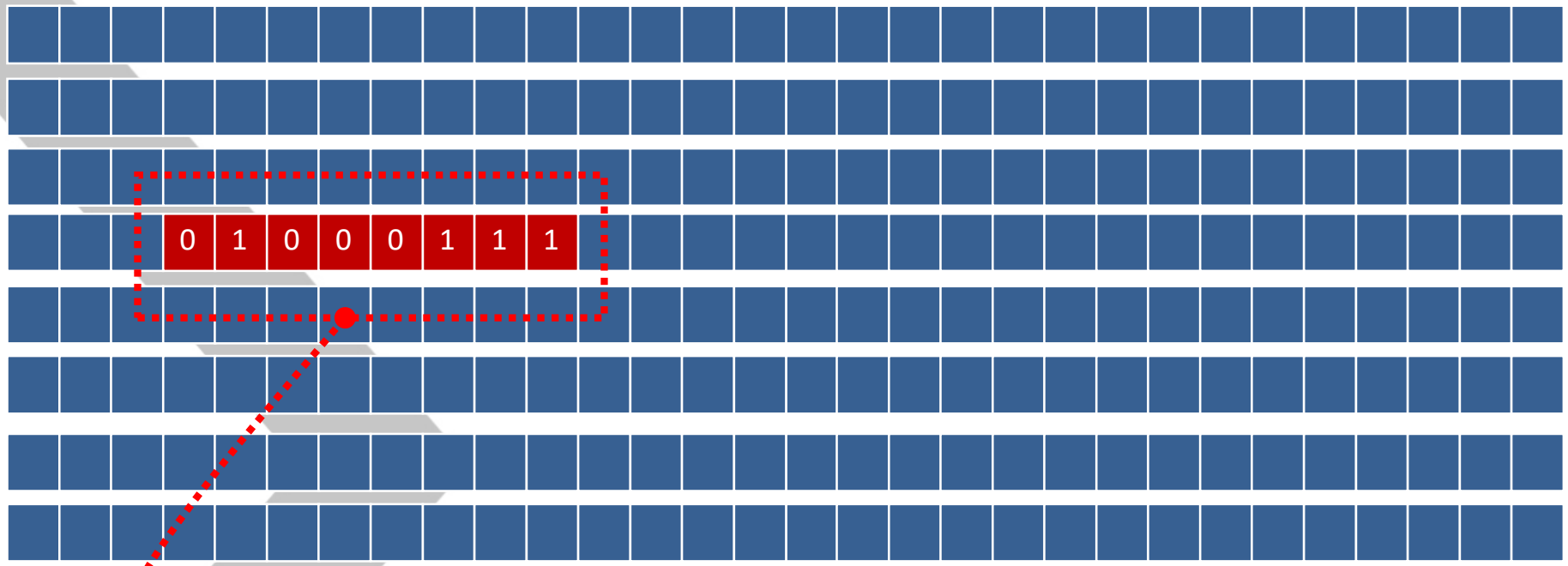
- Be familiar with the *eight primitive data types* and non-primitive data types.
- Understand *memory allocations and value assignments* of variables.
- Be able to use *operators* and some methods to do some computations.

Understanding Data Representation



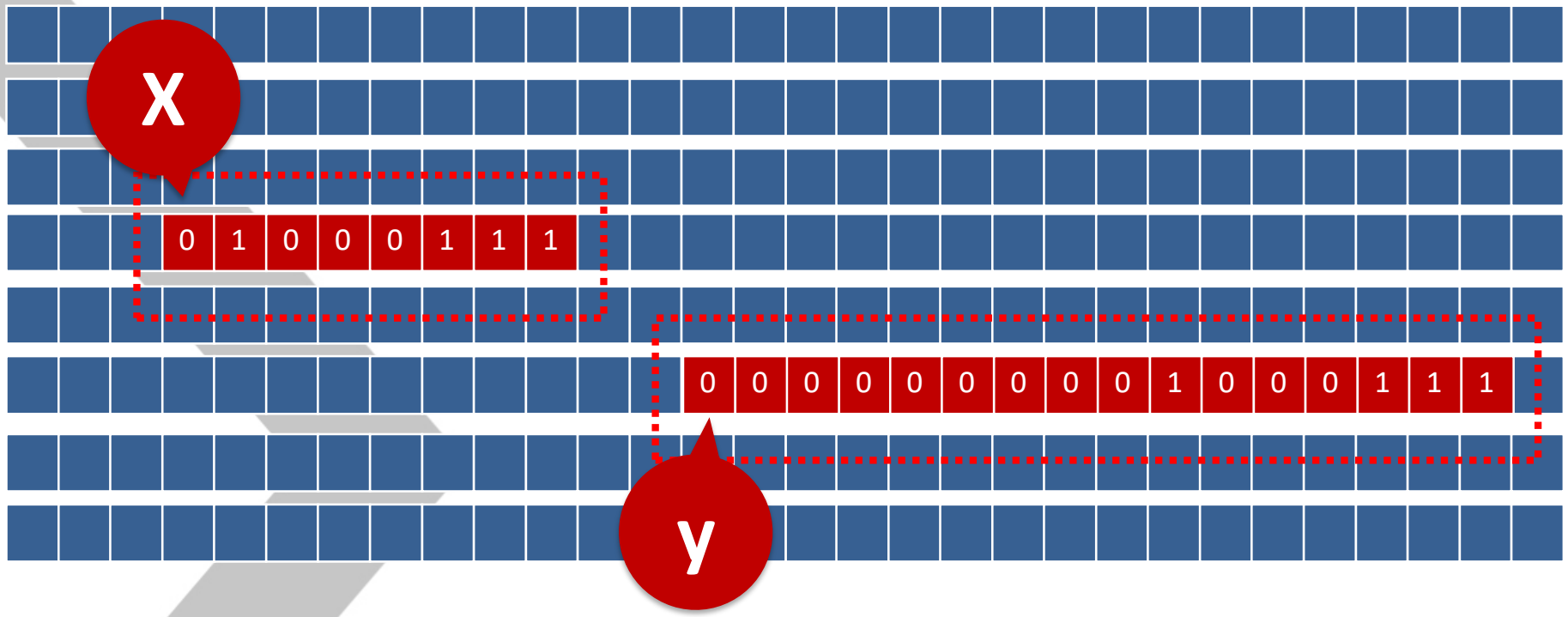
A byte representing the value **71**

Understanding Data Representation



To refer to this value at this location
we use a **variable**

Understanding Data Representation



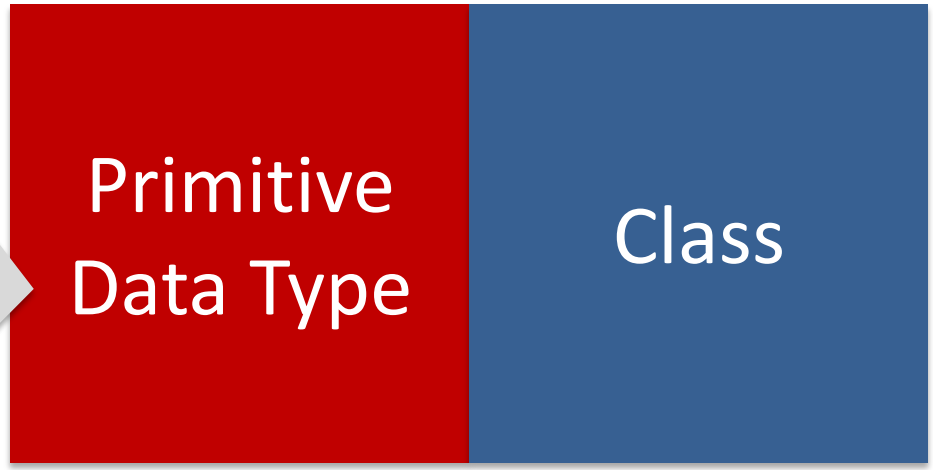
Here, the variable *x contains a value of 71 stored in 1 byte*.
The variable *y contains a value of 71 stored in 2 bytes*.
Both values are of different types.

Different?

Data Types in Java

- Cannot be added/changed
- There are

8 primitive types



- 4 for integer values
- 2 for floating-point values
- 1 for characters
- 1 for boolean (T/F) values

Primitive Data Types in Java

Value type	Primitive data types
Integer (E.g.: 5, -1, 0, etc.)	byte, short, int, long
Floating-point (E.g.: 1.0, 9.75, -3.06, etc.)	float, double
Character (E.g.: 'a', 'ก', '@', '4', etc.)	char
Logical (true/false)	boolean

Ranges of Primitive-type Values

Type	Contains	Size	Range
boolean	true or false	1 bit	true / false
char	Unicode character	16 bits	\u0000 to \uFFFF
byte	Signed integer	8 bits	-128 to 127
short	Signed integer	16 bits	-32768 to 32767
int	Signed integer	32 bits	-2147483648 to 2147483647
long	Signed integer	64 bits	-9223372036854775808 to 9223372036854775807
float	Floating point	32 bits	±1.4E-45 to ±3.4028235E+38
double	Floating point	64 bits	±4.9E-324 to ±1.7976931348623157E+308

Literal

- A **literal** is a notation for representing a fixed value in source code.

Primitive Data Literals

Type	Example
boolean	`true, false`
char	`'A', 'B', '8', '\u0065'`
int	`1, 0, -3, 0b101, 0xAA`
long	`1L, 0L, -3L, 0b101L, 0xAAL`
float	`0.0F, 0f, 2e3F, 2e-3F`
double	`0.0, 2e3, 2e-3, 0.0D, 0.0d`

How to write values of different data types in the source code

Primitive Data Literals

This makes them **keywords**

Type	Example
boolean	true, false
char	'A', 'B', '8', '\u0065'
int	1, 0, -3, 0b101, 0xAA
long	1L, 0L, -3L, 0b101, 0xAAL
float	0.0F, 0f, 2e3F, 2e-3F
double	0.0, 2e3, 2e-3, 0.0D, 0.0d

Primitive Data Literals

Type	Example
boolean	true, false
char	'A', 'B', '8', '\u0065'
int	1, 0, -3, 0b101, 0xAA
long	1L, 0L, -3L, 0b101, 0xAAL
float	0.0F, 0f, 2e3F, 2e-3F
double	0.0, 2e3, 2e-3, 0.0D, 0.0d

Single quoted

To specify the **code** of a character

Primitive Data Literals

Java 7

Type	Example
boolean	true, false
char	'A', 'B', '8', '\u0065'
int	1, 0, -3, 0b101, 0xAA
long	1L, 0L, -3L, 0b101, 0xAAL
float	0.0F, 0f, 2e3f
double	0.0, 2e3, 2e-3, 0.0D, 0.0d

A number with **no decimal points**

The **0b** prefix specifies a **binary** representation

The **0x** prefix specifies a **hexadecimal** representation

Primitive Data Literals

Type	Example
boolean	true, false
char	‘8’, ‘\u0065’
int	1, 0, -3, 0b101, 0xAA
long	1L, 0L, -3L, 0b101, 0xAAL
float	0.0F, 0f, 2e3F, 2e-3F
double	0.0, 2e3, 2e-3, 0.0D, 0.0d

The **L** prefix specifies that the value is of the **long** data type

Primitive Data Literals

Type	Example
boolean	true, false
char	'A', 'B', '8', '\u0065'
int	, 0b101, 0xAA
long	1L, 0L, -3L, 0b101, 0xAAL
float	0.0F, 0f, 2e3F, 2e-3F
double	0.0, 2e3, e-3, 0.0, 0.0d

The **F** prefix specifies that the value is of the **float** data type

Scientific notation (2×10^3)

Scientific notation (2×10^{-3})

Primitive Data Literals

Type	Example
boolean	true, false
char	'A', 'B', '8', '\u0065'
int	1, 0, -3, 0b101, 0xAA
long	1L, 0L, -3L, 0[illegible]
float	0.0F, 0f, [illegible]
double	0.0, 2e3, 2e-3, 0.0D, 0.0d

The **D** prefix specifies that the value is of the **double** data type

A number with **decimal points**

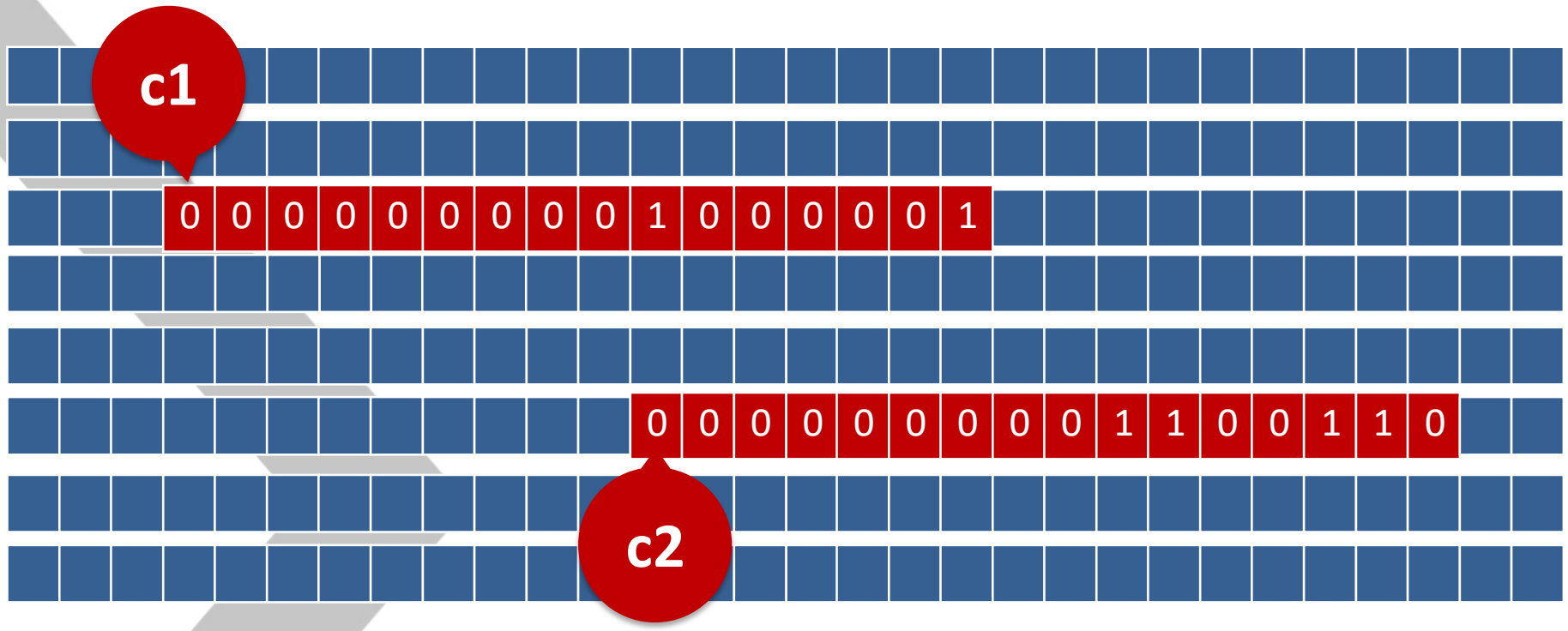
Scientific notation automatically specifies the double data type.

How a `char` is represented

- The underlying representation of a `char` is an integer in the **Unicode** character set coding.
- Characters are encoded in 2 bytes

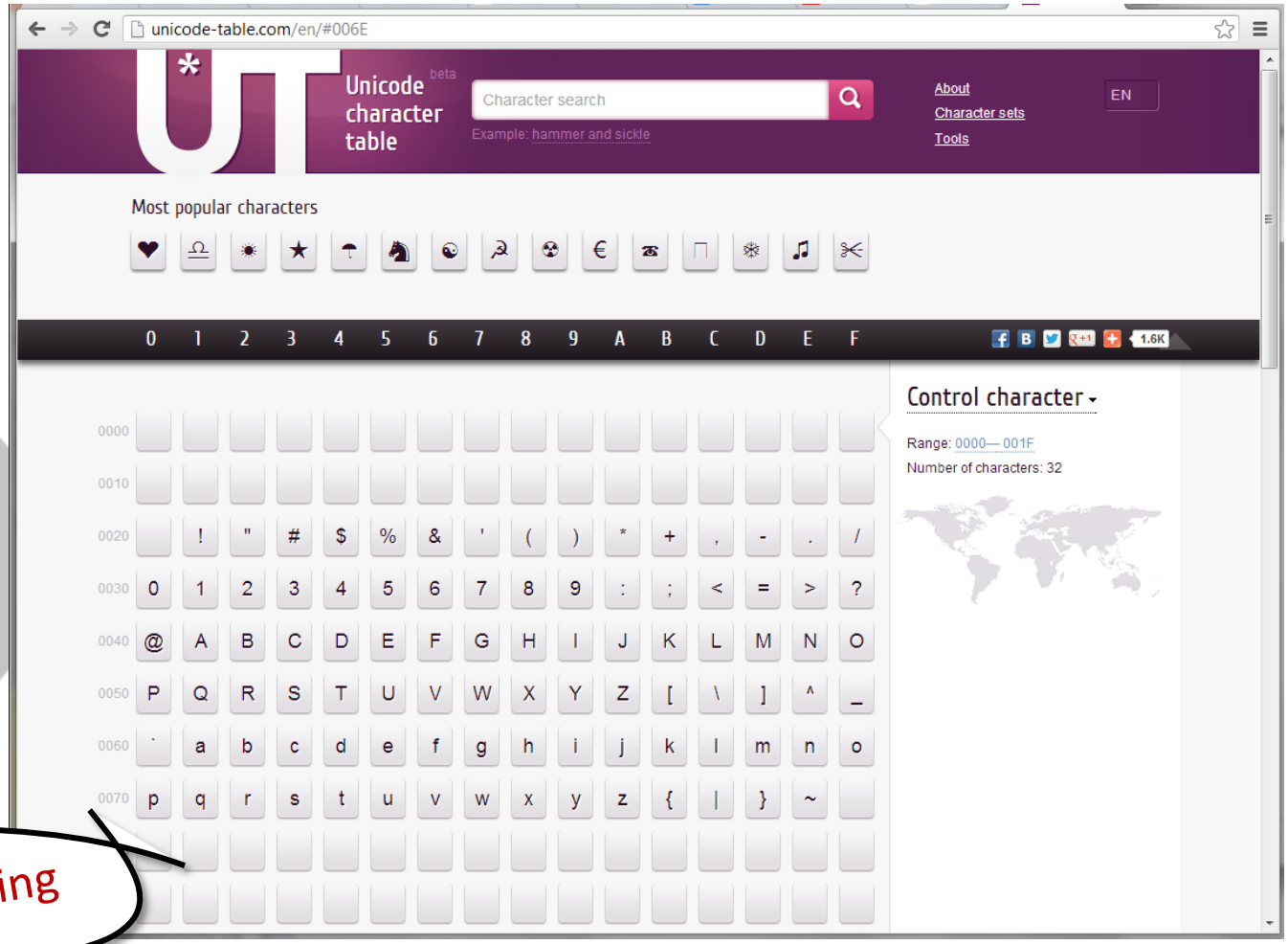
Character	Unicode encoding
'A'	65
'B'	66
'a'	97
'b'	98
'1'	49
'#'	35

How a `char` is represented



The value in c1 is of the type `char`. It is 65 which is the unicode for 'A'.
The value in c2 is of the type `char`. It is 102 which is the unicode for 'f'.

Unicode Table



Observe the ordering

http://unicode-table.com/en/

String

"100" Vs. 100

"Q" Vs. 'q'

- Not a primitive data type.
- It is a class.
  - representing a sequence of one or more characters.
- Double quotes are used to designate the String type.

Declaring Variables

syntax

[Variable data type] [Identifier];

Examples

```
char z;

int j,k,l;

boolean isit;
```

Assigning Values to Variables

syntax

```
[ Variable ] = [ Expression ];
```

- assignment operator (=)
- Variables have types.
- A variable can only store a value that has the same data type as itself, unless the conversion can be done automatically (more on this later).

Assigning Values to Variables

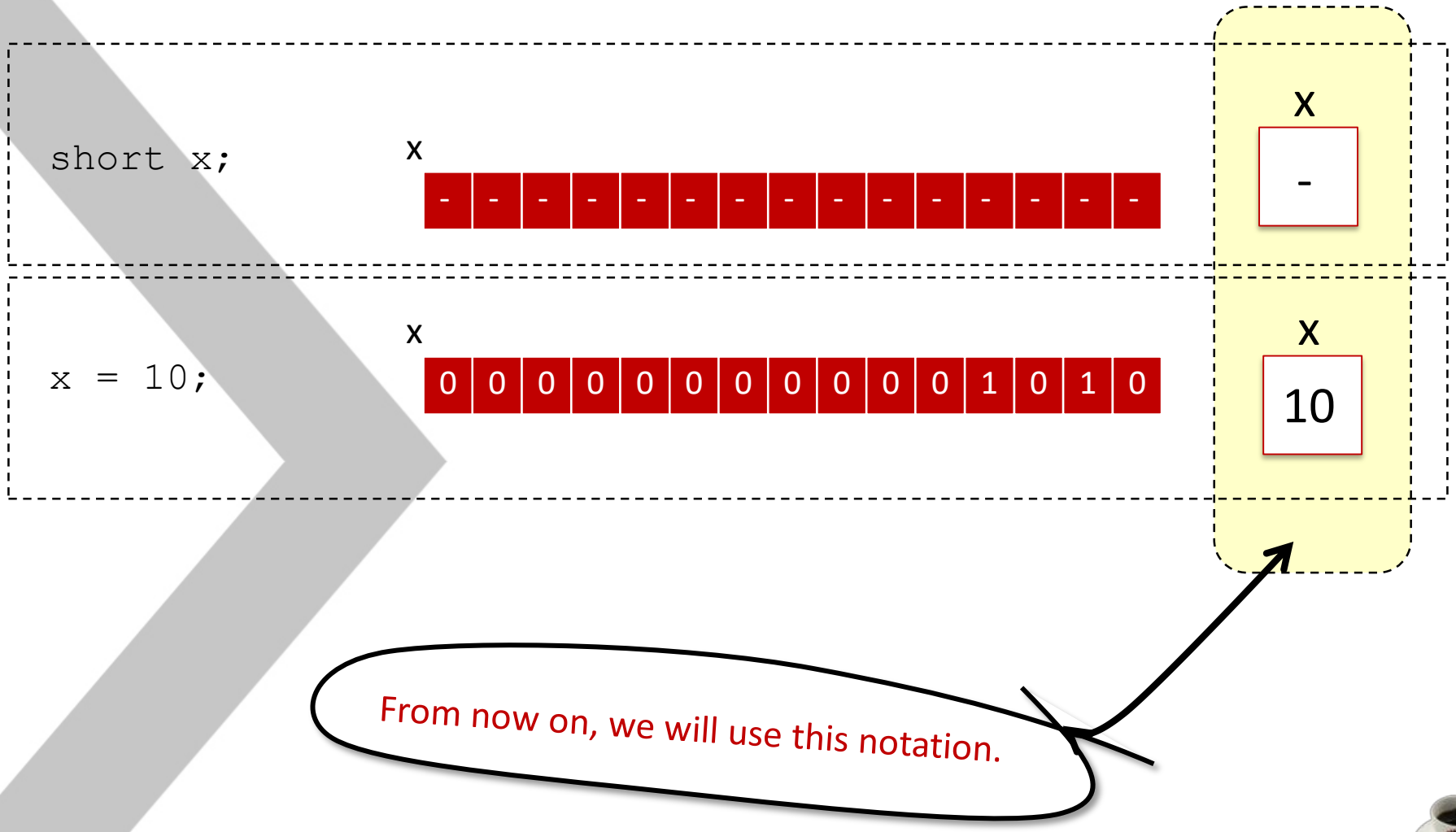
syntax

[Variable] = [Expression];

Examples

```
j = 1; k = 2; l = 3;

isit = false;

j = k;

l = isit; // wrong
```

Notation to be Used

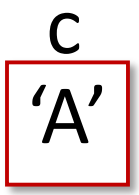


What are stored in actual memory?

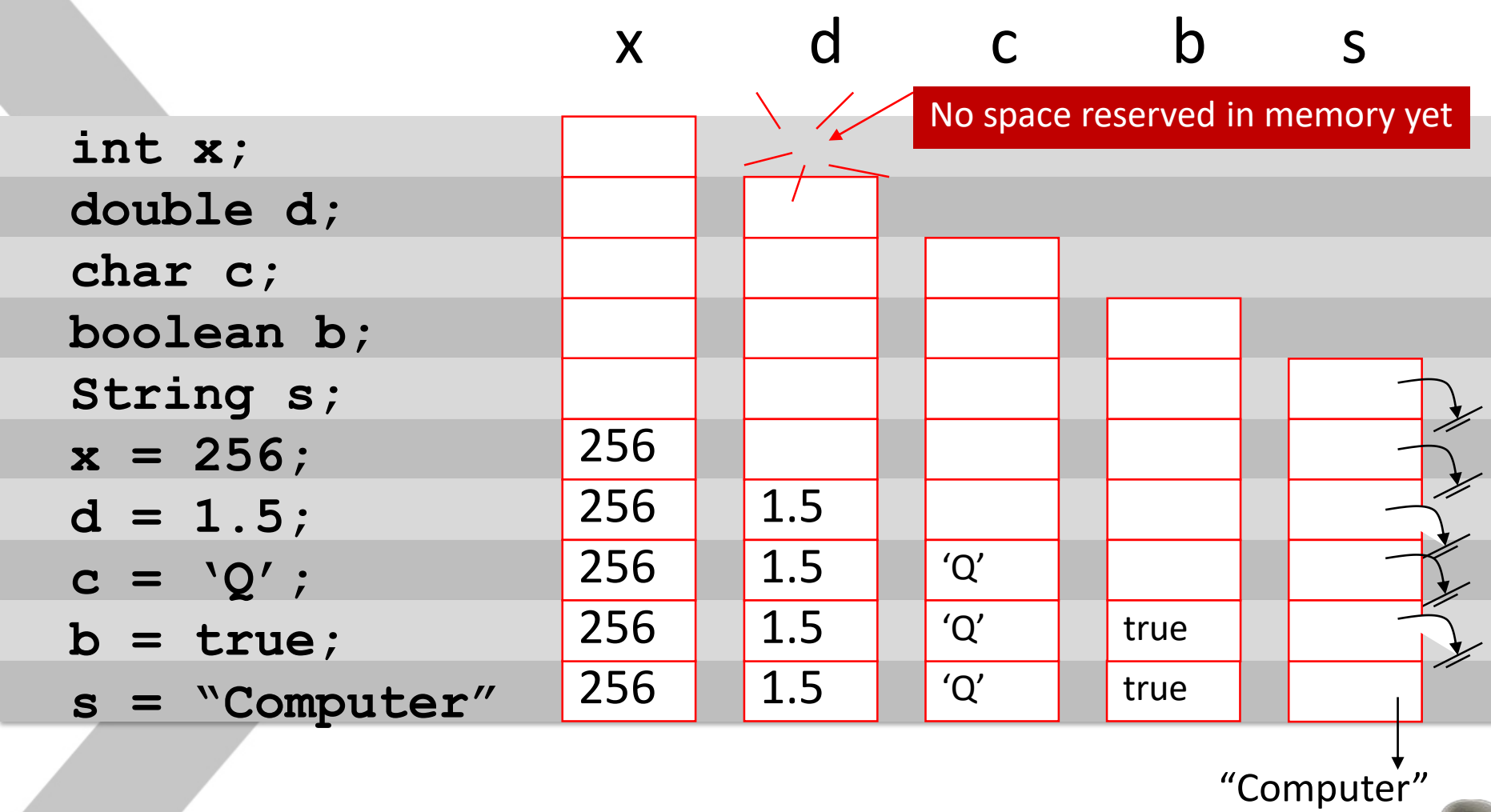
- Variable with **primitive data type**
  - memory space of the size corresponding to its type is allocated.
  - The allocated space is used for storing the value of that type.
- Variable with **Class type**
  - The memory space allocated for such a variable is called a ***reference***.
  - When assigned with a value, the reference points, or refers, to the memory location that actually stores that value.

What are stored in actual memory?


```
char c = 'A';
```

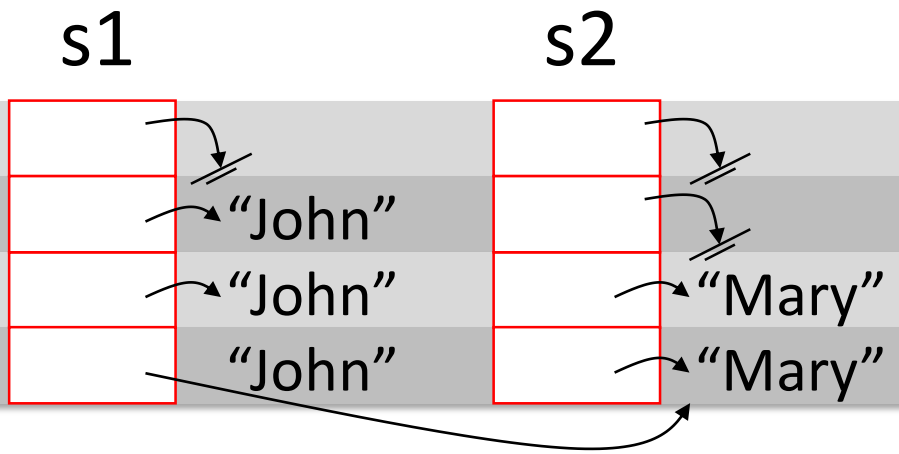



```
String s = "A Good Grip";
```


x
d
c
b
s
No space reserved in memory yet
int x;
double d;
char c;
boolean b;
String s;
x = 256;
d = 1.5;
c = 'Q';
b = true;
s = "Computer"
256
256
1.5
256
1.5
'Q'
256
1.5
'Q'
true
256
1.5
'Q'
true
"Computer"

String Assignments


```
String s1, s2;
s1 = "John";
s2 = "Mary";
s1 = s2;
```



Final Variables

- cannot or will not be changed as long as the program has not terminated.
- The keyword **final** is used when a variable is declared
  - so that the value of that variable cannot be changed once it is initialized, or assigned for the first time.
- Programmers usually use all-uppercase letters for the identifiers of final variables, and use underscore (_) to separate words.
  - Examples: YOUNG_S_MODULUS, SPEED_OF_LIGHT

Final Variables: Examples


```
final double G = 6.67e-11;
final double SPEED_OF_SOUND;
SPEED_OF_SOUND = 349.5;
```

```
final int C;
int k = 6;
C = 80;
C = k + 300;
```

Error, due to the assignment in the last line.

Un-initialized Variables

- When a variable is declared, a space in the memory is reserved for the size of the type of that variable.
- However, as long as it is not assigned with a value, the variable does not contain any meaningful value.
- It is said that the variable has not been *intitialized*.
- If the value of an un-initialized variable is used, an error occurs.

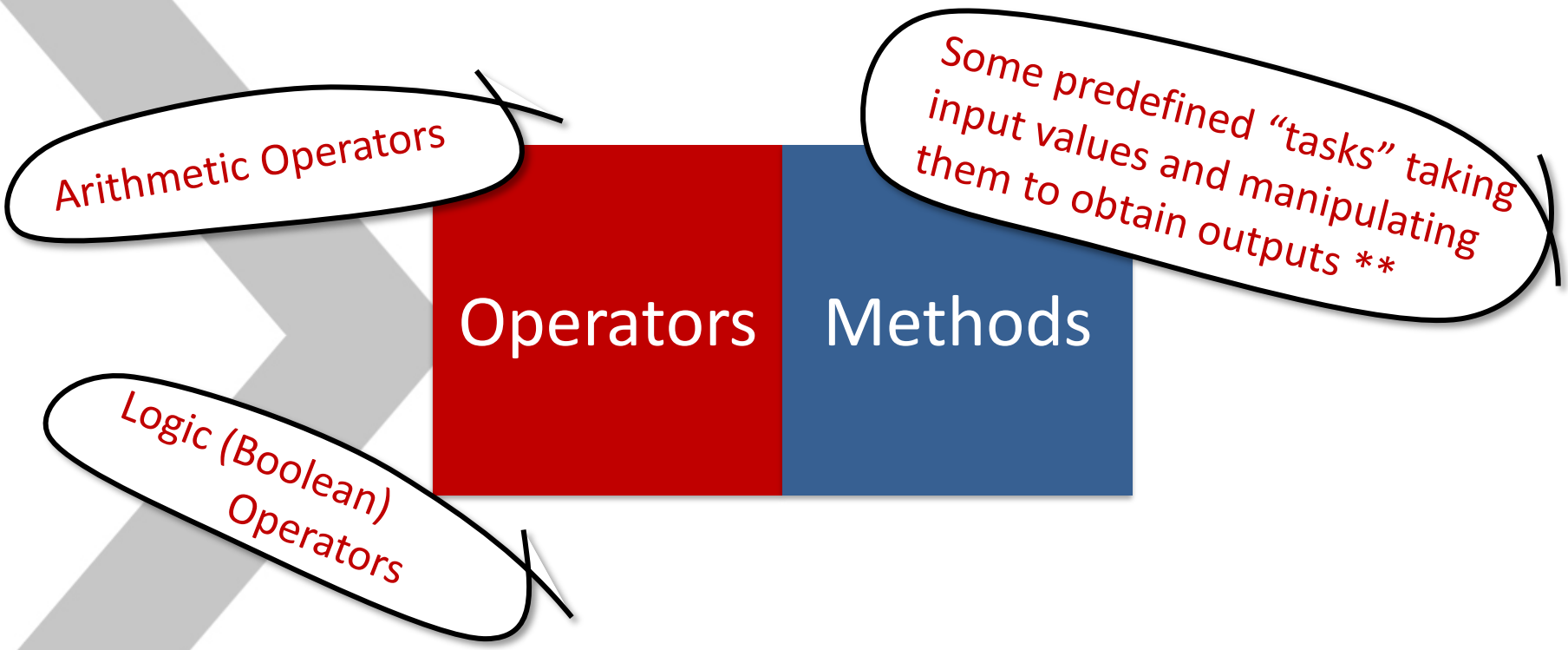
Un-initialized Variables


```
int x, y = 2;
System.out.println(x+y);
```

Error, due to that x is uninitialized.

Operators

- Values can be manipulated using:



** This is not a general definition of methods

Arithmetic Operators

Expression	Value
-3+5-2	0
-3.0+5.0-2.0	0.0
10*8	80
8/4	2
8%4	0
7%4	3
(3+2)*5	25

Logic Operator

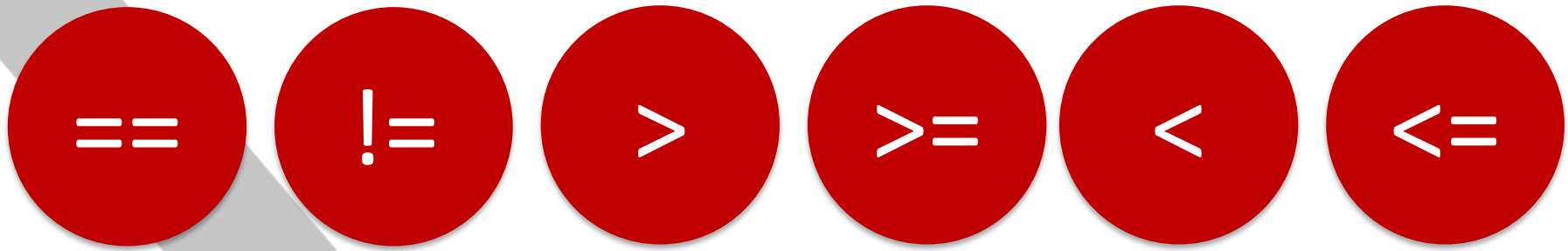
AND OR NOT

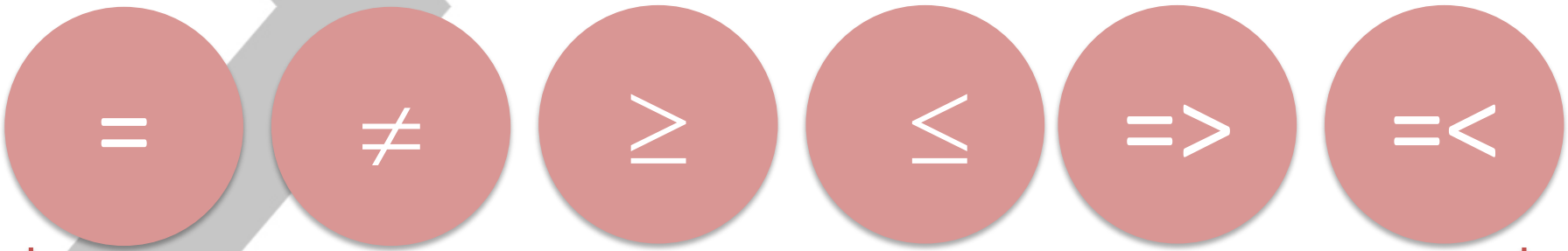
boolean p	boolean q	p && q	p || q	!p
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Operator Categories

- Operators that require two operands are called ***binary operators***.
- Operators that require only one operand are called ***unary operators***.
- Parentheses, (), are called *grouping operators*. They indicate the portions of the calculation that have to be done first.

Comparison Operators





These are incorrect

Assignment and operators: Examples


```
int i = 2, j, k;
j = 3;
k = i + j;
i = i + k;
```

i	j	k
2		
2	3	
2	3	5
7	3	5

```
boolean p = true;
boolean q = false, r = true, s;
s = p && q;
s = s || r;
p = (s == q);
```

p	q	r	s
true			
true	false	true	
true	false	true	false
true	false	true	true
false	false	true	true

String and the addition operator (+)

- Concatenate two Strings together

```
String s1 = "Computer", s2 = "ized", s3;
s3 = s1 + s2;
```

s3 will contain "Computerized".

String and the addition operator (+)

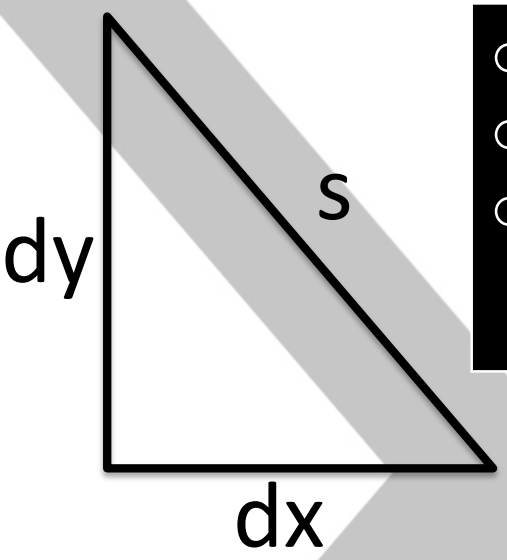
- If a String is added with values of different data types, the non-string value will be converted to String automatically.

meaning that "String concatenation" will be performed by the + operator

- Good news is that the automatic conversion usually returns the String that makes very much sense!

StringConcat.java

Calculations with Methods




```
double dx = 3.0;
double dy = 4.0;
double s =
   Math.sqrt(dx*dx+dy*dy);
```

Using the method Math.sqrt()

Method's name

Input expression

Some useful methods

- **Math.abs(<a numeric value>);**
  - returns the absolute value of the input value.
- **Math.round(<a numeric value>);**
  - returns the integer nearest to the input value.
- **Math.ceil(<a numeric value >);**
  - returns the smallest integer that is bigger than or equal to the input value.
- **Math.floor(<a numeric value >);**
  - returns the biggest integer that is smaller than or equal to the input value.
- **Math.exp(<a numeric value >);**
  - returns the exponential of the input value.
- **Math.max(<a numeric value>,<a numeric value >);**
  - returns the bigger between the two input values.

Some useful methods

- **Math.min(<a numeric value>, <a numeric value>);**
  - returns the smaller between the two input values.
- **Math.pow(<a numeric value>, <a numeric value>);**
  - returns the value of the first value raised to the power of the second value.
- **Math.sqrt(<a numeric value>);**
  - returns the square root of the input value.
- **Math.sin(<a numeric value >);**
  - returns the trigonometric sine value of the input value.
- **Math.cos(<a numeric value >);**
  - returns the trigonometric cosine value of the input value.
- **Math.tan(<a numeric value >);**
  - returns the trigonometric tangent value of the input value.

MathTest.java

Precedence and Associativity


```
int myNumber = 3 + 2 * 6;
```

30 or 15 ???

- In fact, Java compiler has no problem with such ambiguity.
- Order of the operators can be determined using ***Precedence* and *Association* rules**.

Precedence and Associativity

- Operators with higher **precedence** levels are executed before ones with lower precedence levels.
- **Associativity** is also assigned to operators with the same precedence level. It indicates whether operators to the left or to the right are to be executed first.
- In any case, expressions in **parentheses () are executed first**. In the case of nested parentheses, the expression in the innermost pair is executed first.

Precedence and Associativity

Operator	Precedence	Associativity
Grouping operator (())	17	Left to right
Unary operator (+, -, !)	13	Right to left
Multiplicative operator (*, /, %)	12	Left to right
Additive operator (+, -)	11	Left to right
Relational ordering (<, >, <=, >=)	10	Left to right
Relational equality (==, !=)	9	Left to right
Logical and (&&)	4	Left to right
Logical or (||)	3	Left to right
Assignment (=)	1	Right to left

Table 7: Precedence and associativity of some basic operators

Precedence and Associativity


```
4 * 2 + 20 / 4
```

Figure 51: Order of operations in evaluating 4*2+20/4

Precedence and Associativity

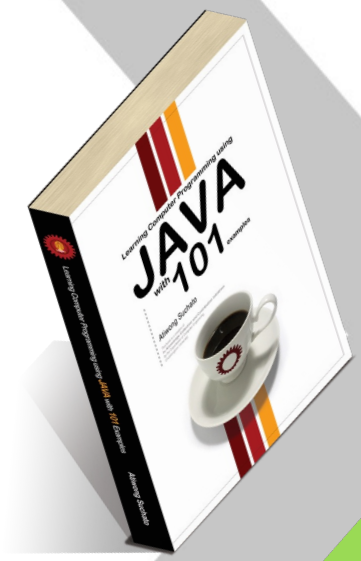

```
2 + 2 == 6 - 2 + 0
```

Figure 52: Order of operations in evaluating 2+2==6-2+0

Chapter 8

Methods

Objectives



Chapter 8

Students should:

- Be able to define new methods and use them correctly.
- Understand the process of method invocation.

100th Anniversary of Chula Engineering 2013

The problems

Source Code

Hard-to-understand

Redundant

- hard to get it right
- less productive to debug
- unmaintainable

```java
import java.util.Scanner;
import java.io.File;
public class BadFiveQuestionsStats{
    public static void main(String [] args){
        Scanner kbdScanner = new Scanner(System.in);
        Scanner fileScanner = null;
        Scanner lineScanner = null;
        boolean fileOK = false;
        do{
            System.out.print("Enter the 5-Qs csv filename:");
            String filename = kbdScanner.next();
            try{
                fileScanner = new Scanner(new File(filename));
                fileOK = true;
                System.out.println(">> OK: "+filename+" opened.");
            }catch(Exception e){
                System.out.println(">> ERROR: Error opening file. Tr
            }
        }while(!fileOK);
        String headerLine = fileScanner.nextLine();
        lineScanner = new Scanner(headerLine);
        lineScanner.useDelimiter(",");
        String idColName = lineScanner.next();
        String q1Name = lineScanner.next();
        String q2Name = lineScanner.next();
        String q3Name = lineScanner.next();
        String q4Name = lineScanner.next();
        String q5Name = lineScanner.next();
        String studentId;
        int sum = 0;
        int sumQ1 = 0;
        int sumQ2 = 0;
        int sumQ3 = 0;
        int sumQ4 = 0;
        int sumQ5 = 0;
        int sumSquare = 0
        int sumSquareQ1
        int sumSquareQ2
        int sumSquareQ3
        int sumSquareQ4
        int sumSquareQ5
        int q1, q2, q3,
        int lineNumber
        int totalOfAPers
        String line;
        while(fileScanner.h
            totalOfAPerson = 0;
            try{
                line = fileScanner.nextLine();
                lineScanner = new Scanner(line);
                lineScanner.useDelimiter(",");
```

```java
while(fileScanner.hasNext()){
    totalOfAPerson = 0;
    try{
        line = fileScanner.nextLine();
        lineScanner = new Scanner(line);
        lineScanner.useDelimiter(",");
        studentId = lineScanner.next();
        q1 = lineScanner.nextInt();
        q2 = lineScanner.nextInt();
        q3 = lineScanner.nextInt();
        q4 = lineScanner.nextInt();
        q5 = lineScanner.nextInt();
        sumQ1 += q1;
        sumQ2 += q2;
        sumQ3 += q3;
        sumQ4 += q4;
        sumQ5 += q5;
        totalOfAPerson = q1+q2+q3+q4+q5;
        sum += totalOfAPerson;
        sumSquareQ1 += q1*q1;
        sumSquareQ2 += q2*q2;
        sumSquareQ3 += q3*q3;
        sumSquareQ4 += q4*q4;
        sumSquareQ5 += q5*q5;
        sumSquare += totalOfAPerson*totalOfAPerson;
        System.out.println(">> OK: Points of "+studentId+" processed
    }catch(Exception e){
        System.out.println(">> ERROR: Problems on line #"+lineNumber
    }
    lineNumber++;
}
int num = lineNumber-1;
double averageQ1 = (double)sumQ1/num;
double averageQ2 = (double)sumQ2/num;
uble averageQ3 = (double)sumQ3/num;
ble averageQ4 = (double)sumQ4/num;
ble averageQ5 = (double)sumQ5/num;
ble averageTotal = (double)sum/num;
uble sdQ1 = Math.sqrt((double)sumSquareQ1/num-averageQ1*averageQ1)
uble sdQ2 = Math.sqrt((double)sumSquareQ2/num-averageQ2*averageQ2)
uble sdQ3 = Math.sqrt((double)sumSquareQ3/num-averageQ3*averageQ3)
uble sdQ4 = Math.sqrt((double)sumSquareQ4/num-averageQ4*averageQ4)
double sdQ5 = Math.sqrt((double)sumSquareQ5/num-averageQ5*averageQ5)
double sdTotal = Math.sqrt((double)sumSquare/num-averageTotal*averag
System.out.println(">> OK: Here are the stats.");
System.out.println("\t\t----\t----\t----\t----\t----\t----");
System.out.println("\t\t"+q1Name+"\t"+q2Name+"\t"+q3Name+"\t"+q4Name
System.out.println("\tAverage\t"+(double)Math.round(averageQ1*100)/1
System.out.println("\tSD\t"+(double)Math.round(sdQ1*100)/100+"\t"+(d
System.out.println("\t\t----\t----\t----\t----\t----\t----");
}
```

This program lets the user pick a CSV file containing student's exam scores, finds related stats, and print the results out.

```java
import java.util.Scanner;
import java.io.File;
public class BadFiveQuestionsStats{
    public static void main(String [] args){
        Scanner kbdScanner = new Scanner(System.in);
        Scanner fileScanner = null;
        Scanner lineScanner = null;
        boolean fileOK = false;
        do{
            System.out.print("Enter the 5-Qs csv filename:");
            String filename = kbdScanner.next();
            try{
                fileScanner = new Scanner(ne
                fileOK = true;
                System.out.println(">>
            String headerLine = fileScanne
            lineScanner = new Scanner(headerLine
            lineScanner.useDelimiter(",");
            String idColName = lineScanner.next();
            String q1Name = lineScanner.next();
            String q2Name = lineScanner.next();
            String q3Name = lineScanner.next();
            String q4Name = lineScanner.next();
            String q5Name = lineScanner.next();
            String studentId;
            int sum = 0;
            int sumQ1 = 0;
            int sumQ2 = 0;
            int sumQ3 = 0;
            int sumQ4 = 0;
            int sumQ5 = 0;
            int sumSquare = 0
            int sumSquareQ1
            int sumSquareQ2
            int sumSquareQ3
            int sumSquareQ4
            int sumSquareQ5
            int q1, q2, q3,
            int lineNumber
            int totalOfAPers
            String line;
            while(fileScanner.h
                totalOfAPerson = 0;
                try{
                    line = fileScanner.nextLine();
                    lineScanner = new Scanner(line);
                    lineScanner.useDelimiter(",");
```

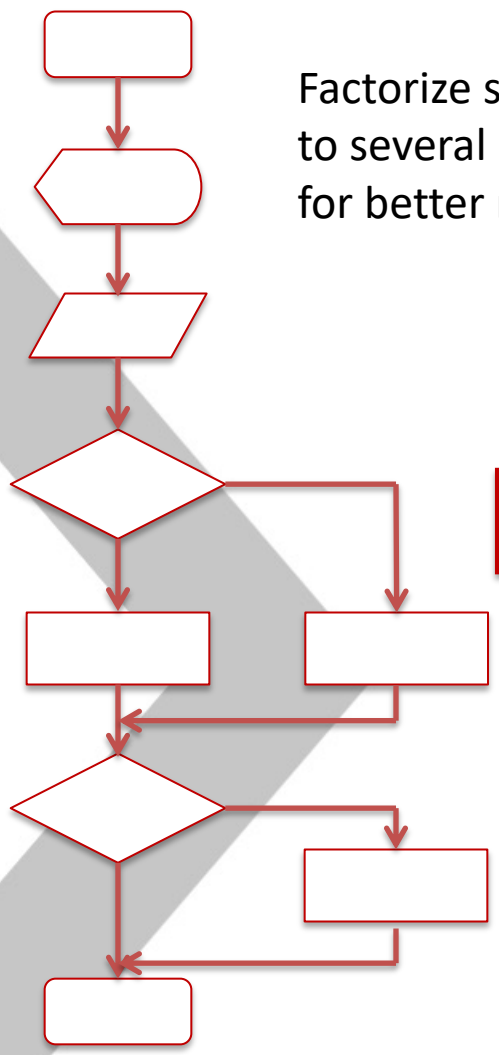
```java
    while(fileScanner.hasNext()){
        totalOfAPerson = 0;
        try{
            line = fileScanner.nextLine();
            lineScanner = new Scanner(line);
            lineScanner.useDelimiter(",");
            studentId = lineScanner.next();
            q1 = lineScanner.nextInt();
            q2 = lineScanner.nextInt();
            q3 = lineScanner.nextInt();
            q4 = lineScanner.nextInt();
                sumSquareQ5 += q5*q5;
                sumSquare += totalOfAPerson*totalOfAPerson;
                System.out.println(">> OK: Points of "+studentId+" processed
            }catch(Exception e){
                System.out.println(">> ERROR: Problems on line #"+lineNumber
            }
            lineNumber++;
        }
        int num = lineNumber-1;
        double averageQ1 = (double)sumQ1/num;
        double averageQ2 = (double)sumQ2/num;
        uble averageQ3 = (double)sumQ3/num;
        ble averageQ4 = (double)sumQ4/num;
        ble averageQ5 = (double)sumQ5/num;
        ble averageTotal = (double)sum/num;
        uble sdQ1 = Math.sqrt((double)sumSquareQ1/num-averageQ1*averageQ1)
        uble sdQ2 = Math.sqrt((double)sumSquareQ2/num-averageQ2*averageQ2)
        uble sdQ3 = Math.sqrt((double)sumSquareQ3/num-averageQ3*averageQ3)
        uble sdQ4 = Math.sqrt((double)sumSquareQ4/num-averageQ4*averageQ4)
        double sdQ5 = Math.sqrt((double)sumSquareQ5/num-averageQ5*averageQ5)
        double sdTotal = Math.sqrt((double)sumSquare/num-averageTotal*averag
        System.out.println(">> OK: Here are the stats.");
        System.out.println("\t\t----\t----\t----\t----\t----\t----");
        System.out.println("\t\t"+q1Name+"\t"+q2Name+"\t"+q3Name+"\t"+q4Name
        System.out.println("\tAverage\t"+(double)Math.round(averageQ1*100)/1
        System.out.println("\tSD\t"+(double)Math.round(sdQ1*100)/100+"\t"+(d
        System.out.println("\t\t----\t----\t----\t----\t----\t----");
    }
}
```

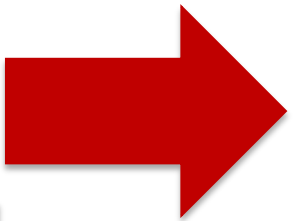
Better?

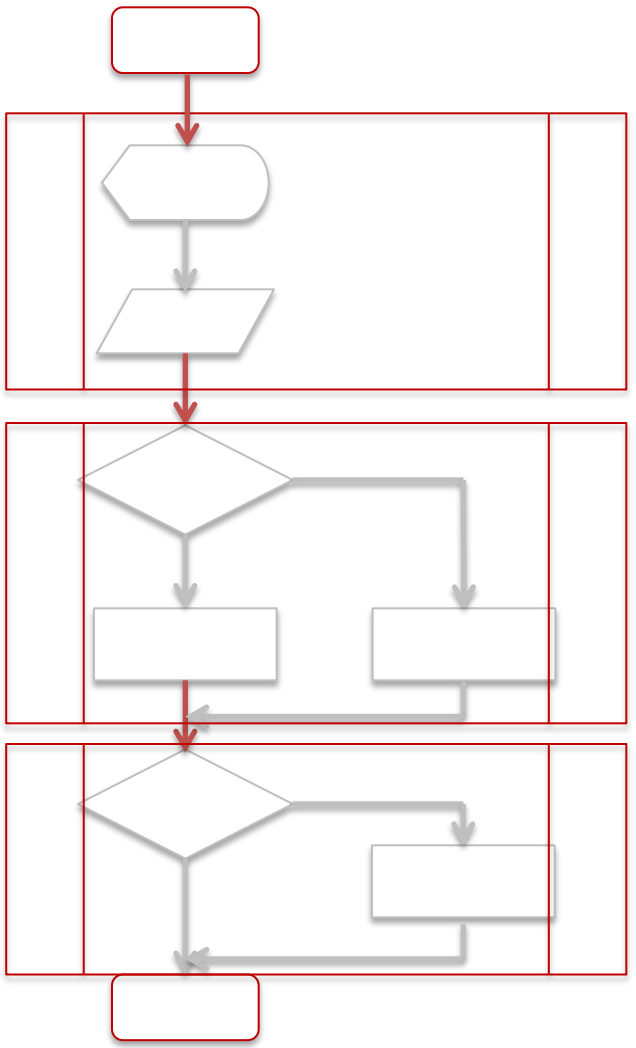
```java
public static void main(String [] args){
    Scanner fileScanner = getFilenameAndtryOpenFile();
    calculateAndShowStats(fileScanner);
}
```

This program lets the user pick a CSV file containing student's exam scores, finds related stats, and print the results out.
(Hands-on Exp # 6)

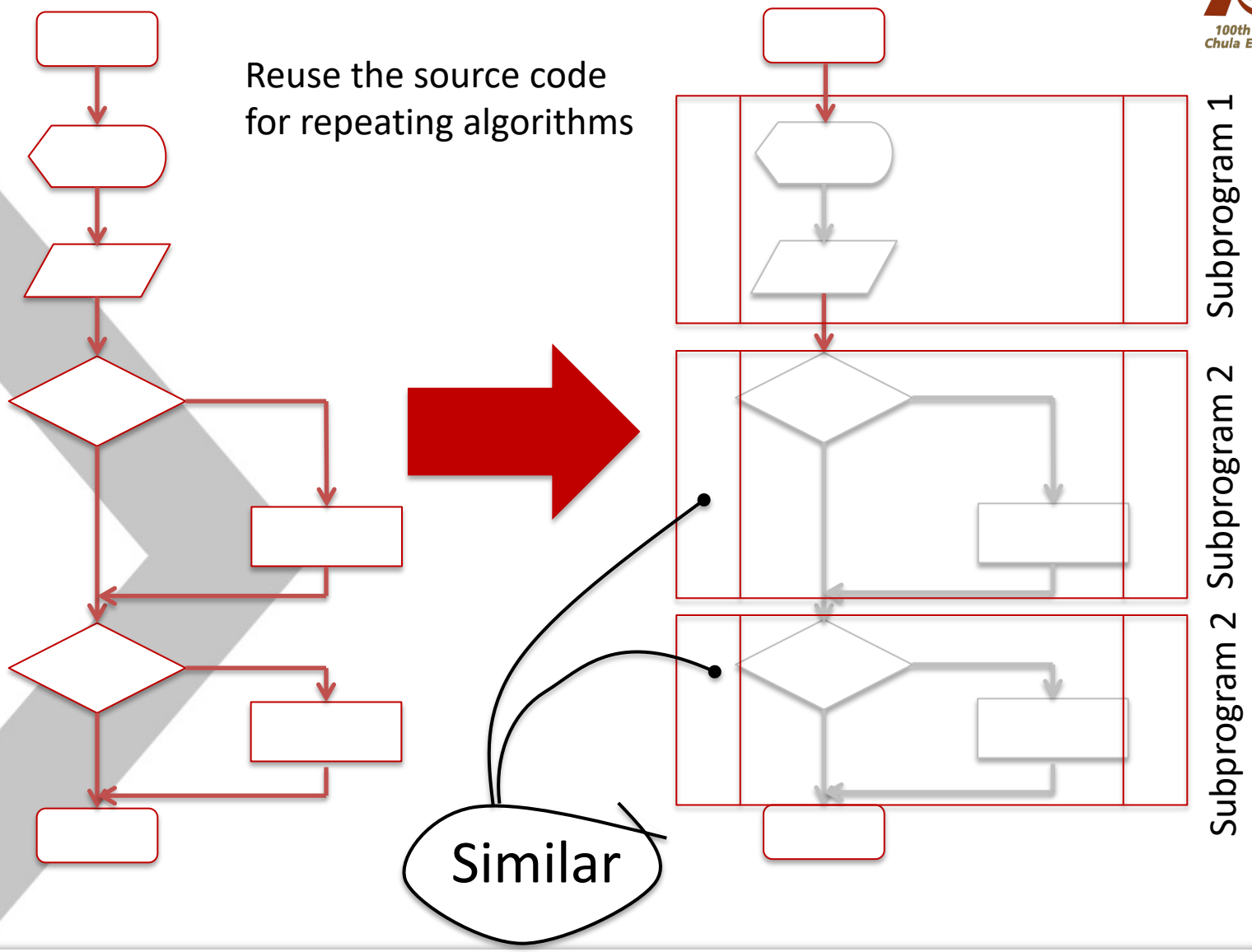
Factorize source code
to several subprograms
for better readability







Reuse the source code
for repeating algorithms



Example

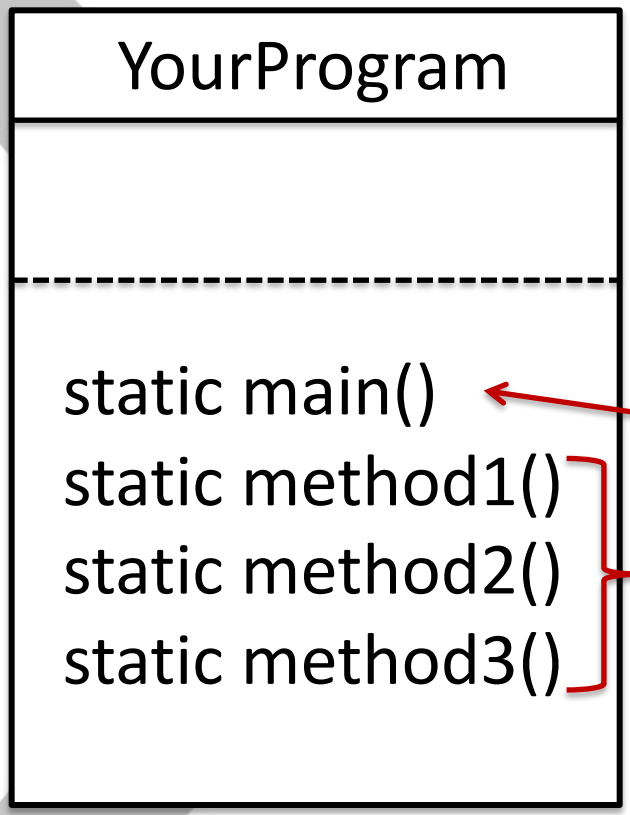
- Compute f(x,n) for x = 1.5, 2.5, 3.5, and 4.5 where n = 3.

$$y = f(x,n) = x + 2x^2 + 3x^3 + \ldots + nx^n$$

MethodDemo1.java

MethodDemo2.java

Subprograms in OOP

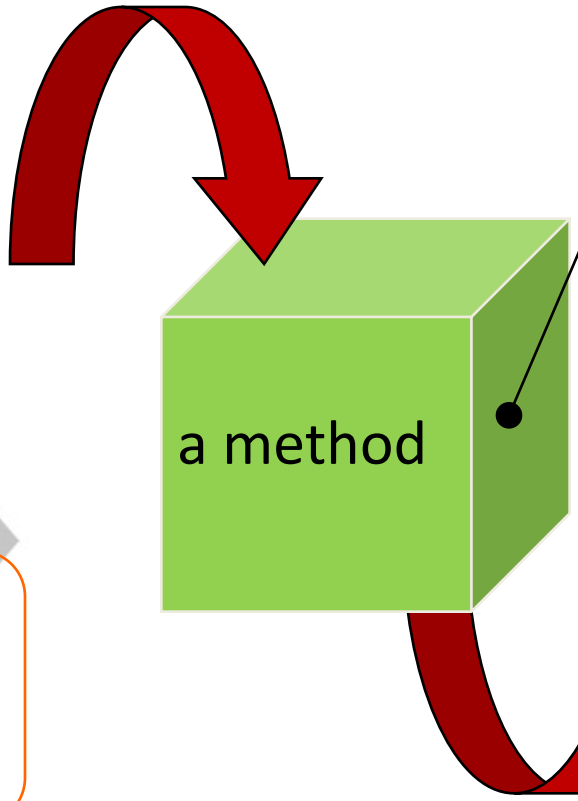


Subprograms are provided in Java as "Static methods" defined in your program.

Java calls this method to start executing your program

These methods can be called in your programs (or in others)

Defining New Methods

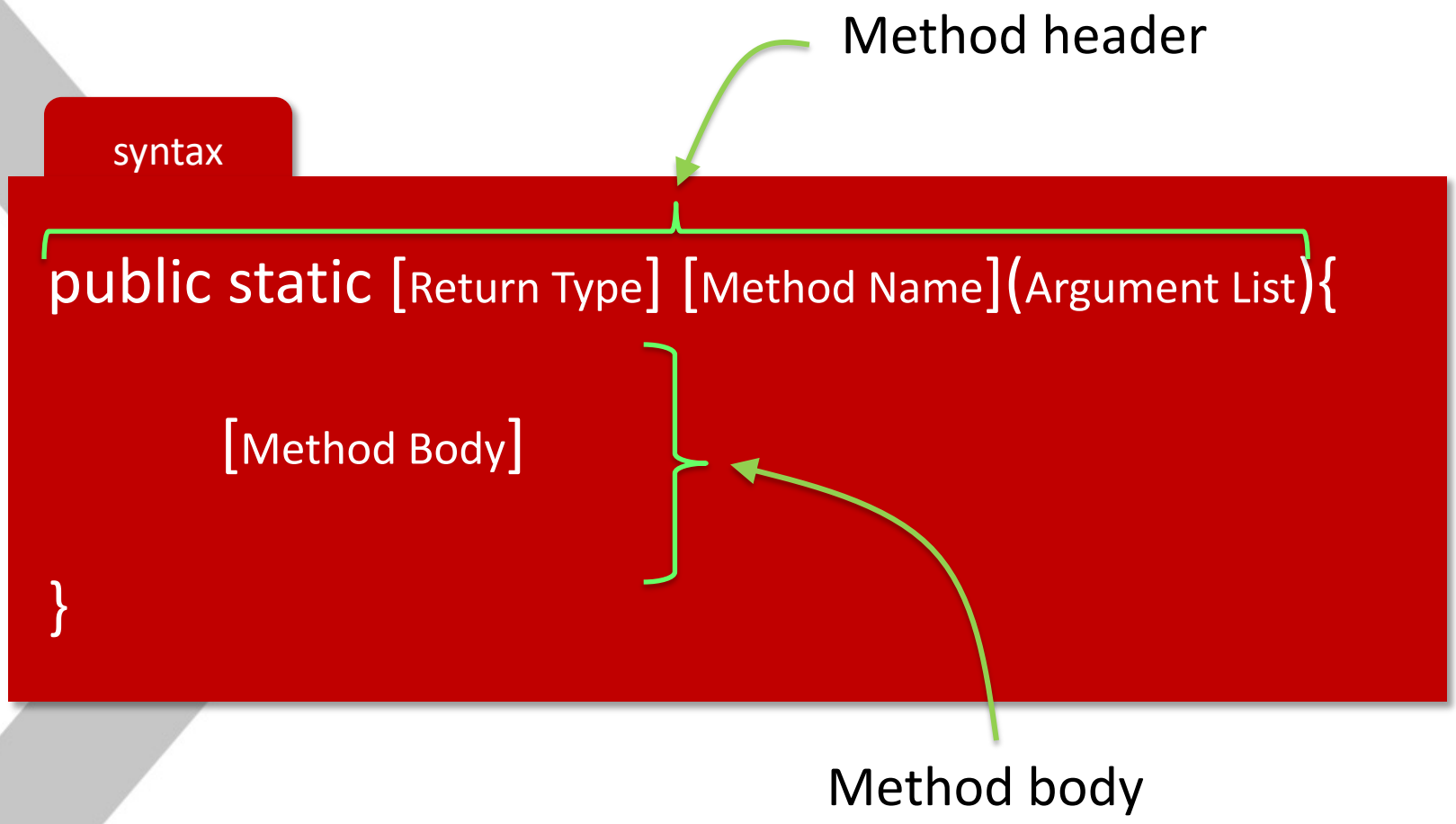


1 What the input argument list must be.

2 What to do with the each input in the argument list.

3 What the (type of) output will be like.

Defining a Method



Public and static

- **public** and **static**
  - are Java keywords.
  - **public** identifies that this method can be used by any classes.
  - **static** identifies that this method is a class method (static method).

Return Type

- The data type expected to be returned by the method.

The return type is the

data type

of the expression invoking the method.

When a method returns something

only **ONE** value

can be returned.

When a method is NOT designed to return a value

void keyword

is used as the return type

Method Name

- **methodName**
  - be replaced with the name (identifier) you give for the method.
  - Java naming rules apply to the naming of methods as well as other identifiers.

Recommendations for Method Naming

- Choose meaningful name
- Start with an "action" (Verb)
- Do not capitalize the first letter

Eg.

getUserInput()
killTheDrone()
launch()
doItAgain()

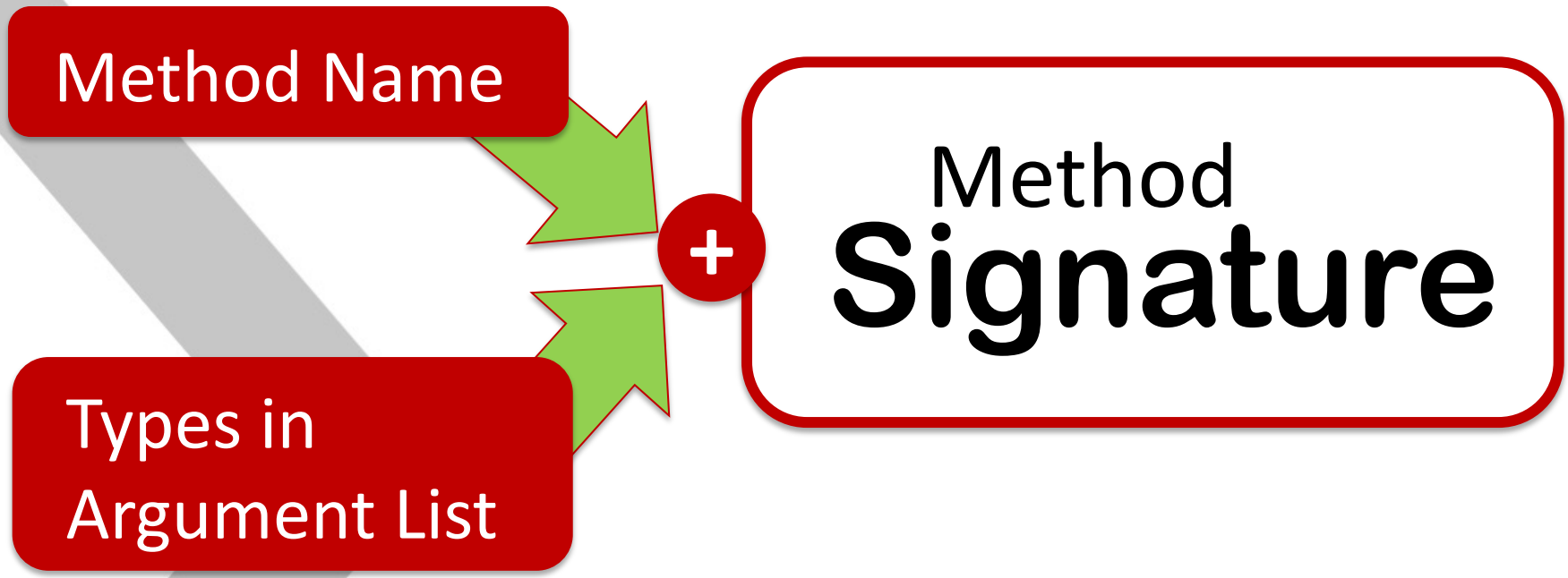
Input Argument List

syntax

```
(Type1 arg1, Type2 arg2, … , TypeN argsN)
```

- consisting of parameters expected to be input to the method.
- When the method does not need any input parameters, do not put anything in the parentheses.

Method Signature



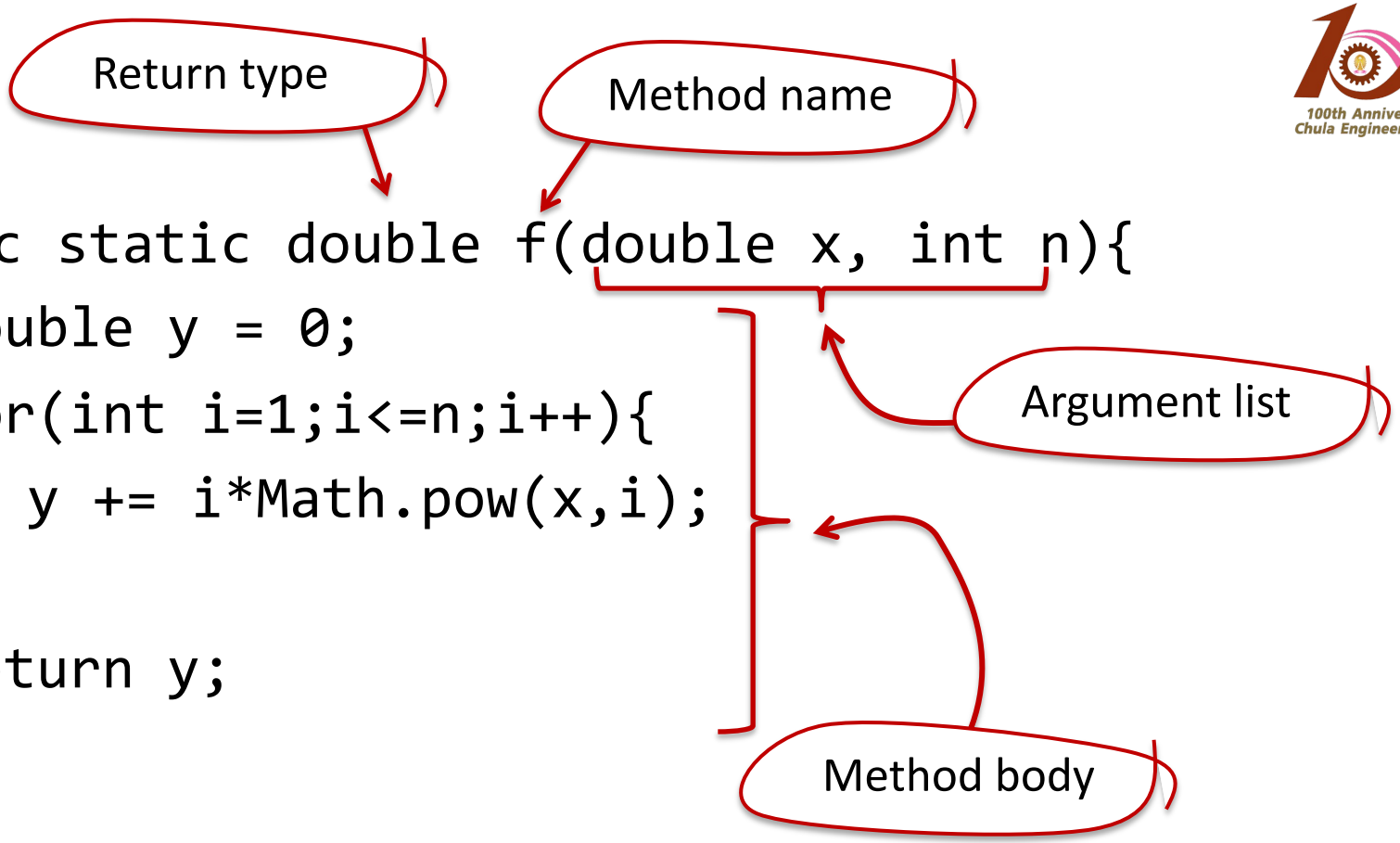
[Method Name](Type1 , Type2 , … , TypeN)

Method Body

- **Method Body**
  - is the list of statements to be executed once the method is called.
  - The body of the method might contain *return* statements
    - the keyword *return* is placed in front of the value wished to be returned to the caller of the method.
    - the value returned is of the same type as, or can be automatically converted to, what is declared as **Return Type** in the method header.

Return statements

- **Return statements**
  - mark terminating points of the method.
  - Whenever a return statement is reached
    - the program flow is passed from the method back to the caller of the method.
  - If there is nothing to be returned
    - i.e. **Return Type** is *void*
    - the keyword *return* cannot be followed by any value.
    - the program flow is still passed back to the caller but there is no returned value.



Method Signature

```
f( double , int )
```

Examples of method definition


```
public static boolean isOdd(int n){
   return (n%2 != 0)? true : false;
}
```

```
public static int unicodeOf(char c){
   return (int)c;
}
```

Examples of method definition


```
public static
   String longer(String s1, String s2){
      return ((s1.length() > s2.length())?s1:s2);
}
```

```
public static int factorial(int n){
    int nFact = 1;
    for(int i=1;i<=n;i++){
       nFact *= i;
    }
    return nFact;
}
```

Examples of method definition

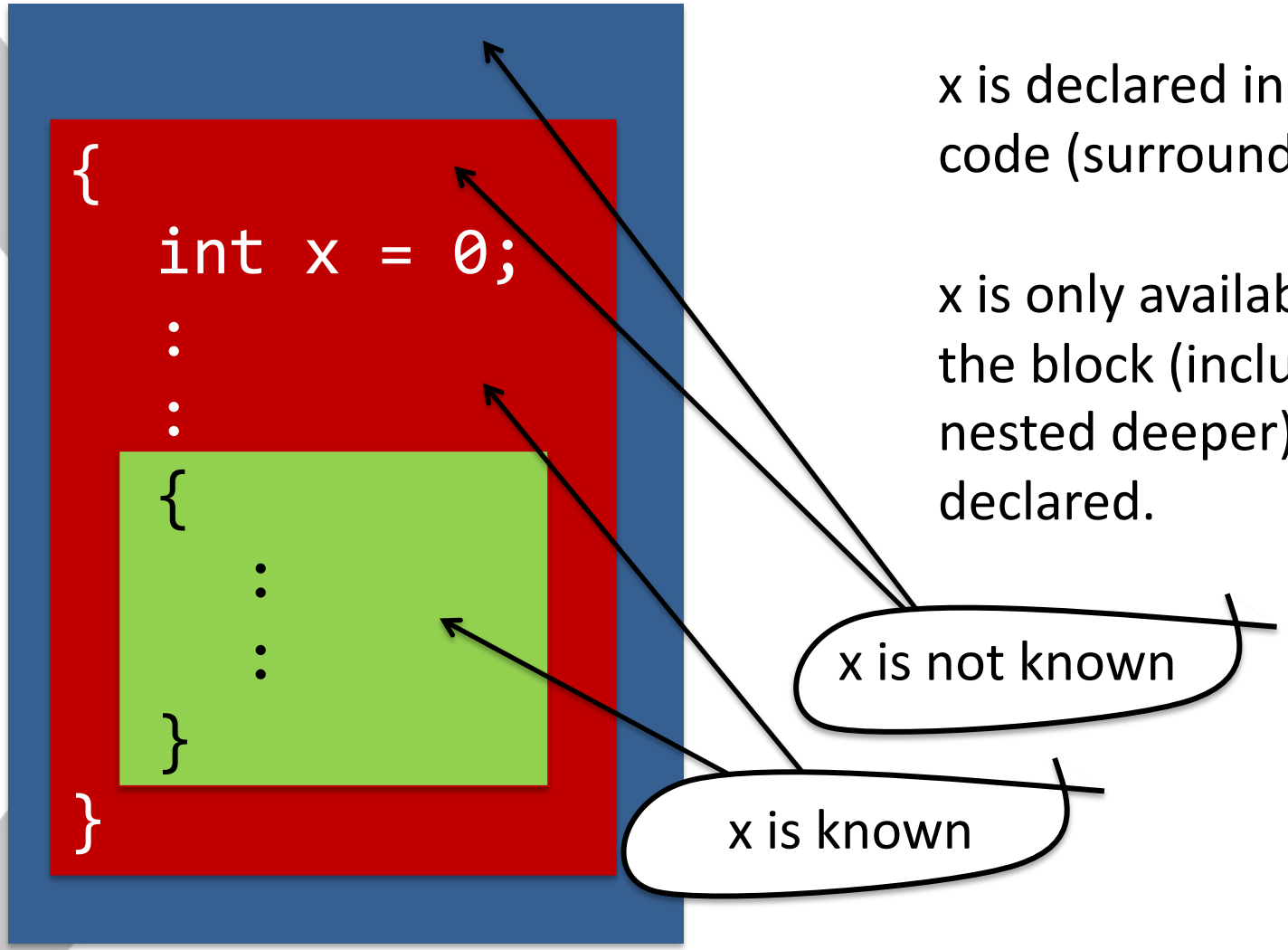

```
public static void printGreetings(String name){
    System.out.println("Hello "+name);
    System.out.println("Welcome to ISE mail system.");
    System.out.println("----------------------------");
}
```

Scopes of Variables

Variables declared in a method are **Local** to that method.

- Variables declared in the
  - argument list
  - inside the method
- are only available to the method and destroyed once the method is terminated.

Scopes of Variables



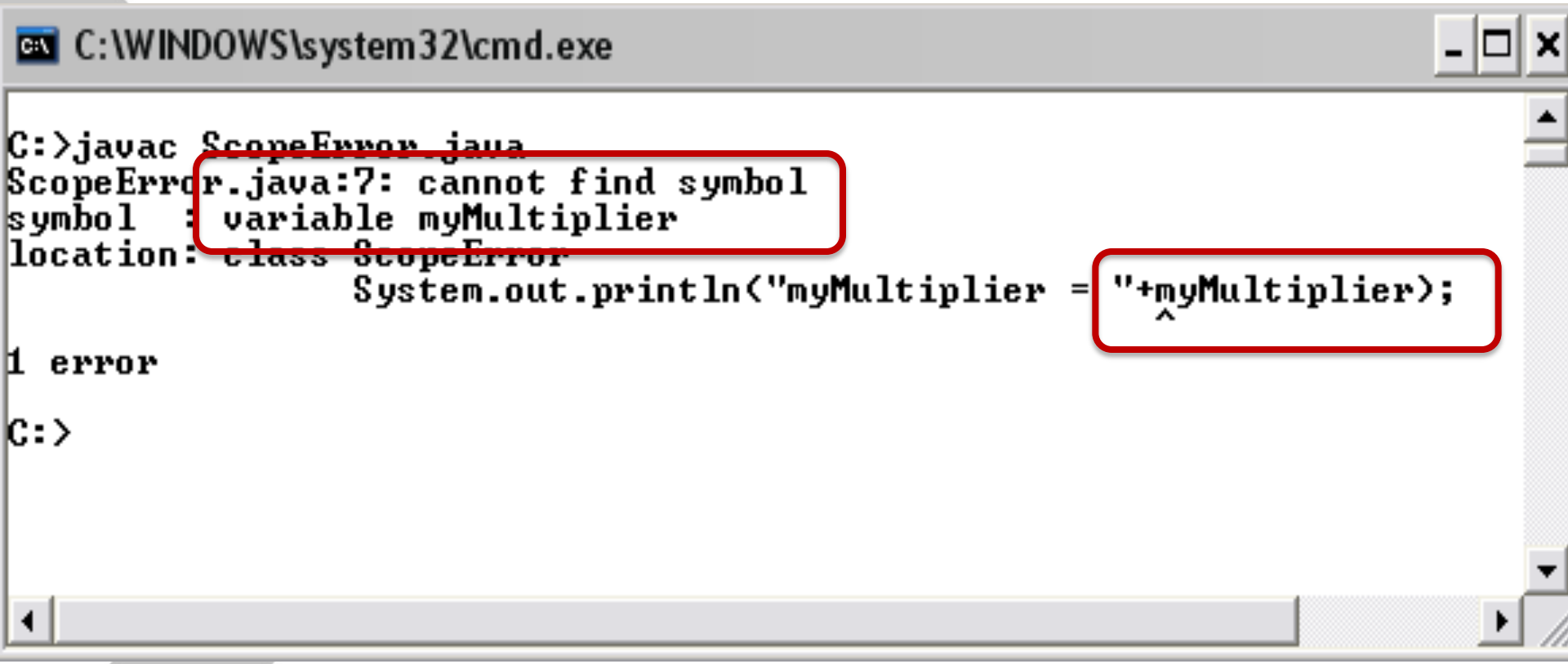
x is declared in a block of code (surrounded by {}).

x is only available inside the block (including block nested deeper) after it is declared.

Cannot Find Symbol !?!


```
public class ScopeError
{
   public static void main(String[] args)
   {
      int x=0, y=0, z;
      z = f(x,y);
      System.out.println("myMultiplier = "+myMultiplier);
      System.out.println("z="+z);
   }
   public static int f(int a, int b)
   {
      int myMultiplier = 256;
      return myMultiplier*(a+b);
   }
}
```

Cannot Find Symbol !?!

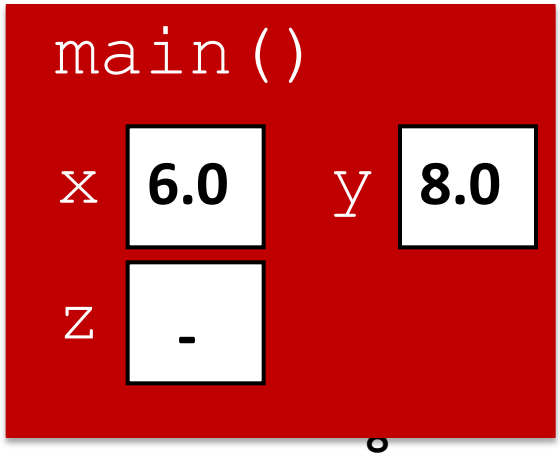


Method Invocation Mechanism


```
public class MethodInvokeDemo                                    1
{                                                                2
   public static void main(String[] args)                       3
   {                                                             4
    double x = 6.0, y = 8.0, z;                                  5
    z = f(x,y,2);                                                6
    System.out.println(z);                                       7
   }                                                             8
   public static double f(double a,double b, int n)              9
   {                                                             10
    double an = Math.pow(a,n);                                   11
    double bn = Math.pow(b,n);                                   12
    return Math.pow(an+bn,1.0/n);                                13
   }                                                             14
}                                                                15
```

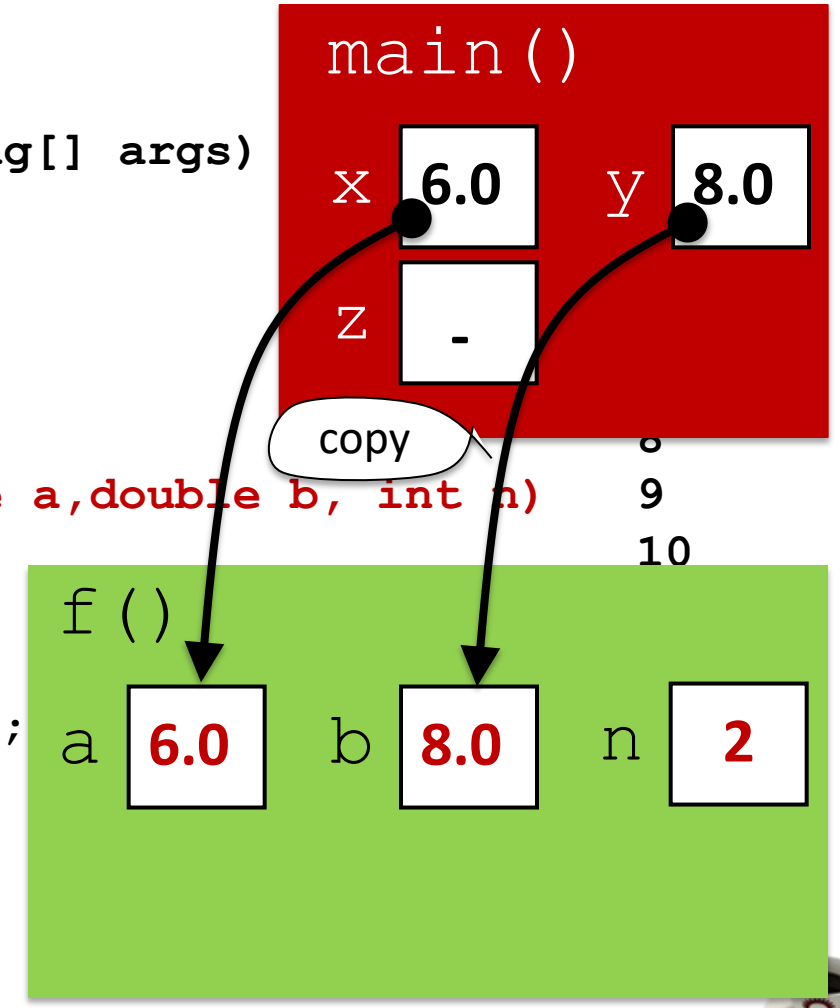
Method Invocation Mechanism


```
public class MethodInvokeDemo
{
   public static void main(String[] args)
   {
    double x = 6.0, y = 8.0, z;
    z = f(x,y,2);
    System.out.println(z);
   }
   public static double f(double a,double b, int n)
   {
    double an = Math.pow(a,n);
    double bn = Math.pow(b,n);
    return Math.pow(an+bn,1.0/n);
   }
}
```



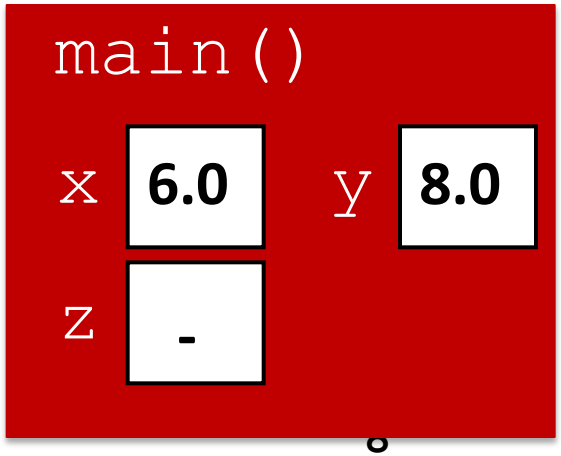
Method Invocation Mechanism

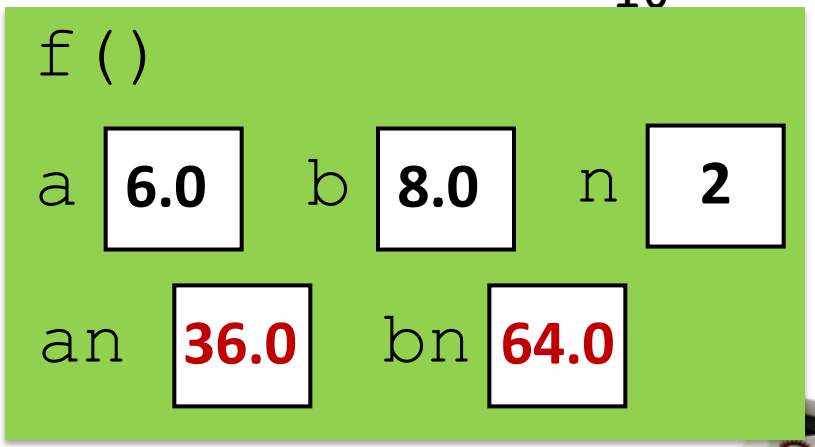

```
public class MethodInvokeDemo
{
   public static void main(String[] args)
   {
    double x = 6.0, y = 8.0, z;
    z = f(x,y,2);
    System.out.println(z);
   }
   public static double f(double a,double b, int n)
   {
    double an = Math.pow(a,n);
    double bn = Math.pow(b,n);
    return Math.pow(an+bn,1.0/n);
   }
}
```



Method Invocation Mechanism

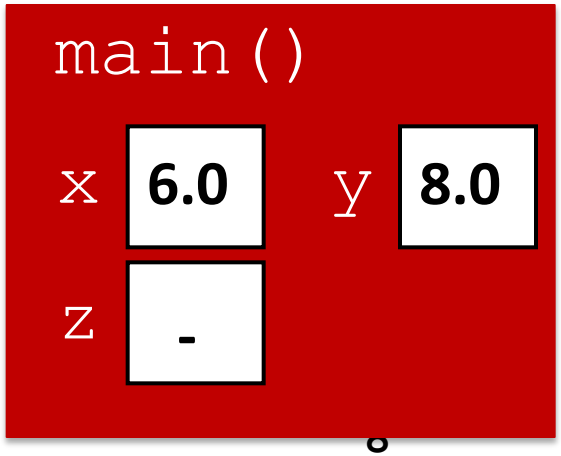

```
public class MethodInvokeDemo
{
   public static void main(String[] args)
   {
    double x = 6.0, y = 8.0, z;
    z = f(x,y,2);
    System.out.println(z);
   }
   public static double f(double a,double b, int n)
   {
    double an = Math.pow(a,n);
    double bn = Math.pow(b,n);
    return Math.pow(an+bn,1.0/n);
   }
}
```

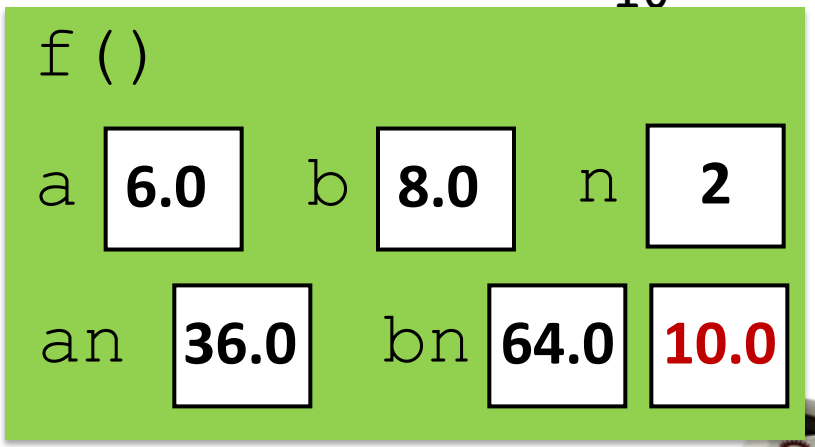




Method Invocation Mechanism

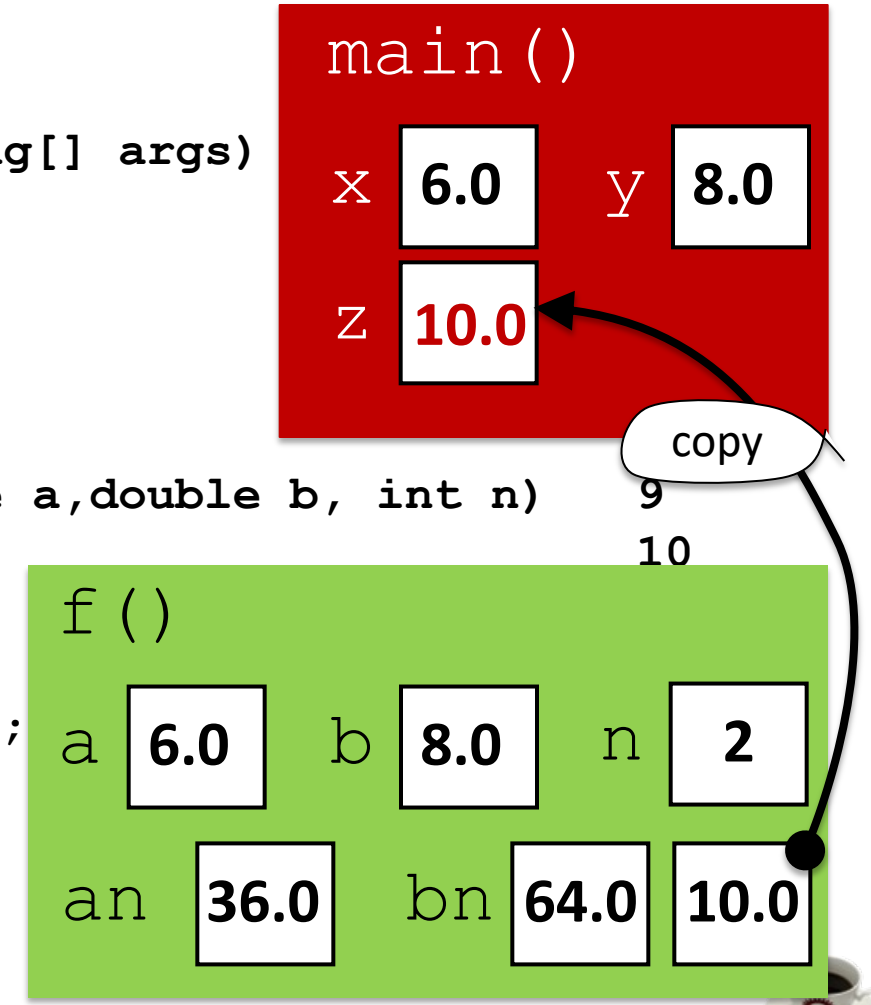

```
public class MethodInvokeDemo
{
   public static void main(String[] args)
   {
    double x = 6.0, y = 8.0, z;
    z = f(x,y,2);
    System.out.println(z);
   }
   public static double f(double a,double b, int n)
   {
    double an = Math.pow(a,n);
    double bn = Math.pow(b,n);
    return Math.pow(an+bn,1.0/n);
   }
}
```





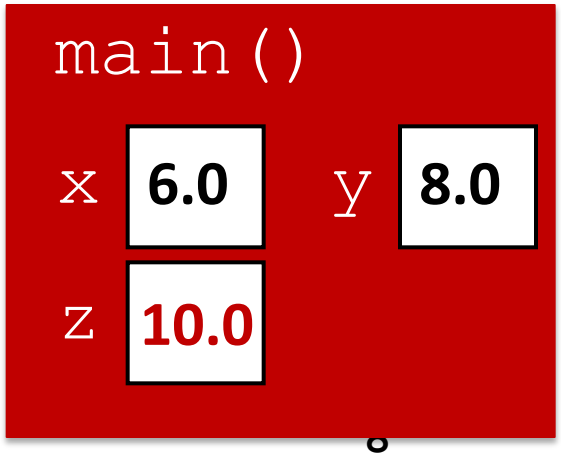
Method Invocation Mechanism

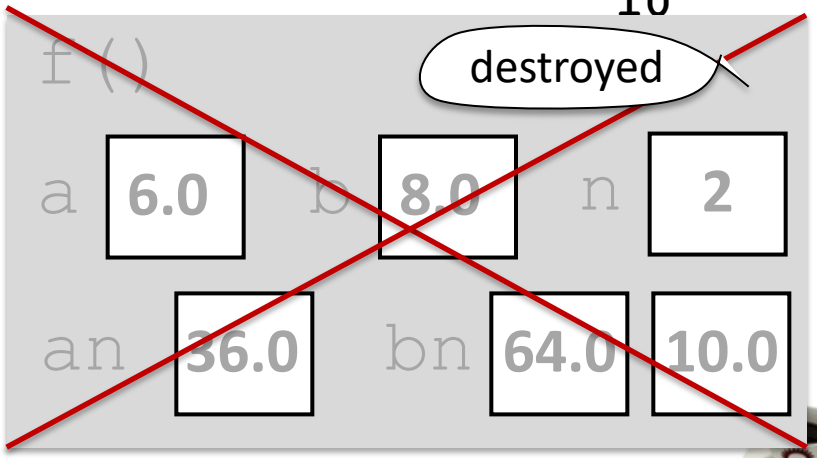

```
public class MethodInvokeDemo
{
   public static void main(String[] args)
   {
    double x = 6.0, y = 8.0, z;
    z = f(x,y,2);
    System.out.println(z);
   }
   public static double f(double a,double b, int n)
   {
    double an = Math.pow(a,n);
    double bn = Math.pow(b,n);
    return Math.pow(an+bn,1.0/n);
   }
}
```



Method Invocation Mechanism


```
public class MethodInvokeDemo
{
   public static void main(String[] args)
   {
    double x = 6.0, y = 8.0, z;
    z = f(x,y,2);
    System.out.println(z);
   }
   public static double f(double a,double b, int n)
   {
    double an = Math.pow(a,n);
    double bn = Math.pow(b,n);
    return Math.pow(an+bn,1.0/n);
   }
}
```





Example

```
public static void main(String[] args){
    int x=1,y=1,w;
    w = add(x,y);
    System.out.println(x+“,”+y);
}

public static int add(int x,int y){
    int z = x + y;
    x = 0;
    y = 0;
    return z;
}
```

What is the output?

A) (0,0)

B) (1,1)

Method Overloading

- Method overloading
  - Different methods *can have the same name as long as their argument lists are different*.

Different method signatures

- Why is it useful?
  - It is useful when we need methods that perform similar tasks but with different argument lists
    - i.e. argument lists with different numbers or types of parameters.

Method Overloading(2)

- How Java know which method to be called?
  - comparing the number and types of input parameters with the argument list of each method definition.
- Note that methods are overloaded based on the difference in the argument list, not their return types.

Method Overloading


```
public class OverloadingDemo                                    1
{                                                               2
   public static void main(String[] args)                      3
   {                                                            4
    System.out.println(numericAdd(1,2));                        5
    System.out.println(numericAdd(1,2,3));                      6
    System.out.println(numericAdd(1.5,2.5));                    7
    System.out.println(numericAdd(1.5,2.5,3.5));                8
    System.out.println(numericAdd('1','2'));                    9
   }                                                            10
```

Method Overloading


```
public static int numericAdd(int x,int y)
{
    return x + y;
}
public static double numericAdd(double x, double y)
{
    return x + y;
}
public static int numericAdd(int x, int y, int z)
{
    return x + y + z;
}
```

Signature
numericAdd(int,int)

Signature
numericAdd(double,double)

Signature
numericAdd(int,int,int)

Method Overloading


```
public static double numericAdd
                  (double x,double y, double z)        24
{                                                      25
    return x + y + z;
}
```

Signature

numericAdd(double,double,double)

```
public static int numericAdd(char x, char y)         28
{
    int xInt = x - '0';                                29
    int yInt = y - '0';                                30
    return xInt+yInt;                                  31
}                                                      32
```

Signature

numericAdd(char,char)

Wrong Method Overloading


```
public static void f(int x, int y){
   …
};
```

```
public static void f(int a, int b){
   …
};
```

Signature

f (int , int)

Wrong Method Overloading


```
public static void f(int x, int y){
   …
};
```

```
public static int f(int x, int y){
   …
};
```

Signature

f (int , int)

Automatic Type Conversion

Signature A

h(int , int)

Signature B

h(double , double)

Which one will be called if the statement

h(1,1.0)

is executed?

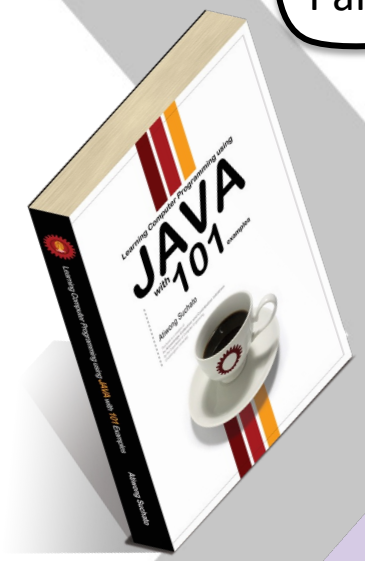
Arrays

Searching & Sorting Data

Chapter 9

Part I

Objectives



Chapter 9

Part I

To come in Part II

Students should:

- Be able to *define, initialize, and use* one-dimensional as well as multidimensional arrays correctly.
- Be able to use arrays as well as their elements as parameters to methods.
- Be able to write code to *sort* array elements in any orders desired.
- Be able to write code to *search for* elements in an array.
- Be able to use arrays in *problem solving* using computer programs.

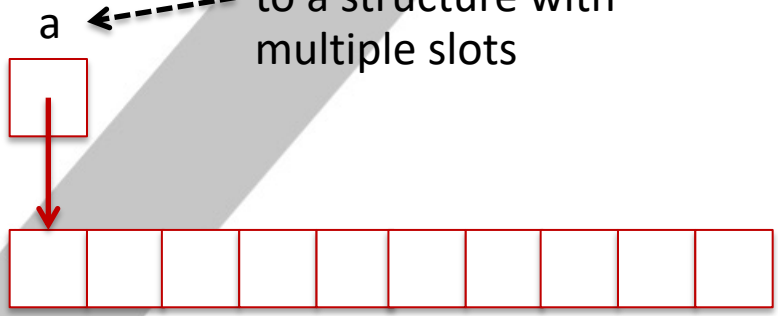
Requirement for a List of Values

CountDigitFrequency.java

- Write a program that counts the number (frequency) of each digit from 0 to 9 in a *String* entered by the user.

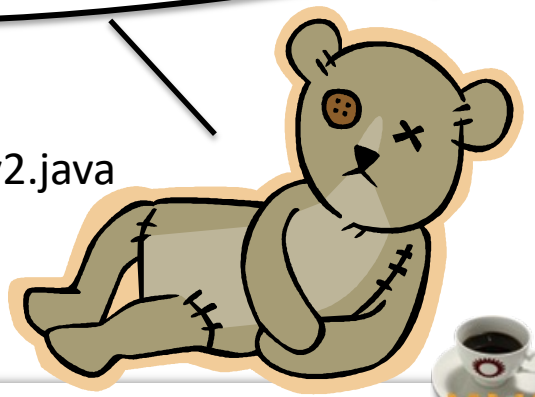
Array

A single identifier refers to a structure with multiple slots

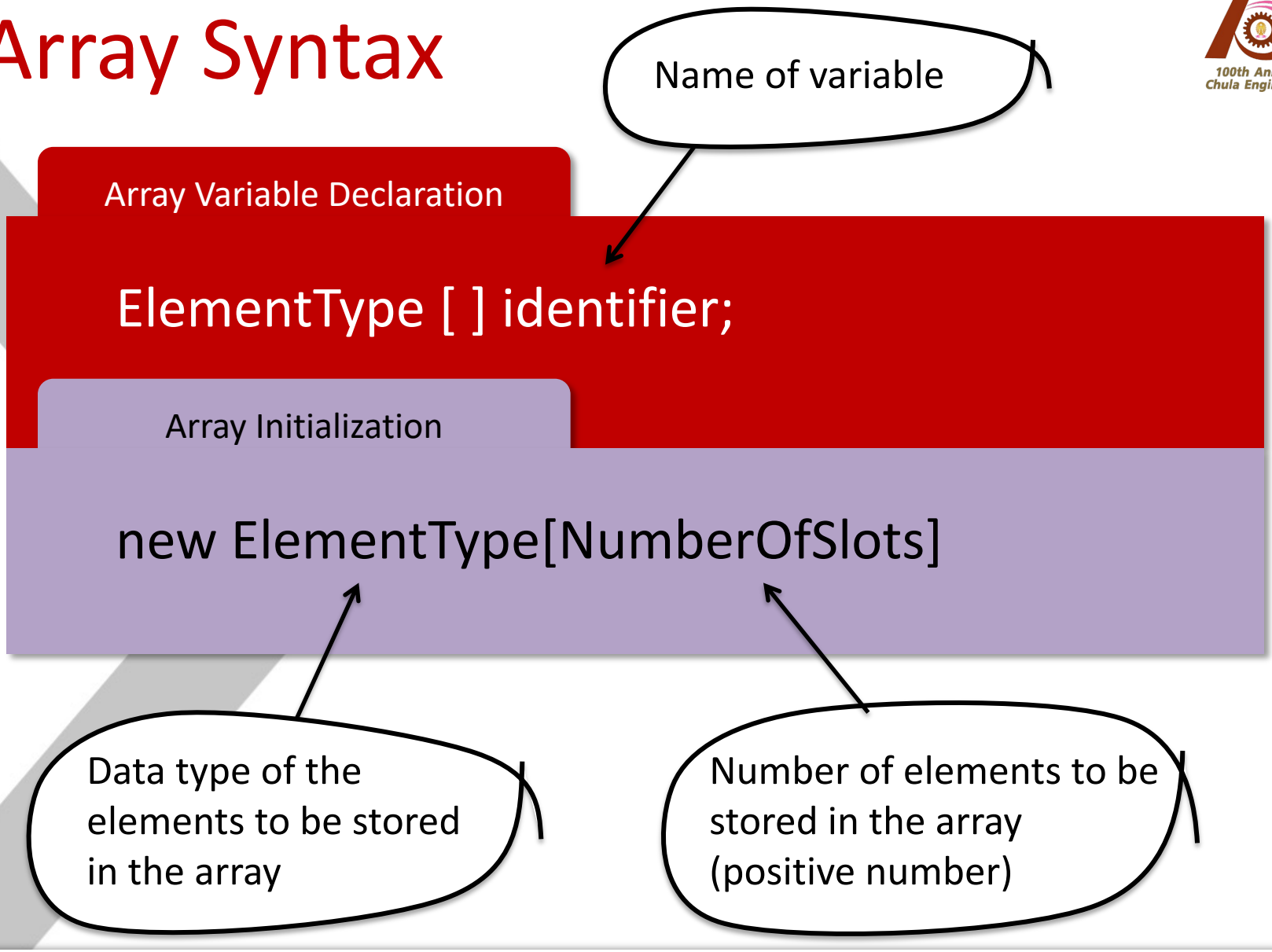


Now, let's use a single array to keep track of the frequencies of every digit.

CountDigitFrequency2.java

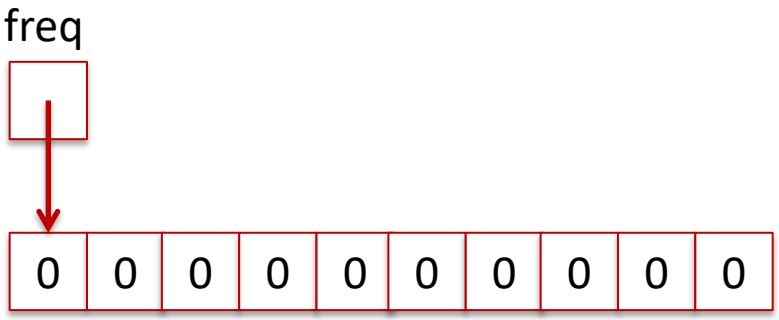


Array Syntax

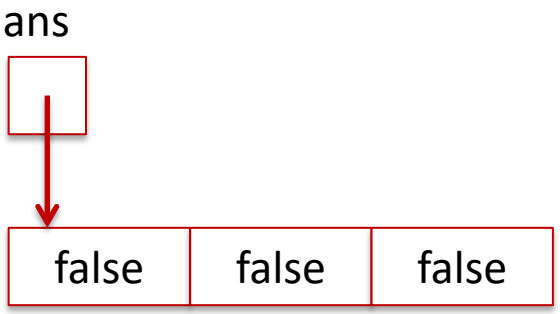


Array Declaration/Initialization

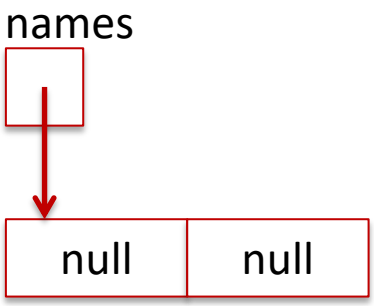

```
int [] freq;
freq = new int[10];
```




```
boolean [] ans;
ans = new boolean[3];
```




```
String [] names = new String[2];
```



Default Values for Elements

When an array is initialized, all elements are automatically set to default values, which are:

Zeros	false	null
for numeric primitive data types	for boolean	for non-primitive data types (classes)

null

- A Java constant.
- can be referenced with any non-primitive variables
- normally used as a representation of "nothingness"

null

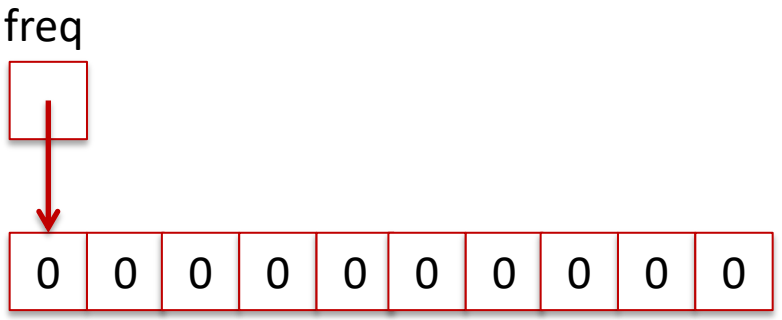
```
public class NullDemo{
    public static void main(String [] args){
        String [] s = new String[5];
        for(int i=0;i<s.length;i++){
            System.out.println("s["+i+"] = "+s[i]);
        }
        if(s[0]==null){
            System.out.println("Yes, s[0] is null.");
        }
        System.out.println("Its length is "+s[0].length());
    }
}
```

will throw NullPointerException

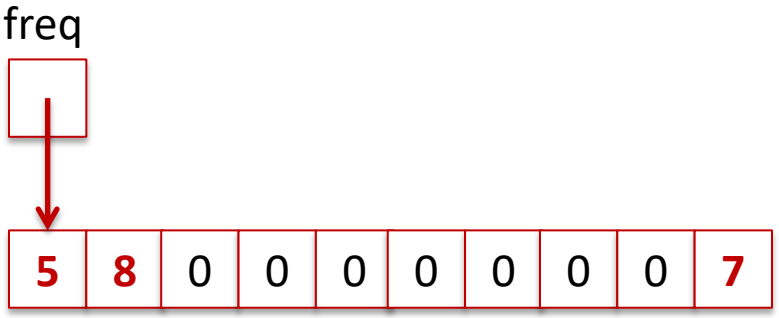
NullDemo.java

Accessing an Element


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



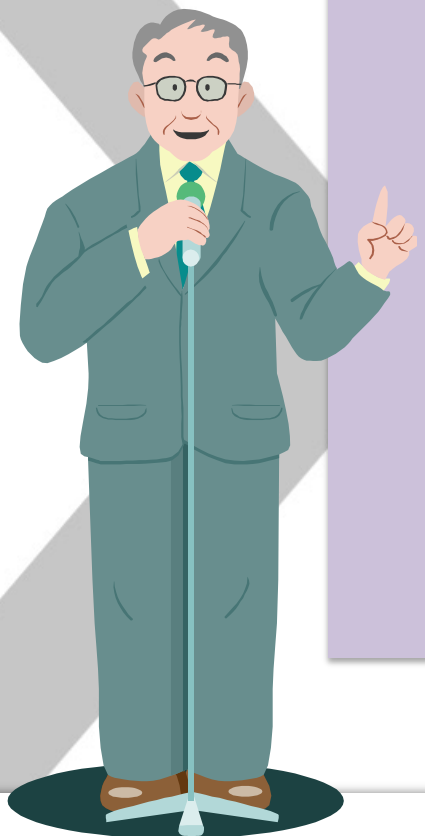

```
freq[0] = 5;
freq[1] = 8;
freq[9] = 7;
```




```
System.out.println(freq[1]);
```

8

Indices of an array



The first element in an array is at the index

0

NOT 1 !!

The last element of an array with n elements is at the index

n-1

NOT n !!

Initializer List

- Use to *explicitly assign value to each element* in the array at the time it is initialized.

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

String [] programs =
    {“ADME”, ”AERO”, “ICE”, ”NANO”};

boolean [] allTrue =
    {true, true, true};
```

instead of letting them be the default values at initialization of the array

!!!

```
int [] a;
a = {1,2,3,4,5}; // This is invalid.
```

Can ONLY be used in the same statement where the variable is declared.

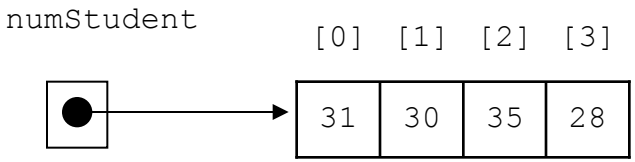
Array Traversal


```
String [] programs
                = {“NANO”, “ADME”, “ICE”, “AERO”};

for(int i=0;i<programs.length;i++){

   System.out.println(programs[i]);

}
```

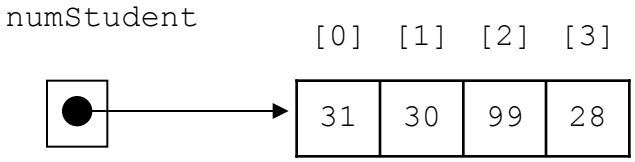
An attribute of an array whose value equals the number of slots.

Array Variables are Reference Variables

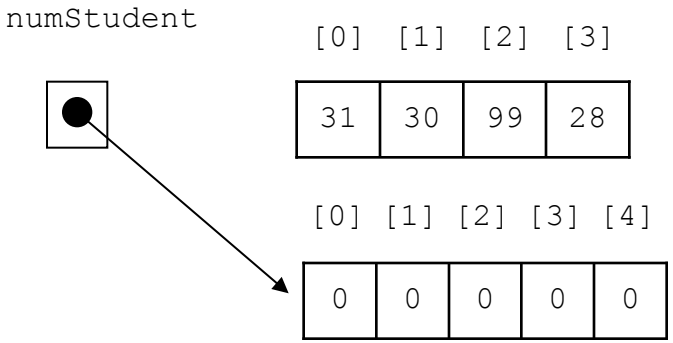

```
int [] numStudent =
            {31,30,35,28};
```




```
numStudent[2] = 99;
```



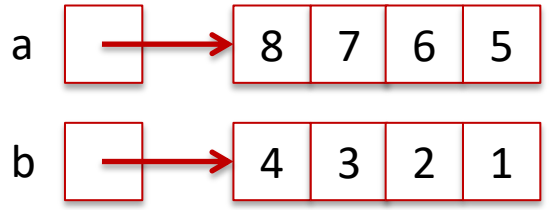

```
numStudent = new int[5];
```



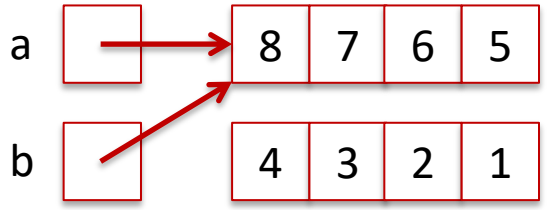
What is the output?

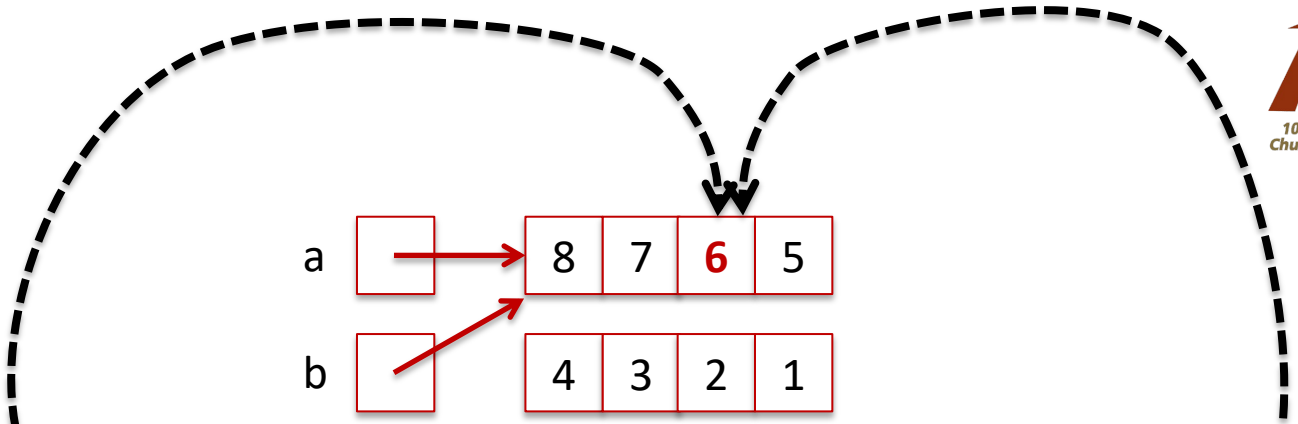
```
int [] a = {8,7,6,5};
int [] b = {4,3,2,1};
b = a;
a[1] = b[2] + a[2];
System.out.println(b[1]);
```

```
int [] a = {8,7,6,5};
int [] b = {4,3,2,1};
b = a;
a[1] = b[2] + a[2];
System.out.println(b[1]);
```

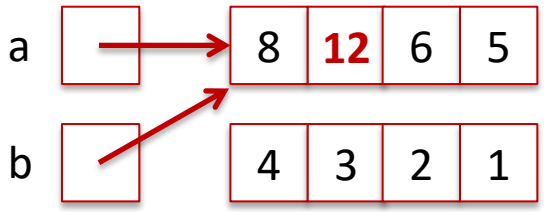



```
int [] a = {8,7,6,5};
int [] b = {4,3,2,1};
b = a;
a[1] = b[2] + a[2];
System.out.println(b[1]);
```






```
int [] a = {8,7,6,5};
int [] b = {4,3,2,1};
b = a;
a[1] = b[2] + a[2];
System.out.println(b[1]);
```



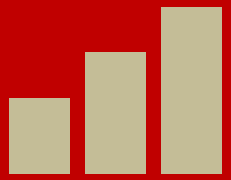

```
int [] a = {8,7,6,5};
int [] b = {4,3,2,1};
b = a;
a[1] = b[2] + a[2];
System.out.println(b[1]);
```

12

Important Algorithms

SEARCHING

looking for a specific value in an array



SORTING

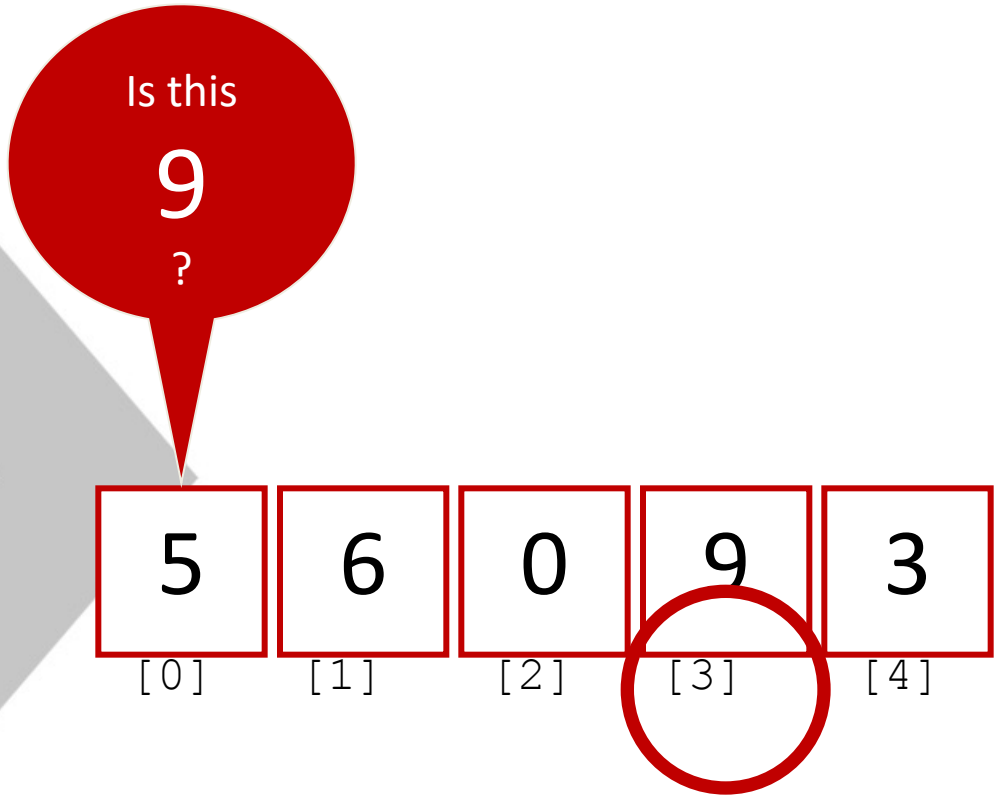
arranging elements in an array so that they are in specific orders

Sequential Search

“Checking every element **one at a time** **in sequence**”

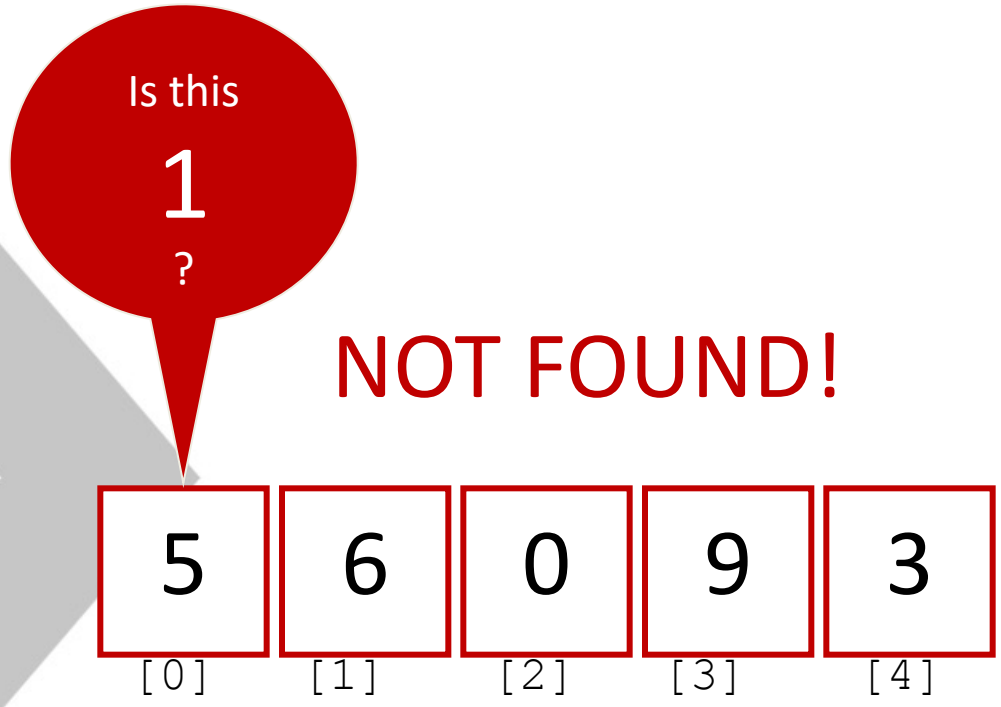
Sequential Search

Looking for the position that contains 9



Sequential Search

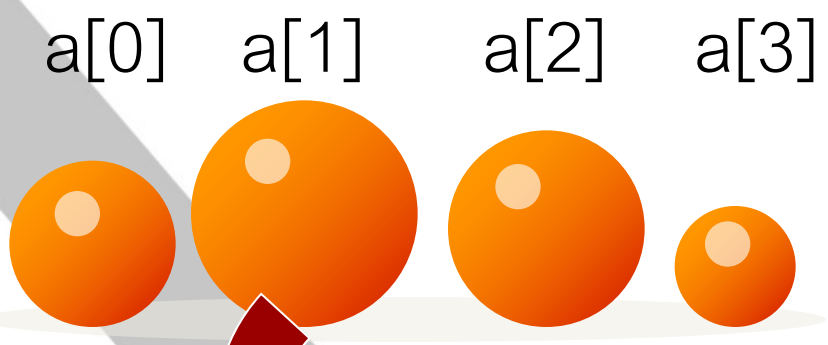
Looking for the position that contains 1



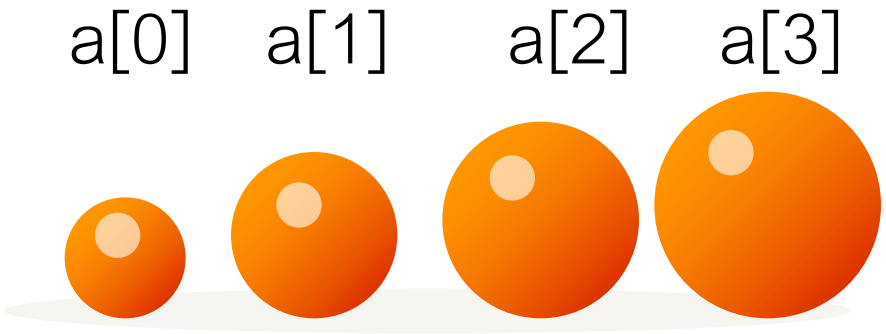
Sequential Search


```
public static int seqSearch(int [] a, int k){
   int i = 0;
   int len = a.length;
   while(i<len && a[i]!=k){
       i++;
    }
   if(i>=len) i=-1;
   return i;
}
```

Sorting



Sort Increasingly!



Selection Sort

"In the i^{th} iteration,
find the i^{th} smallest item
and
place it in the correct position."

or the i^{th} biggest

How many iteration is needed to sort **N** elements?

Selection Sort

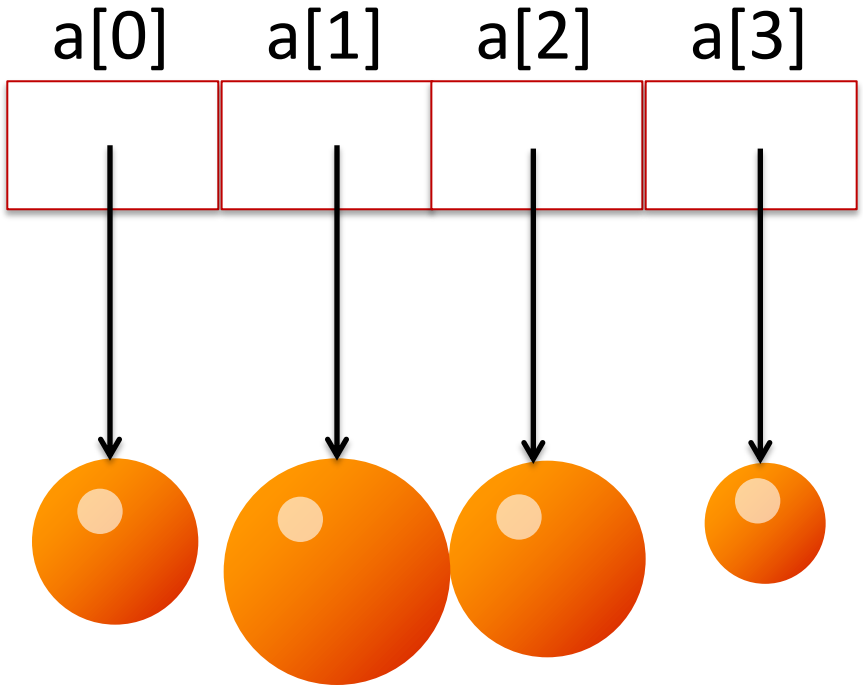
It #

1

Find the (1st) smallest item

Place it at a[0]

The correct position



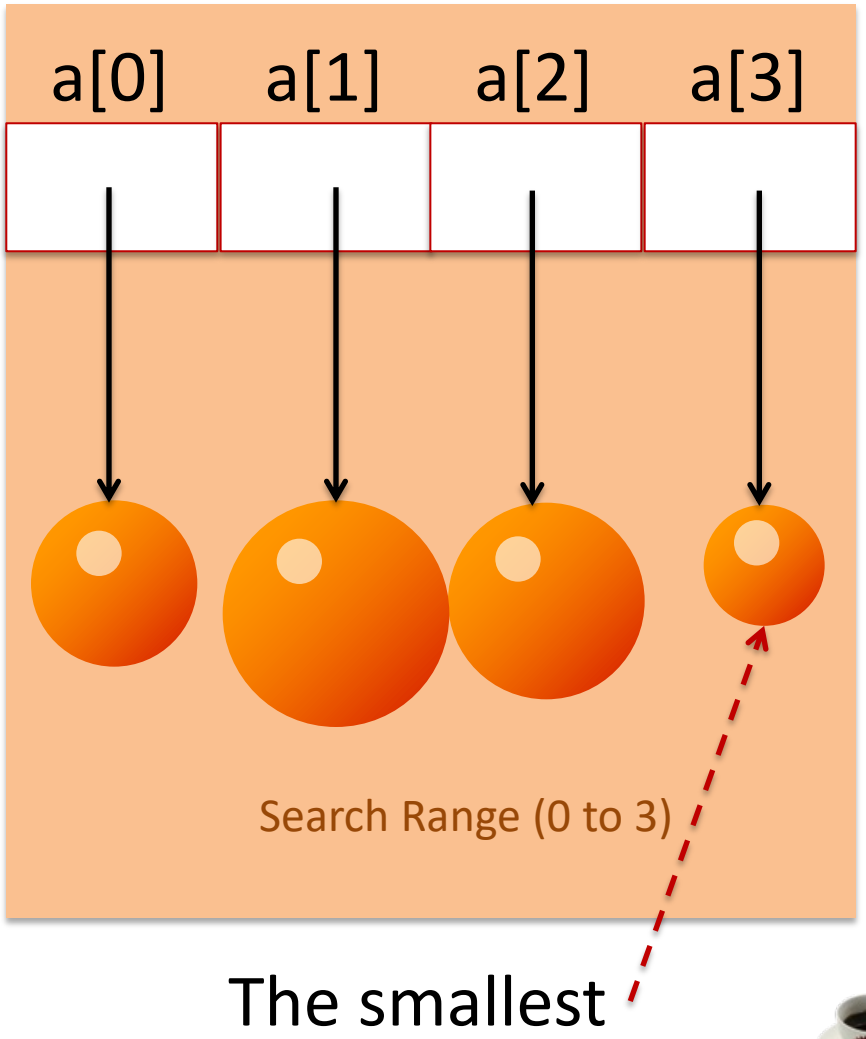
Selection Sort

It #

1

Find the (1st) smallest item

Place it at a[0]



Selection Sort

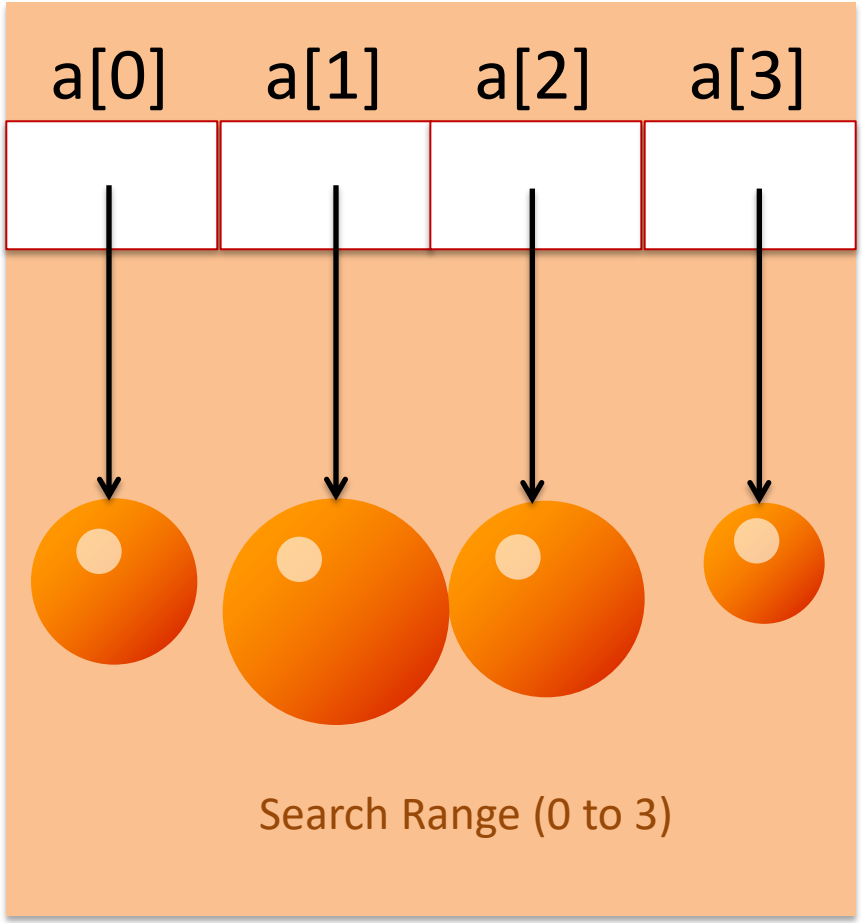
It #

1

Find the (1st) smallest item

Place it at a[0]

Swap element between a[0] and a[3]



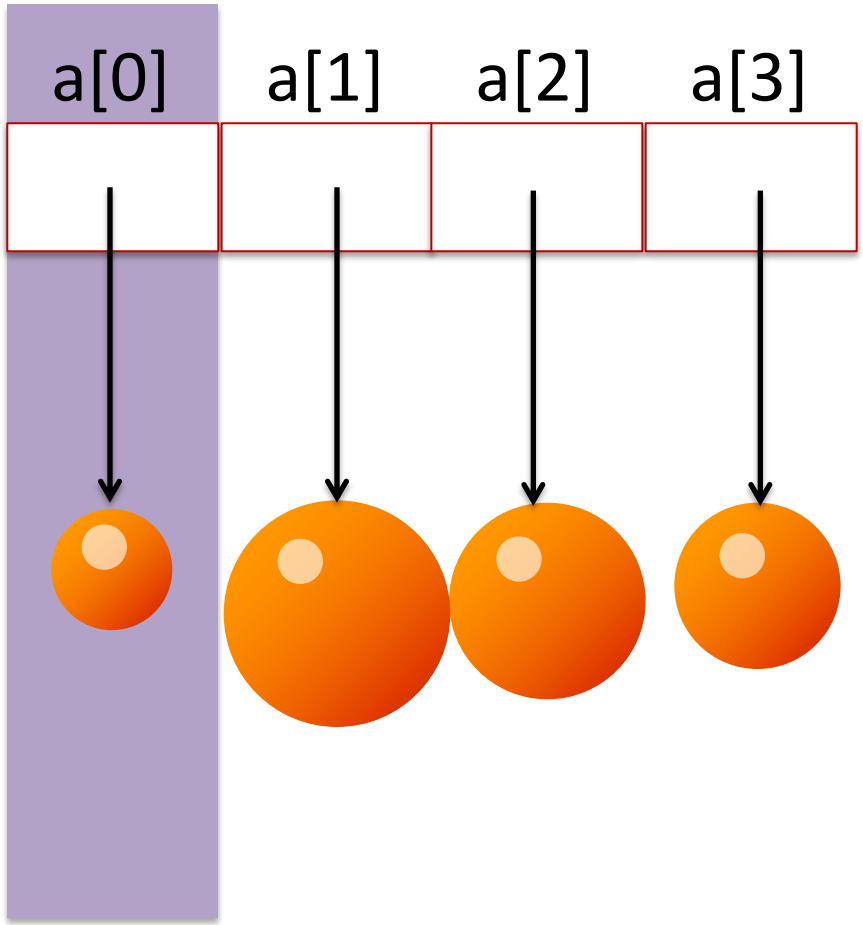
Selection Sort

It #

1

Find the (1st) smallest item

Place it at a[0]



Iteration # 1 is finished with a[0] sorted

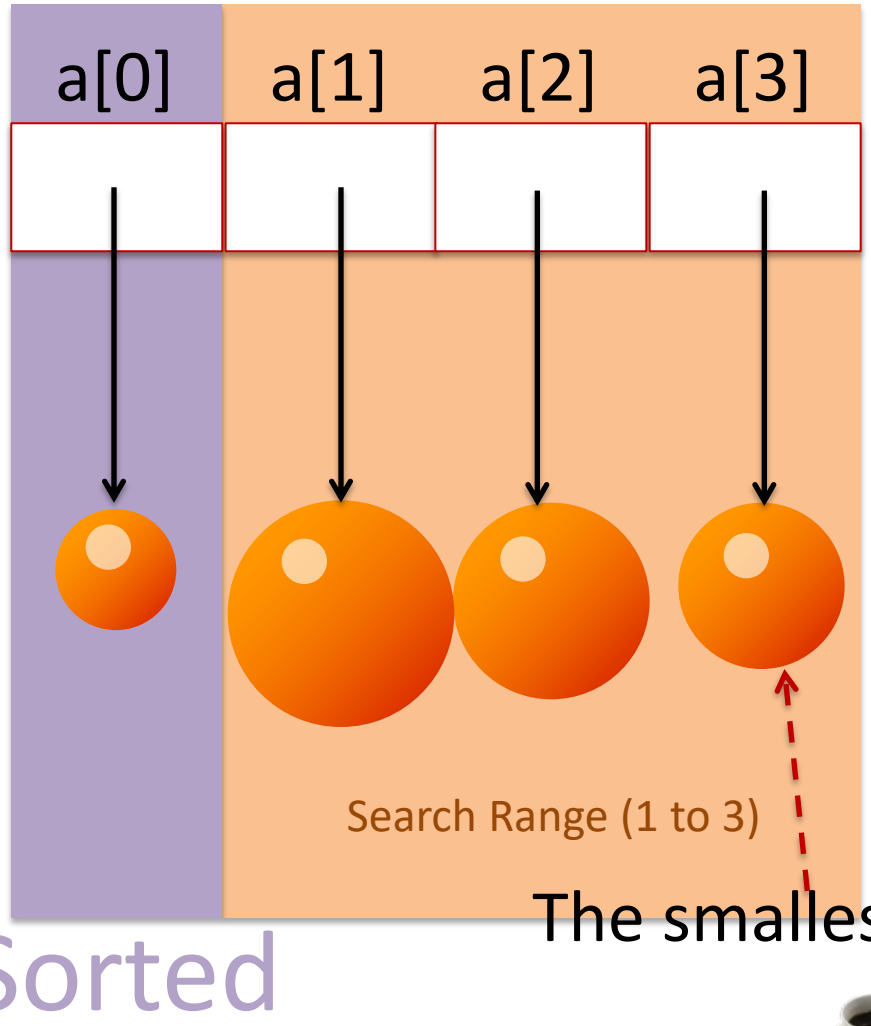
Selection Sort

It #

2

Find the 2nd smallest item

Place it at a[1]



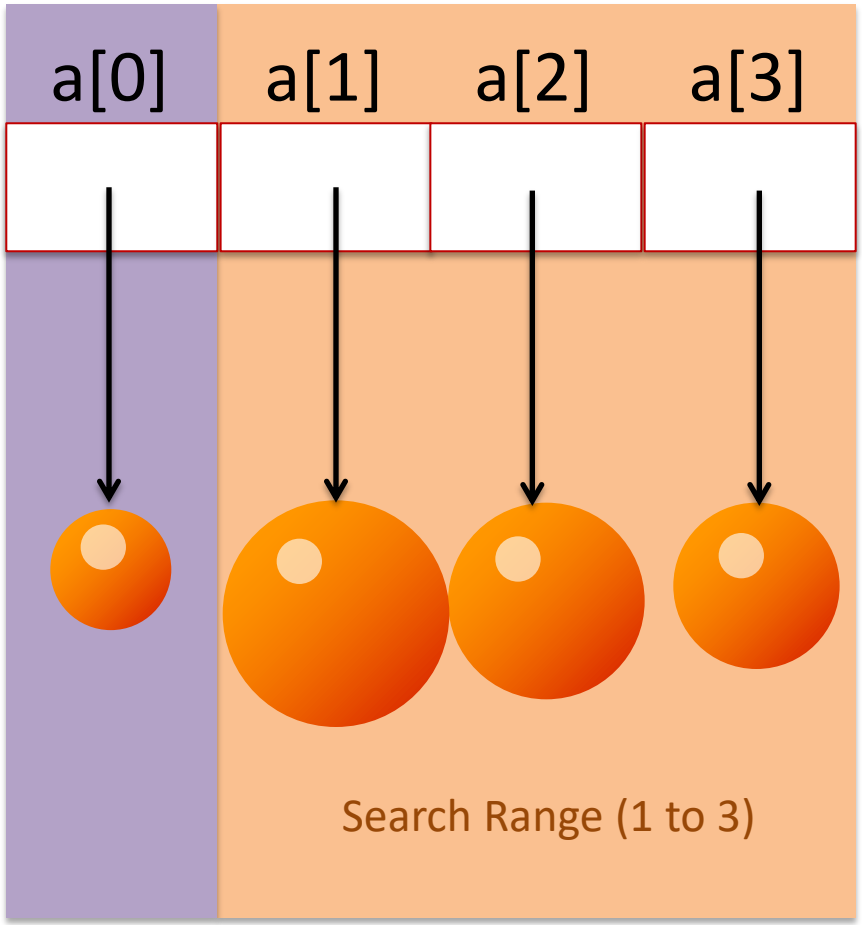
Selection Sort

It #

2

Find the 2nd smallest item

Place it at a[1]



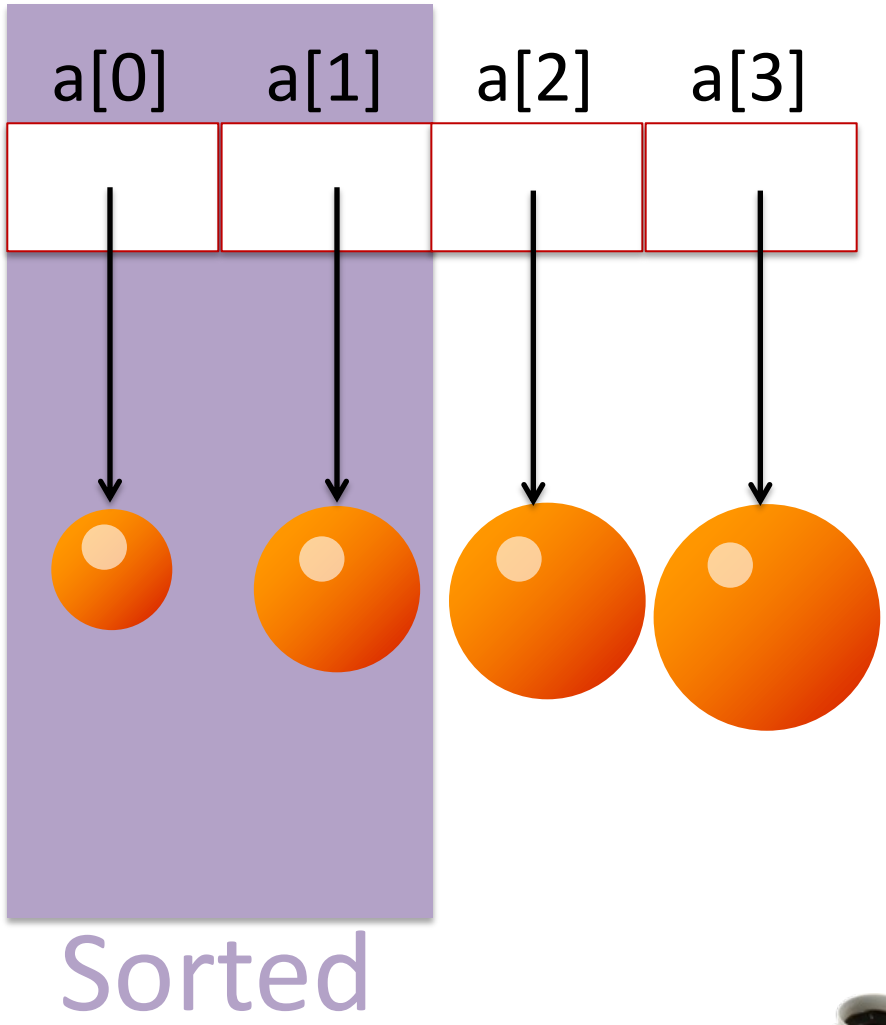
Selection Sort

It #

2

Find the 2nd smallest item

Place it at a[1]



Iteration # 2 is finished with a[0]-a[1] sorted

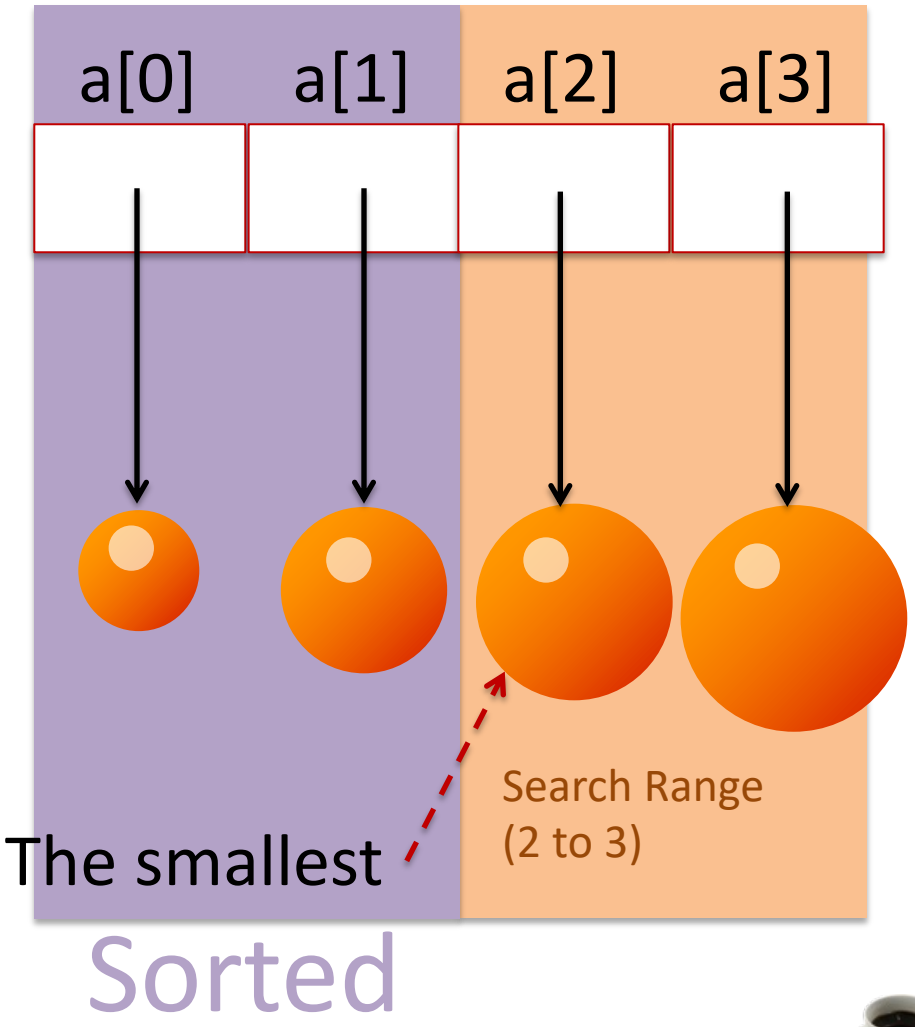
Selection Sort

It #

3

Find the 3rd smallest item

Place it at a[2]



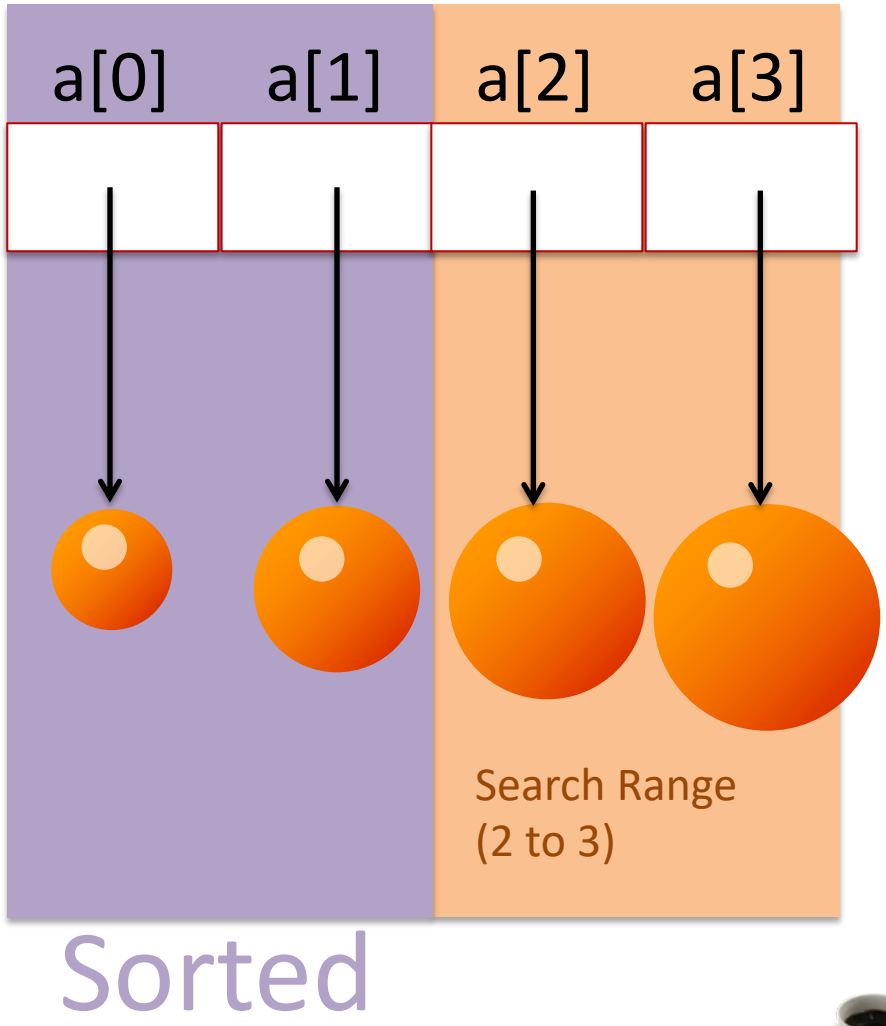
Selection Sort

It #

3

Find the 3rd smallest item

Place it at a[2]



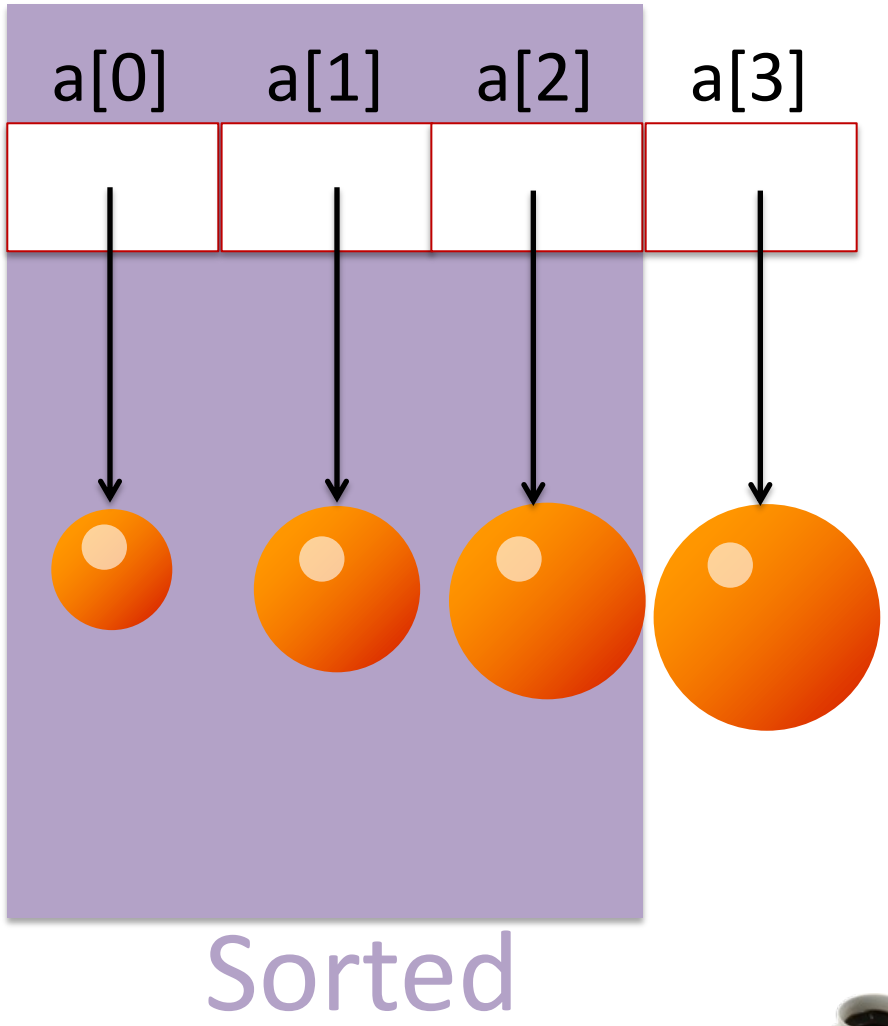
Selection Sort

It #

3

Find the 3rd smallest item

Place it at a[2]



Iteration # 3 is finished with a[0]-a[2] sorted

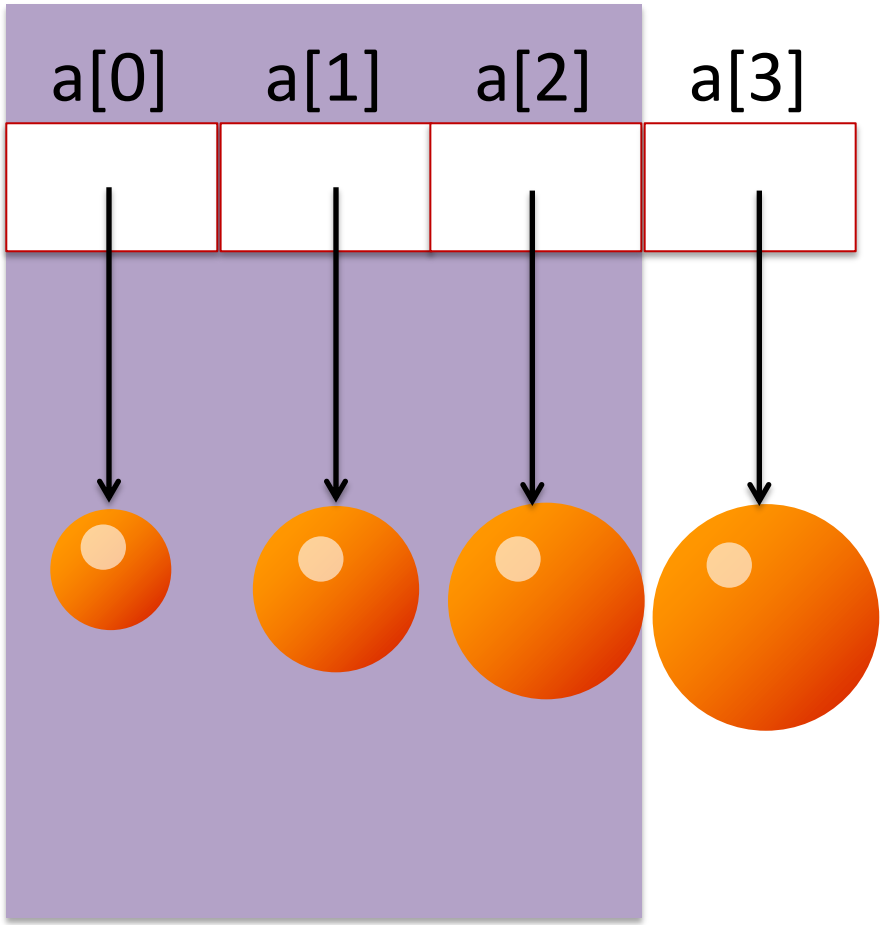
Selection Sort

It #

3

Find the 3rd smallest item

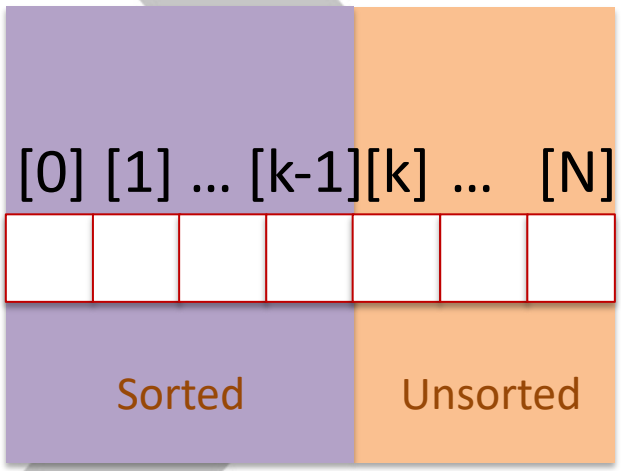
Place it at a[2]



Selection Sort Algorithm

Sorting N elements

Let *k* be the first index the unsorted array.

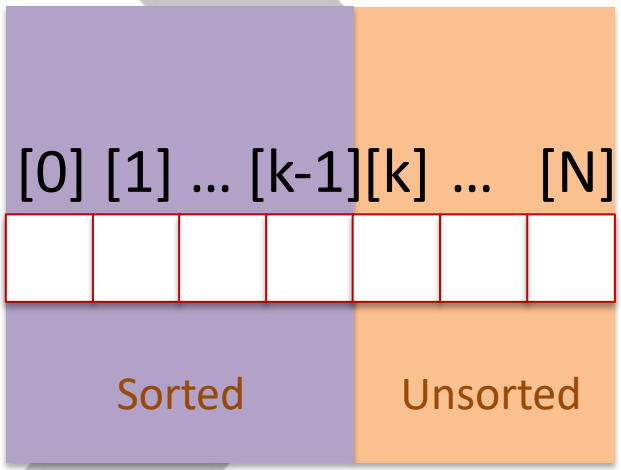


Selection Sort Algorithm

Sorting N elements

Let *k* be the first index the unsorted array.

How does the value of k change during the entire sorting procedure?

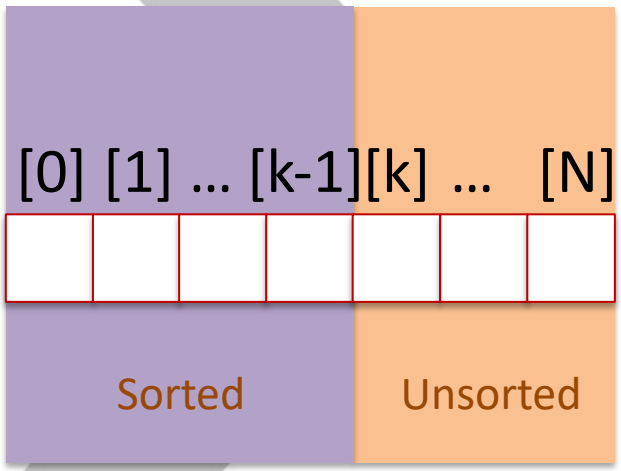


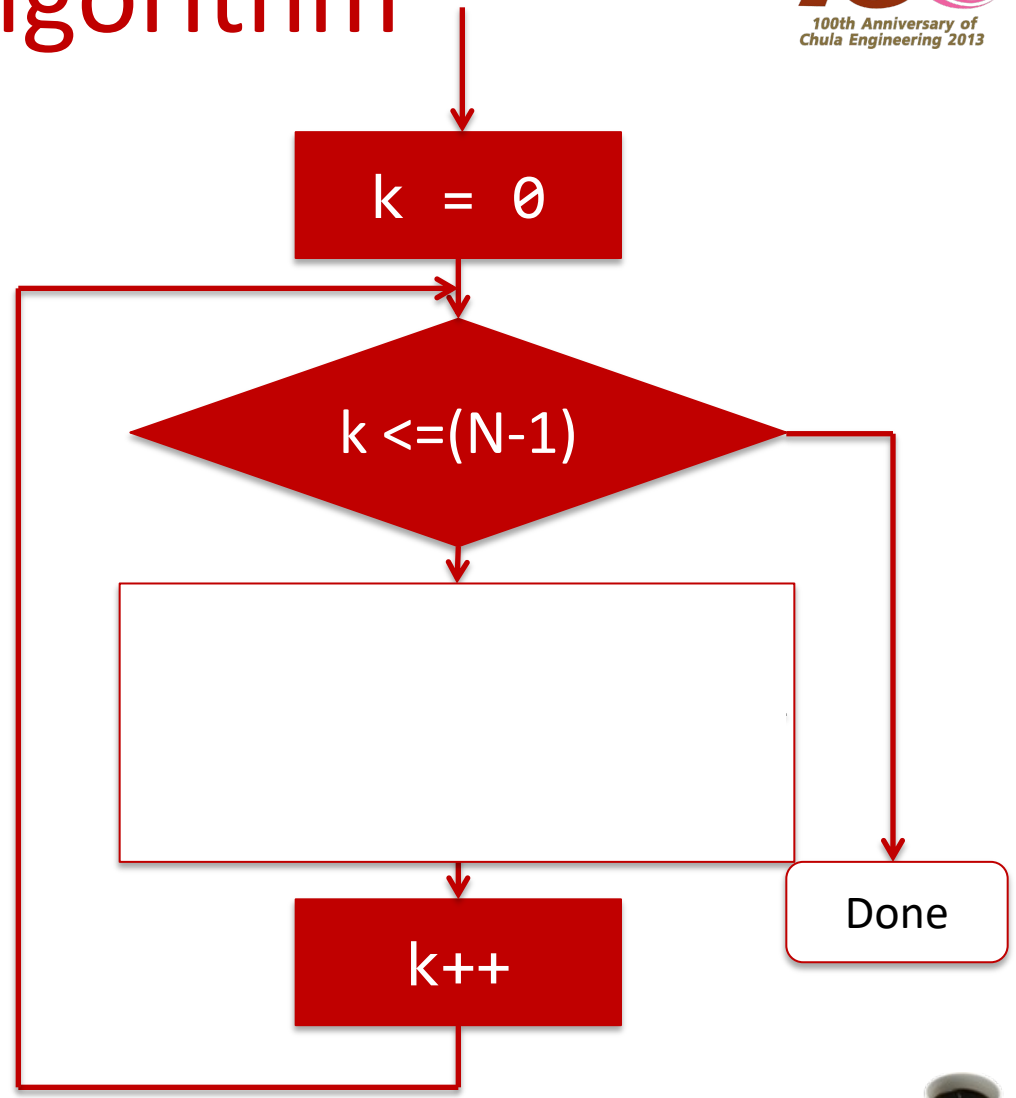
	k is
At the beginning,	
When the sorting is completed	

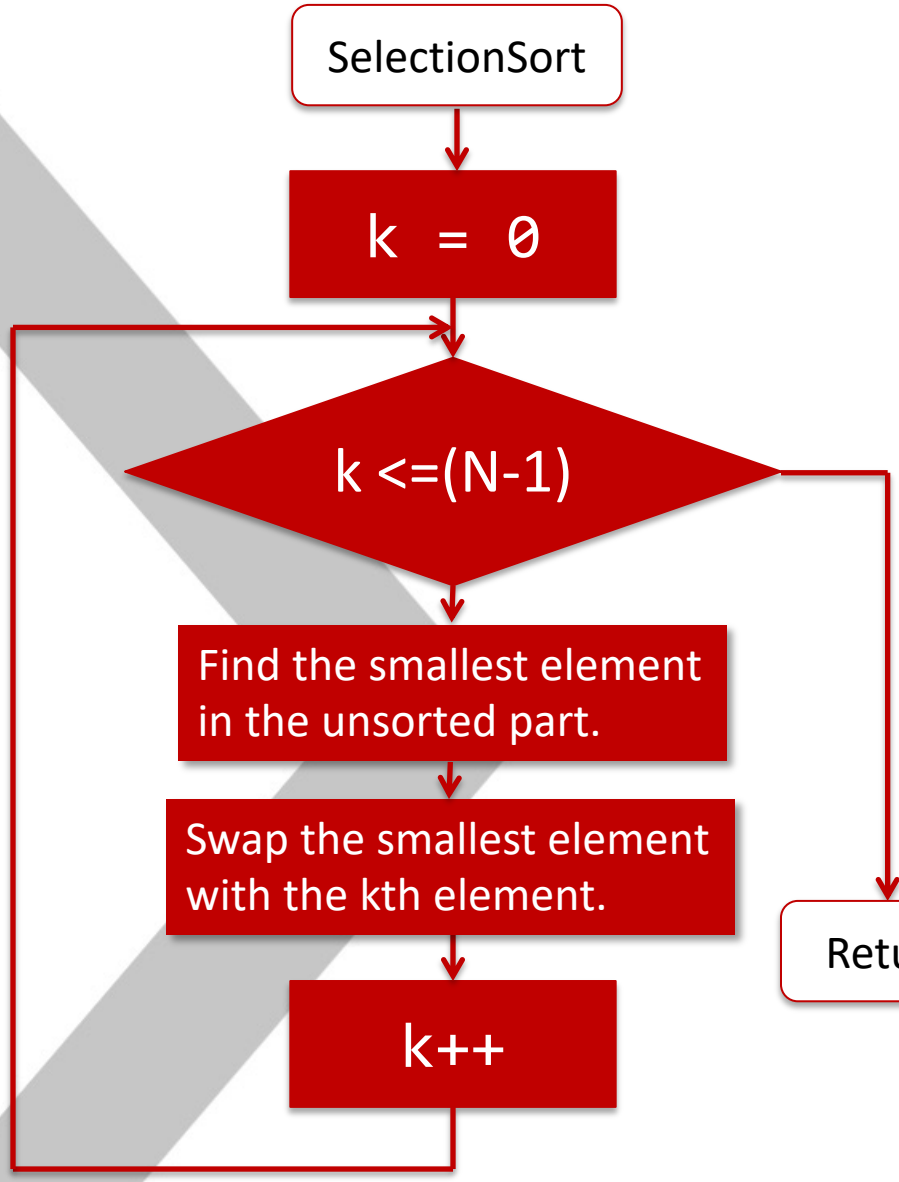
Selection Sort Algorithm

Sorting N elements

Let *k* be the first index the unsorted array.








```
public static void
    selectionSort(double [] a){

    // Try implementing
    // the procedure
    // in the flowchart

}
```

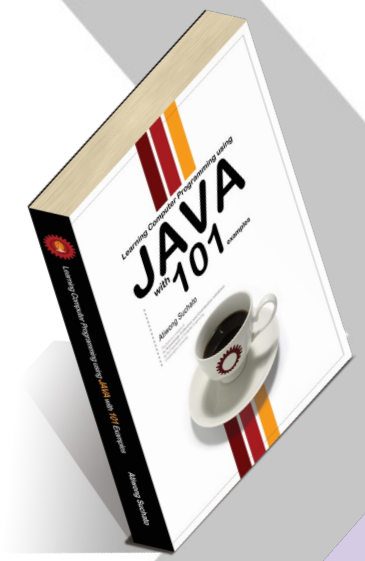
Chapter 9

Part II

Arrays

Multi-dimensional Arrays

Objectives



Chapter 9

Part II

Students should:

- Be able to *define, initialize, and use* one-dimensional as well as multidimensional arrays correctly.
- Be able to use arrays as well as their elements as parameters to methods.
- Be able to write code to *sort* array elements in any orders desired.
- Be able to write code to *search for* elements in an array.
- Be able to use arrays in *problem solving* using computer programs.

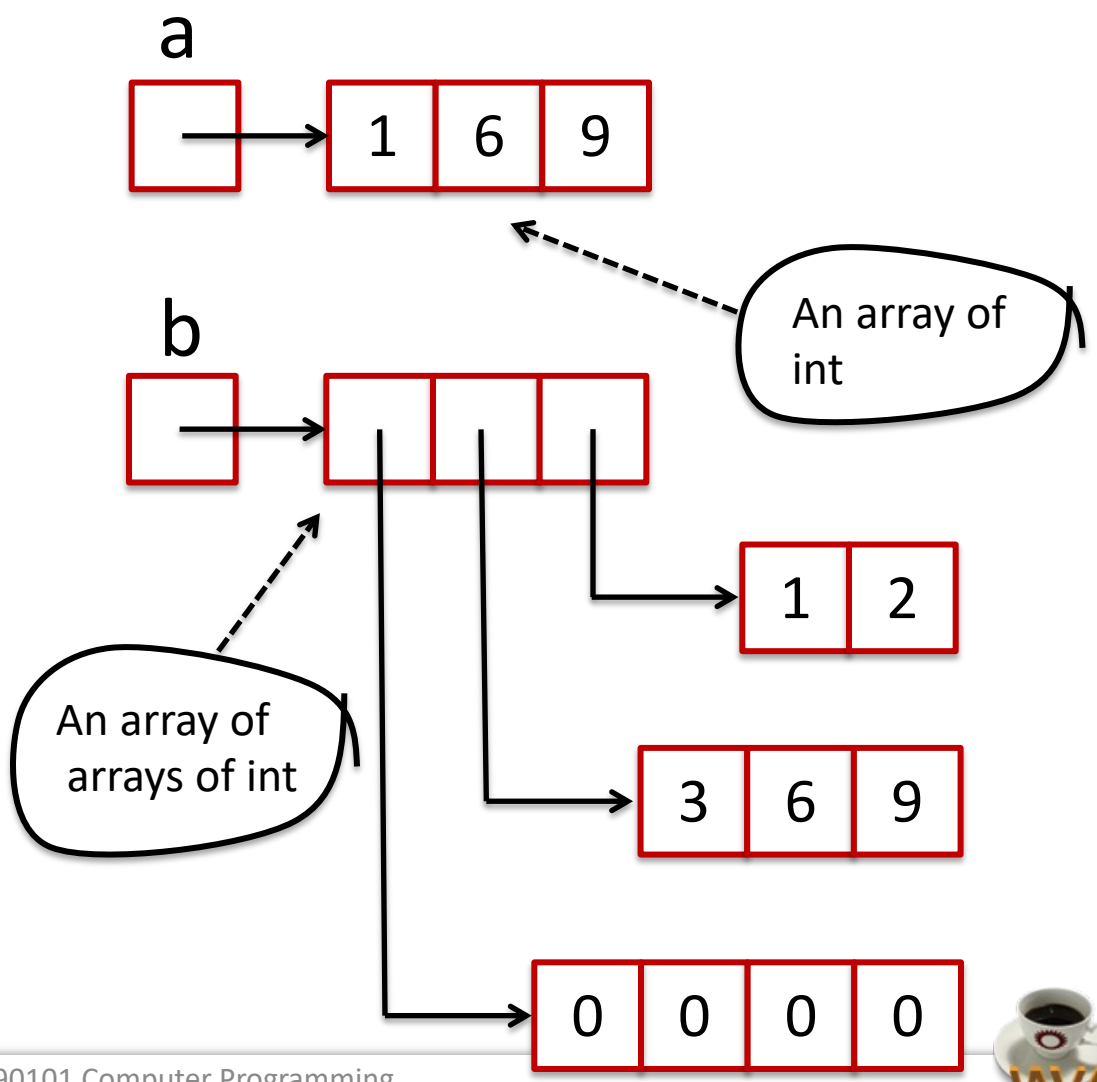
Multi-dimensional Arrays

They are just arrays of which

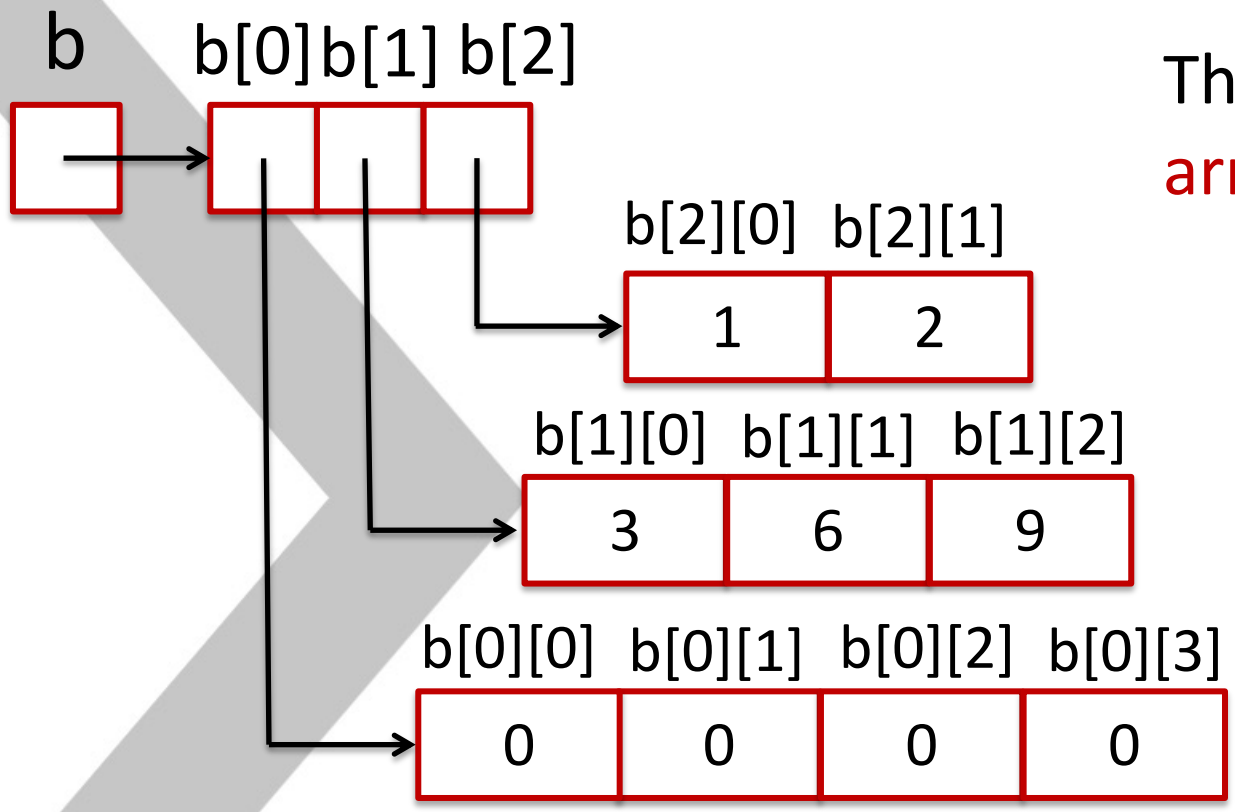
Each Element

is also

An Array



Element Indices



The type of b is
array of arrays of int

Two-dimensional
Array of int

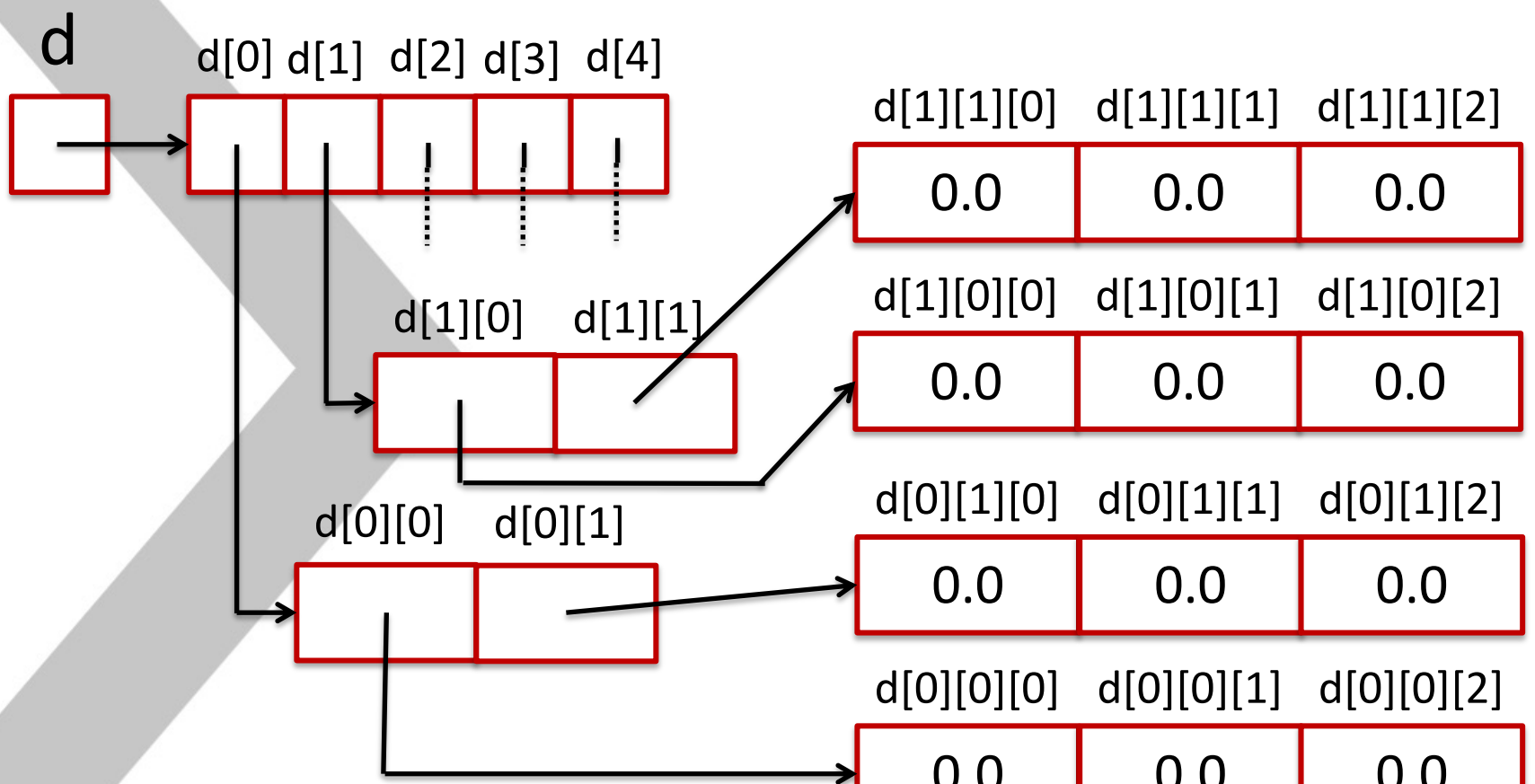
Variable Declaration/Initialization


```
int [][] lookUpTable;
String [][] classRoster;
double [][][] data;

lookUpTable = new int[2][3];
classRoster = new String[5][10];
data = new double[5][2][3];
```

array of arrays of arrays of double

Visualizing Multi-dim Array

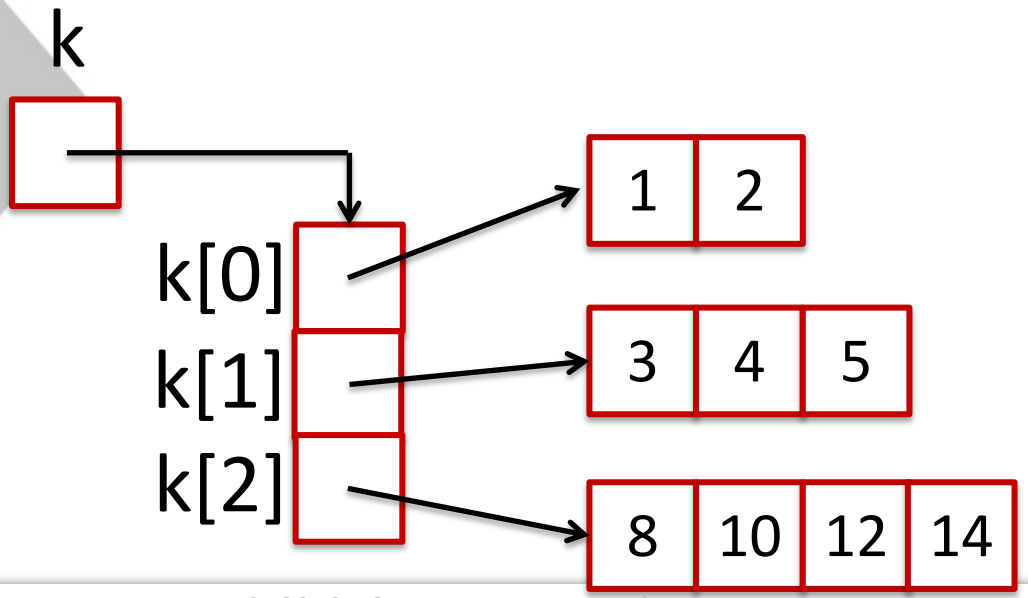

```
double [][][] data = new double[5][2][3];
```



Initializer List

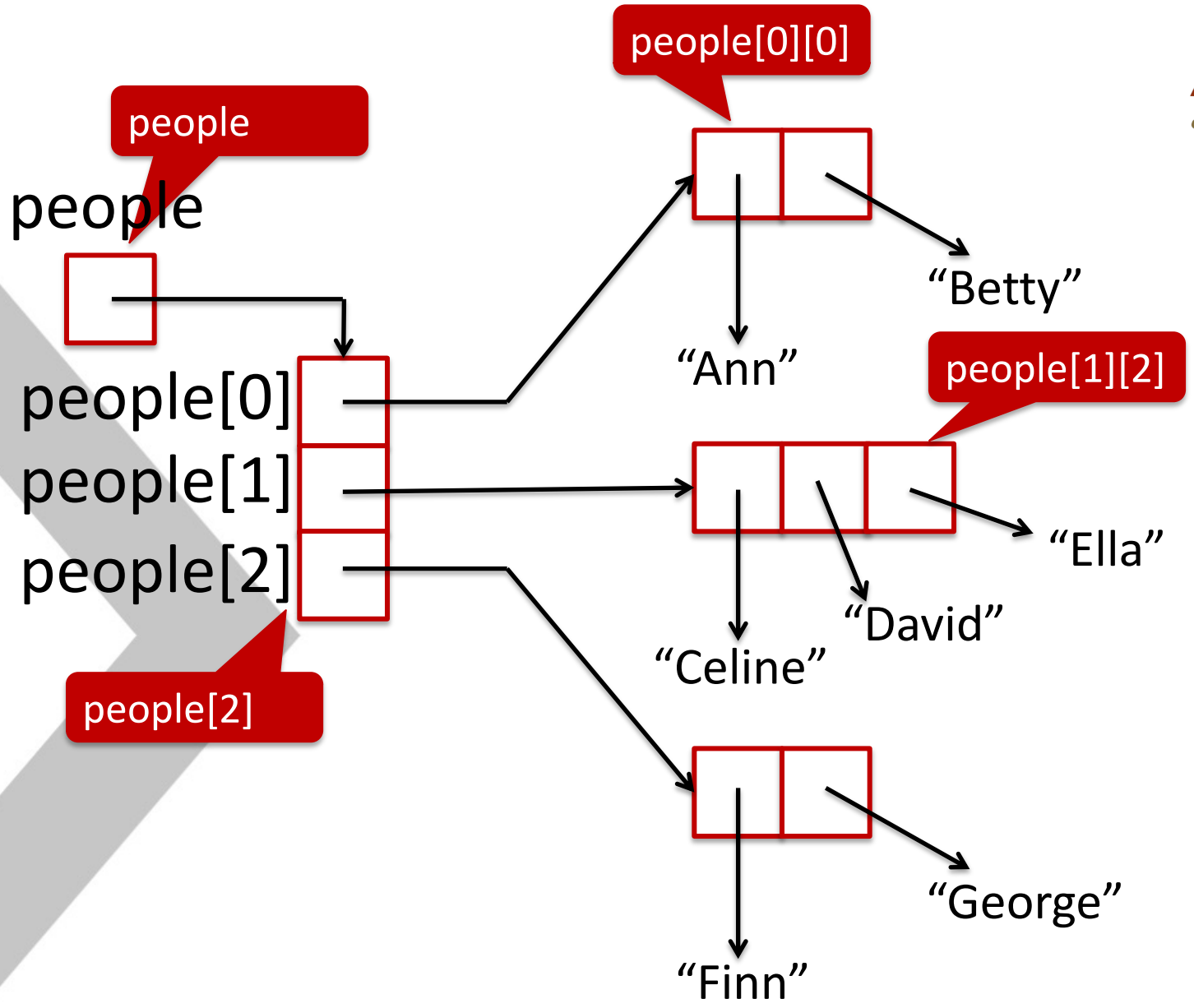
```
int [][] k
  ={{1,2},{3,4,5},{8,10,12,14}};
```

k[0] k[1] k[2]



Example

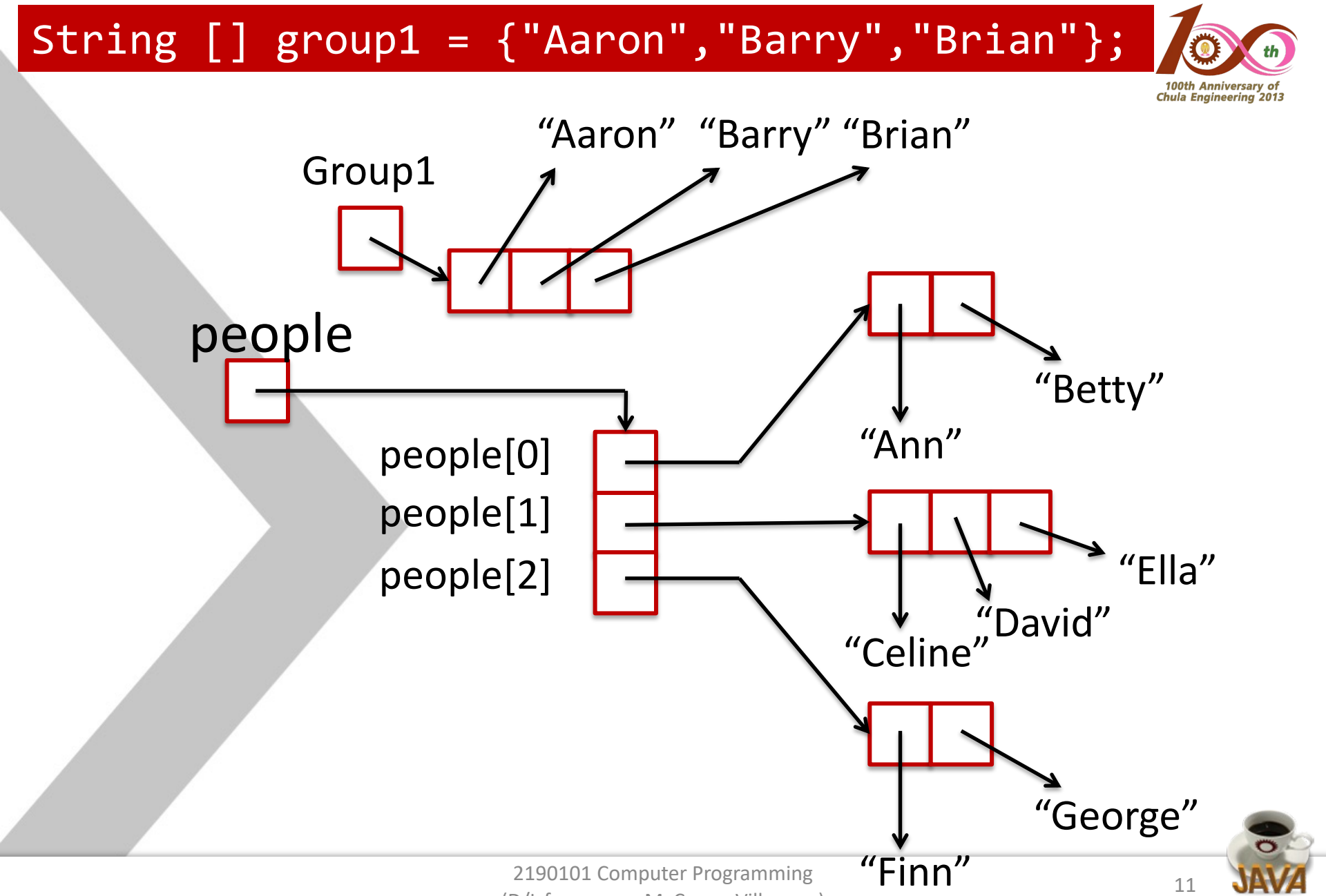
```
public class MultiDimArrayPeople1 {
   public static void main(String [] args) {
      String [][] people = {
            {"Ann","Betty"},
            {"Celine","David","Ella"},
            {"Finn","George"}
      };
      System.out.println(people[0][0]);
      System.out.println(people[1][2]);
      System.out.println(people[2].length);
      System.out.println(people.length);
   }
}
```


people[0][0]
people
people
"Betty"
"Ann"
people[0]
people[1][2]
people[1]
"Ella"
people[2]
"David"
"Celine"
people[2]
"George"
"Finn"

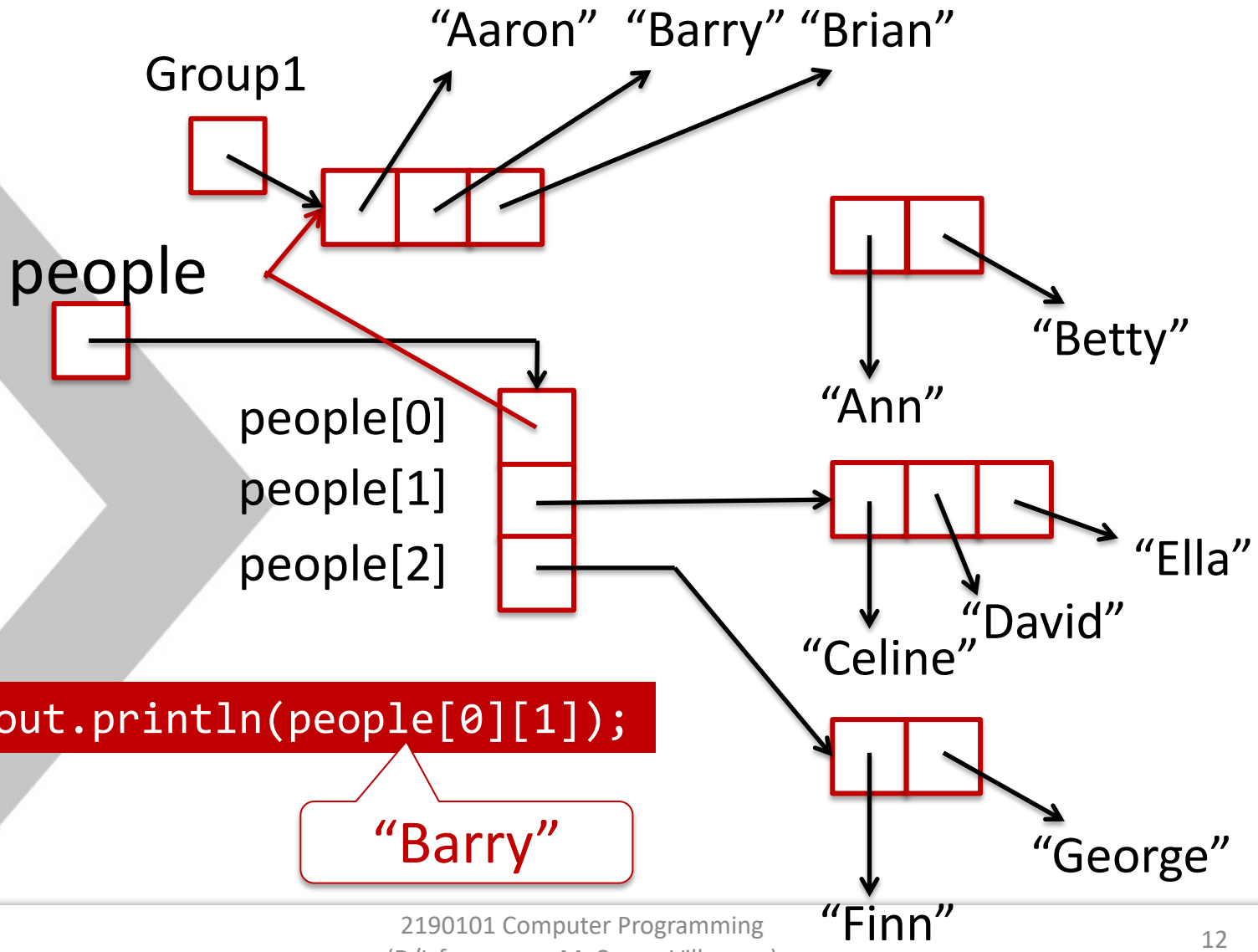
Example

```
public class MultiDimArrayPeople2 {
   public static void main(String [] args) {
      String [][] people = {
            {"Ann","Betty"},
            {"Celine","David","Ella"},
            {"Finn","George"}
      };
      String [] group1 = {"Aaron","Barry","Brian"};
      people[0] = group1;
      System.out.println(people[0][1]);
   }
}
```

```
String [] group1 = {"Aaron","Barry","Brian"};
```



```
people[0] = group1;
```




```
System.out.println(people[0][1]);
```

"Barry"

Multi-dimension traversal (1)


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

for (int i = 0; i < d.length; i++) {
   for (int j = 0; j < d[i].length; j++) {
      for (int k = 0; k < d[i][j].length; k++) {
         // you can access element i,j,k
         // d[i][j][k]
      }
   }
}
```

ArrayTravelsal.java

Multi-dimension traversal (2)


```
int [][][] array3D = {
   { {1, 3, 2}, {5, 4}, {7, 6, 8, 9} },
   { {3, 2, 1}, {4, 3, 1, 5} } };

for (int[][] array2D : array3D) {
   for (int [] array1D : array2D) {
      for (int data: array1D) {
         // you can access each element
         // in array as data
      }
   }
}
```

ArrayTravelsalForEach.java

Hands-on Experiment

- Practice using multi-dim arrays through simple image processing tasks.

RGB-colored Pixels

Red Component: 0 - 255

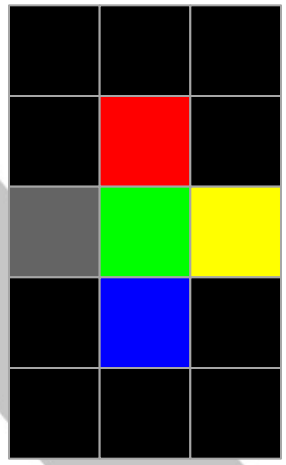
Green Component: 0 - 255

Blue Component: 0 - 255

Hands-on Experiment

- An image of `w` pixel-wide and `h` pixel-high is represented using an `int[w][h][3]` array.
- where
  - The int value at `[i][j][0]` is the "Red" component of the pixel at `(i,j)`.
  - The int value at `[i][j][1]` is the "Green" component of the pixel at `(i,j)`.
  - The int value at `[i][j][2]` is the "Blue" component of the pixel at `(i,j)`.

```
int[][][] image1 = new int[3][5][3];
```

Hands-on Experiment

- Given with:

Java101ImageUtil

```
static int [ ][ ][ ] getRGBArrayFromFile()
static void showViewer(int [ ][ ][ ] rgb,title)
static void showViewer(
                int [ ][ ][ ] rgb1,
                int [ ][ ][ ] rgb2,
                title)
static void showViewer(int [ ][ ][ ][ ] rgbs,title)
```

Hands-on Experiment


```
static int [ ][ ][ ] getRGBArrayFromFile()
```

Once called, the method show a file chooser dialog box for the user to pick an image file of either the .jpg or .gif format. The method tries to open the file and returns the 3D array containing the RGB values of the pixels in the image. It returns null if the user clicks the cancel button of the dialog box.

Hands-on Experiment


```
static void showViewer(
        int [ ][ ][ ] rgb,title)
```

The method shows a GUI window with the specified `title` and a panel hosting an images whose pixels are corresponding to the RGB values in `rgb`.

From Ch. 8 Method
- Method Overloading
- Example: Ch. 8 Page 38 – numericAdd
- How many *showViewer?*

Hands-on Experiment


```
static void showViewer(
        int [ ][ ][ ] rgb1,
        int [ ][ ][ ] rgb2,
        title)
```

The method shows a GUI window with the specified `title` and two panels hosting images. The first panel shows an image corresponding to the RGB values in `rgb1` while the other panel shows an image corresponding to the RGB values in `rgb2`.

2 panels; 1 image for each panel

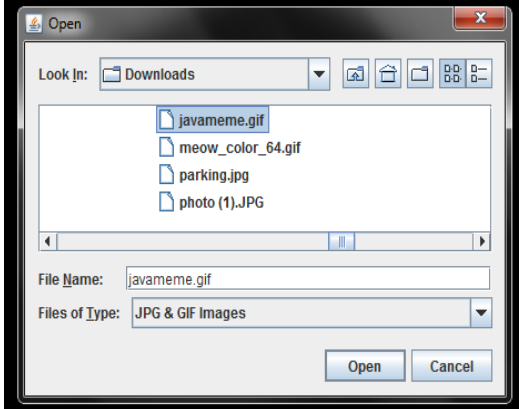
Hands-on Experiment


```
static void showViewer(
        int [ ][ ][ ][ ] rgbs,title)
```

The method shows a GUI window with the specified `title` and a series of panels hosting images each of which is corresponding to the RGB values in `rgbs[i]`, where `i` is from `0` to `rgbs.length()`.

Hands-on Experiment


```
>java Java101ImageUtilExample
```




```
>java Java101ImageUtilExample
Pick how to show the images
--------------------------------
1 : Show only the loaded file
2 : Show the loaded file + its
3 : Show both + a red patch
--------------------------------
>>
```

```
>java Java101ImageUtilExample
Pic
---
1
2
3
---
>>
Ima
Image size=(200,245)
Close the image viewer to exit
```

Chapter 11

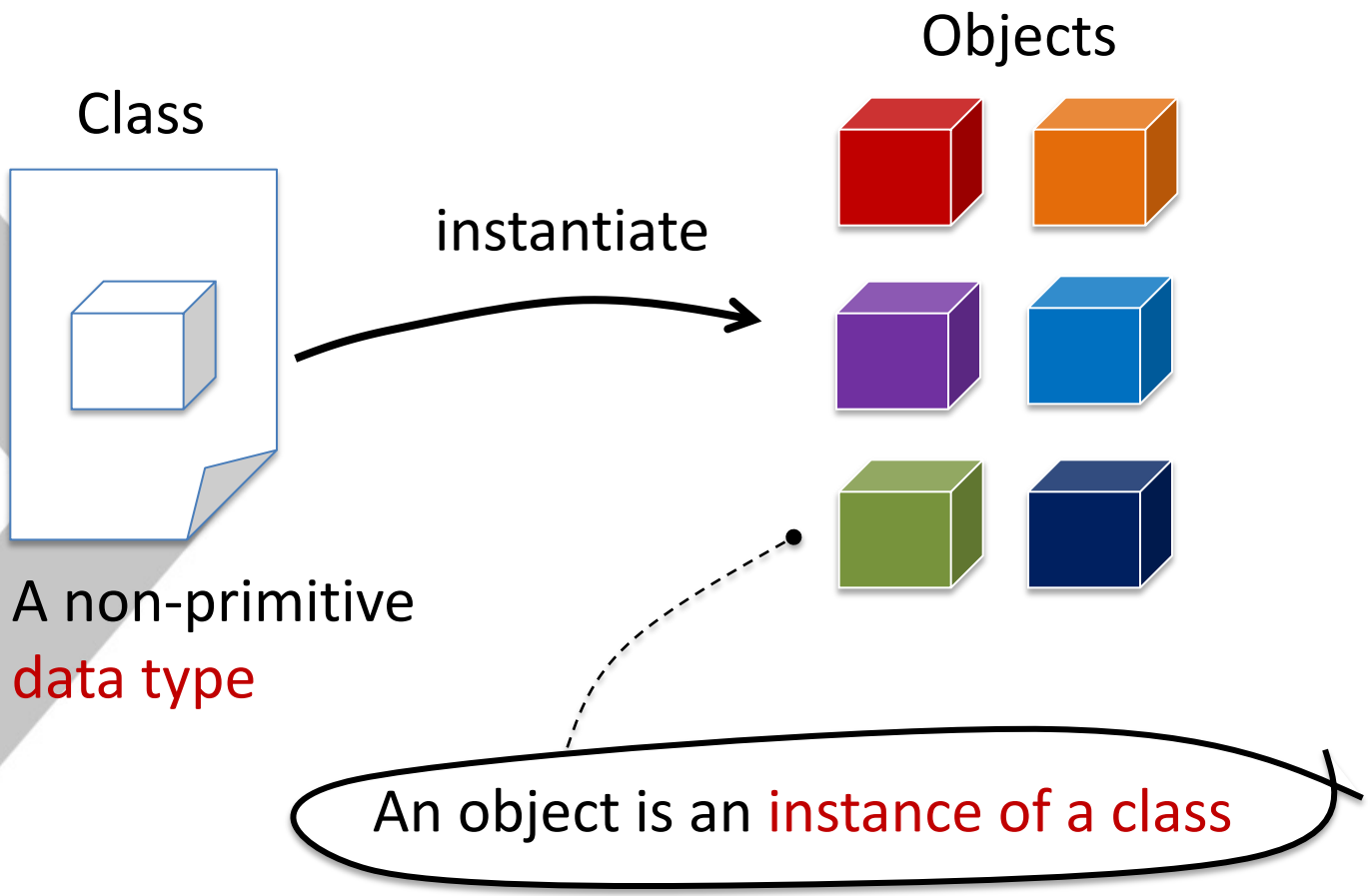
Creating Classes

Objectives

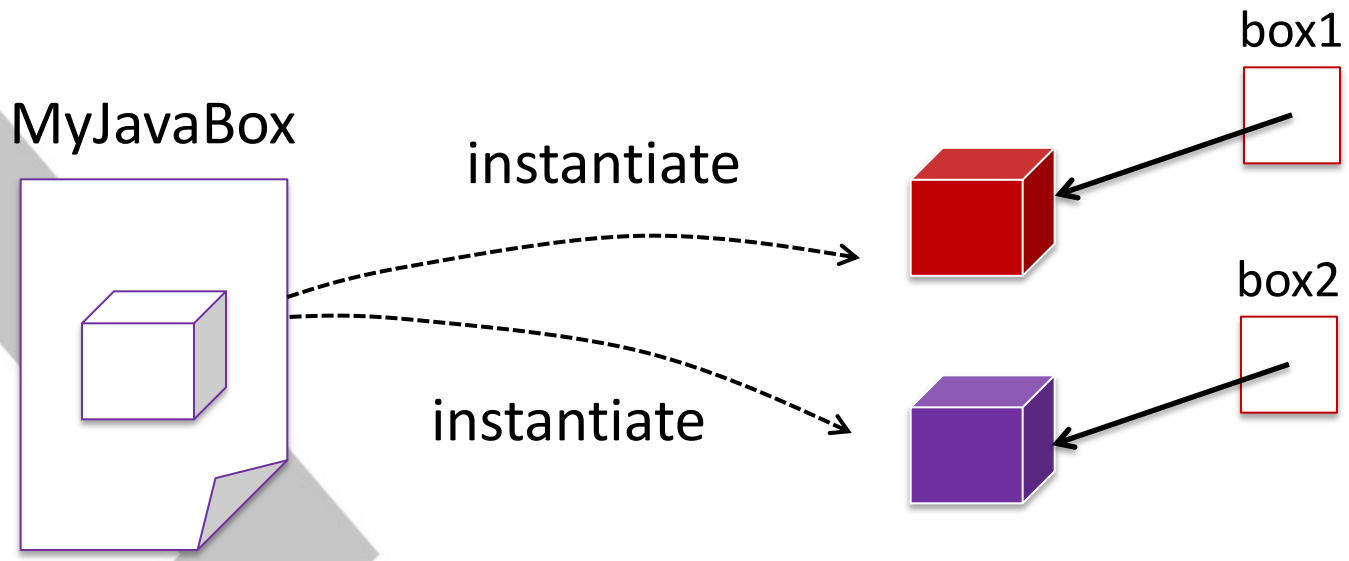
Students should:

- Recall the meaning of classes and objects in Java
- Know the components in the definition of a Java class
- Understand how constructors work
- Be able to create class and object methods
- Be able to create new Java classes and use them correctly

Classes and Objects



Classes and Objects




```
MyJavaBox box1 = new MyJavaBox("red");
MyJavaBox box2 = new MyJavaBox("purple");
```

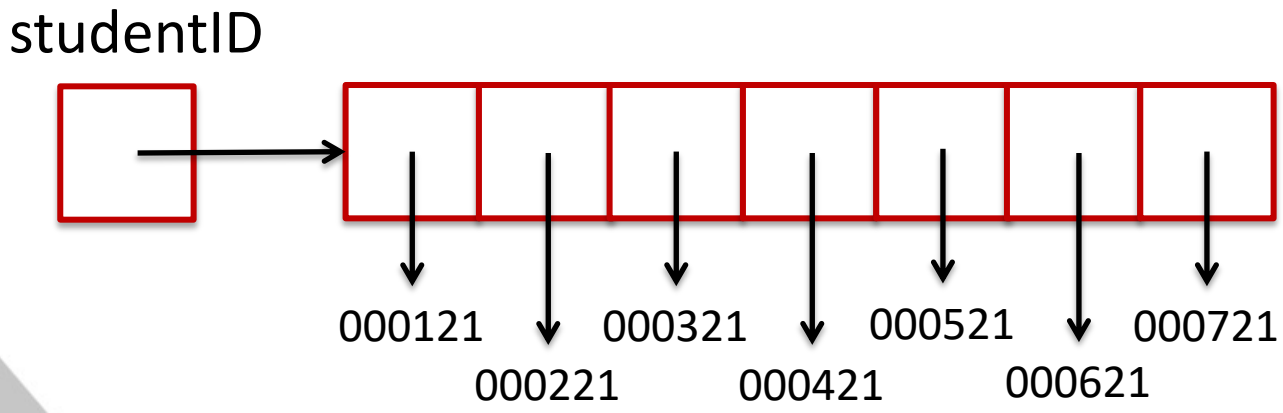
Define Your Own Data Type

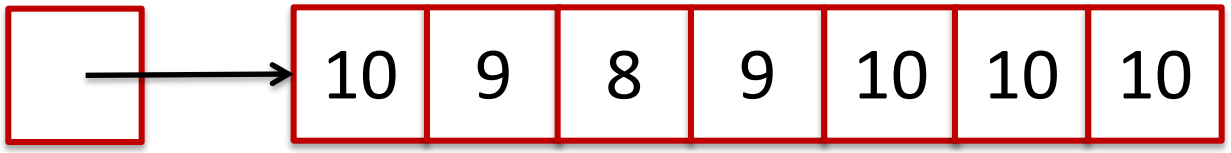
Suppose we want a data type that is used to store students' exam scores

Students' Exam Scores

ID	Score
000121	10
000221	9
000321	8
000421	9
000521	10
000621	10
000721	10

Possible Options: Two Arrays





Alternative: Array of *StudentScore* Objects

scoreData

id = “000121”
score = 10

id = “000221”
score = 9

id = “000321”
score = 8

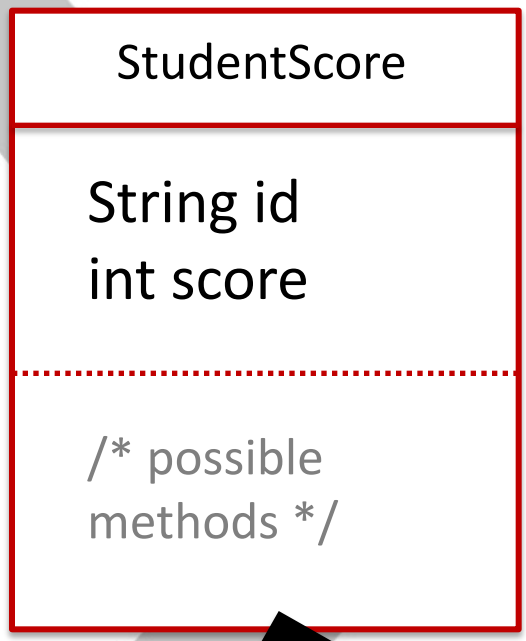
id = “000421”
re = 9

0521”
= 10

0621”
= 10

id = “000721”
10

Your Own Data Type: *StudentScore*




```
public class StudentScore{
   public String id;
   public int score;
}
```

StudentScore.java

UML Class Diagram

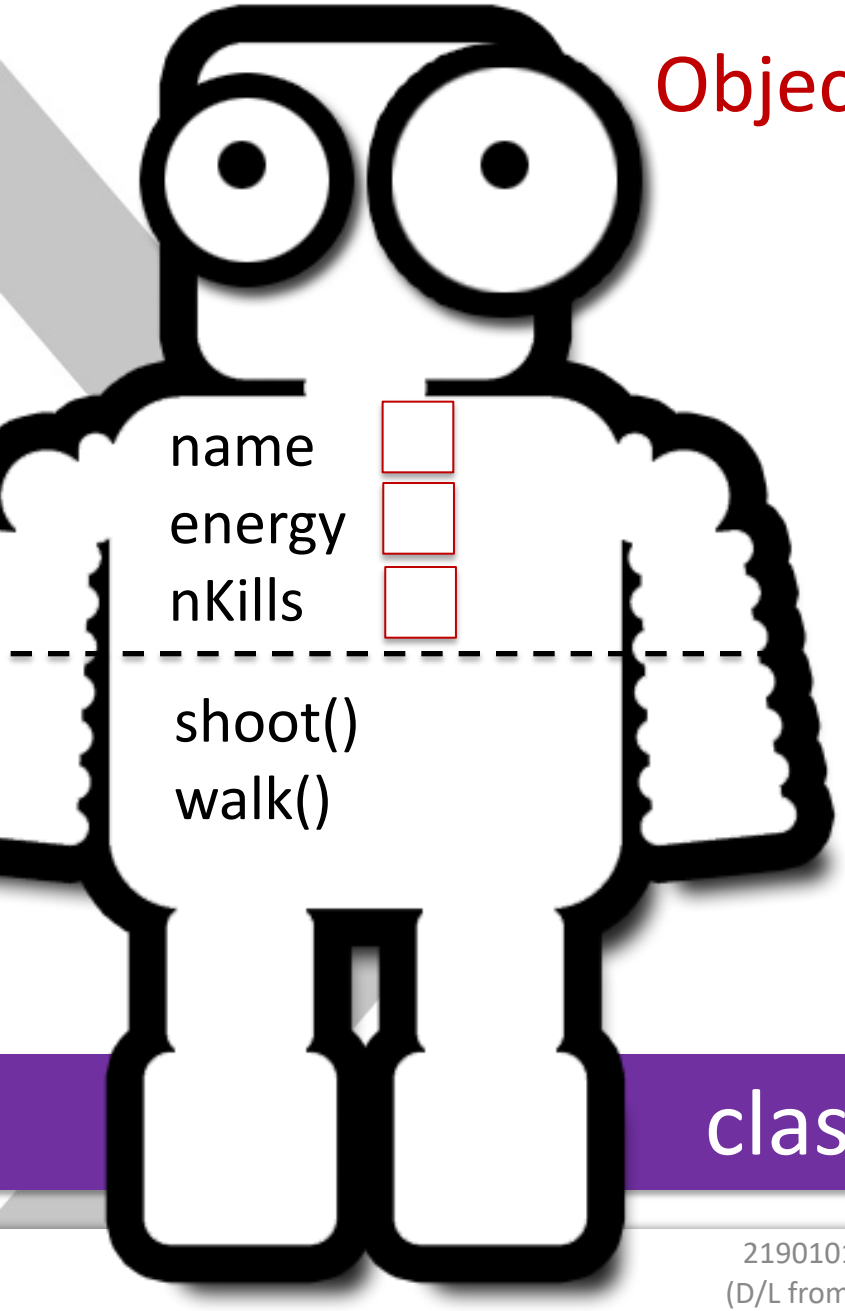
Class Definition

Using *StudentScore*


```
public class StudentScoreDemo1{
   public static void main(String [] args){
      StudentScore data1 = new StudentScore();
      data1.id = "000121";
      data1.score = 10;
      StudentScore data2 = new StudentScore();
      data2.id = "000221";
      data2.score = 9;
      System.out.println(
            data1.id+" got "+data1.score+" points.");
      System.out.println(
            data2.id+" got "+data2.score+" points.");
   }
}
```

StudentScoreDemo1.java

Object Data and Methods



data

fields
attributes
instance variables

methods

class MyRobo

Class Definition


```
public class StudentScore{
   public String id;
   public int score;
}
```

Access-level modifier

fields
attributes
instance variables
data members

Private Attributes


```
public class StudentScorePrivate{
   private String id;
   private int score;
}
```

Cannot be accessed using “dot operators” outside of the class definition

Accessing Private Attributes


```
public class StudentScoreDemo2{
    public static void main(String [] args){
        StudentScorePrivate data1 = new StudentScorePrivate();
        data1.id = "000121";
        data1.score = 10;
        StudentScorePrivate data2 = new StudentScorePrivate();
        data2.id = "000221";
        data2.score = 9;
        System.out.println(data1.id+" got "+data1.score+" points."
        System.out.println(data2.id+" got "+data2.score+" points."
    }
}
```

StudentScoreDemo2.java

Accessing Private Attributes


```
>javac StudentScoreDemo2.java
StudentScoreDemo2.java:4: error: id has private access in StudentScorePrivate
                data1.id = "000121";
                     ^
StudentScoreDemo2.java:5: error: score has private access in StudentScorePrivate

                data1.score = 10;
                     ^
StudentScoreDemo2.java:7: error: id has private access in StudentScorePrivate
                data2.id = "000221";
                     ^
StudentScoreDemo2.java:8: error: score has private access in StudentScorePrivate

                data2.score = 9;
                     ^
```

StudentScoreDemo2.java

Accessors/Mutators

getter

Accessors

Methods provided for other classes to read the values of (private) data members.

setter

Mutators

Methods provided for other classes to change the values of (private) data members.

Data Hiding

Accessors

Mutators

```
public class StudentScoreHiding{
    private String id;
    private int score;
    public String getId(){
        return id;
    }
    public int getScore(){
        return score;
    }
    public void setId(String id){
        this.id = id;
    }
    public void setScore(int score){
        this.score = score;
    }
}
```

Constructors

Constructors

- Special methods invoked whenever an object of the class is created
- Defined in the same fashion as defining methods.
- Must have the same name as the class name with the return type ommitted.

```
public class StudentScoreWithConstructor{
    private String id;
    private int score;
    public StudentScoreWithConstructor(){
        setId("000000");
        setScore(0);
    }
    public StudentScoreWithConstructor(String id,int score){
        setId(id);
        setScore(score);
    }
    public StudentScoreWithConstructor(
                                StudentScoreWithConstructor s){
        setId(s.getId());
        setScore(s.getScore());
    }
    /*Accessors/Mutators are omitted from being shown here*/
}
```

No-argument Constructor

Detailed Constructor

Copy Constructor

StudentScoreWithConstructor.java

Using Constructors


```
public class StudentScoreDemo4{
    public static void main(String [] args){
        StudentScoreWithConstructor data1 =
                        new StudentScoreWithConstructor();
        StudentScoreWithConstructor data2 =
                        new StudentScoreWithConstructor("000121",10);
        StudentScoreWithConstructor data3 =
                        new StudentScoreWithConstructor(data2);
        data3.setId("000821");
        System.out.println(
            data1.getId()+" got "+data1.getScore()+" points.");
        System.out.println(
            data2.getId()+" got "+data2.getScore()+" points.");
        System.out.println(
            data3.getId()+" got "+data3.getScore()+" points.");
    }
}
```

StudentScoreDemo4.java

The *toString()* method

In order to provide a meaningful *String* representation of the class we create, it is sensible to provide the method named exactly as *toString()* that returns the *String* representation we want.

The *toString()* method


```
public String toString(){
      return "(ID:"
            + getId()
            + ", score:"
            + getScore()
            + ")";
}
```

StudentScoreDemo5.java

Additional Behaviors


```
public class StudentScore2{
    /* ------------------------------ */
    /* Attributes, Constructors, toString() */
    /* Accessors/Mutators are omitted from showing */
    /* ------------------------------ */
        :
    public void receiveBonus(int bonus){
        setScore(getScore()+bonus);
    }
    public void penalizeByPercent(double percent){
        double newScore = (1.0-percent/100)*getScore();
        setScore((int)Math.round(newScore));
    }
    public boolean hasHigherScoreThan(StudentScore2 s){
        return getScore()>s.getScore();
    }
}
```

StudentScore2.java

Additional Behaviors


```
public class StudentScoreDemo6{
    public static void main(String [] args){
        StudentScore2 data1 = new StudentScore2("000121",10);
        StudentScore2 data2 = new StudentScore2("000221",9);
        data1.penalizeByPercent(50);
        data2.receiveBonus(2);
        System.out.println(data1+" Vs."+data2);
        if(data1.hasHigherScoreThan(data2)){
            System.out.println(
                data1.getId()+" gets more than "+data2.getId()
            );
        }else{
            System.out.println(
                data1.getId()+" does not get more than "+data2.getId()
            );
        }
    }
}
```

StudentScoreDemo6.java

Example: Complex Numbers

$$(a + jb)$$

- A complex number is of the form $a+jb$

  where a and b are real numbers

  j is a quantity representing $\sqrt{-1}$.

- We would like to define a new class for complex numbers.

Add $(a + jb) + (c + jd) = (a + c) + j(b + d)$

Subtract $(a + jb) - (c + jd) = (a - c) + j(b - d)$

Multiply $(a + jb)(c + jd) = (ac - bd) + j(bc + ad)$

Divide $$\frac{(a + jb)}{(c + jd)} = (a + jb)(c + jd)^{-1}$$

Inverse $$(a + jb)^{-1} = \left(\frac{a}{a^2 + b^2}\right) + j\left(\frac{-b}{a^2 + b^2}\right)$$

Magnitude $$|(a + jb)| = \sqrt{(a^2 + b^2)}$$

Example: Complex Numbers

Complex.java

Complex
private double re private double im
public Complex adds(Complex z) public Complex subtracts(Complex z) public Complex multiplies(Complex z) public Complex divides(Complex z) public Complex multInverse() public Complex conjugate() public double magnitude() public String toString() /* constructors, accessors,mutators are omitted from showing */

To do:

- Read about creating Static attributes/methods
- Compare the concepts in this slides with the *MyPoint* class in the textbook.
- Find out about what will happen if a class has constructors but does not have the “no-argument” constructor.