Java Primer I

CMSC 202

Variable Declaration

- Syntax: `<type> <legal identifier>;`
- Examples:

```
int sum;
float average;
double grade = 98;
```

Semicolon required!

  - Must be declared before being used
  - Must appear within a class declaration (no “globals”)
  - Must be declared of a given type (e.g. int, float, char, etc.)

Java's Legal Identifiers

- Are case-sensitive
  - Cat, CAT, CaT are all different variable names
- Typically consist of letters, numbers and underscores
- Must not begin with a number
- Must not contain whitespace
- Must not be a reserved/key word

Naming Conventions

- Naming Conventions
  - Additional rules that restrict the names of variables resulting in improving consistency/readability
  - Most places of work and education have a set of naming conventions
  - These are not language or compiler enforced

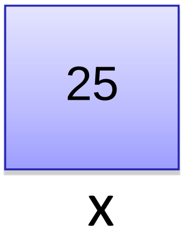
- CMSC 202 Naming Conventions
  - Variables & methods
    - Start with a lowercase letter
    - Indicate “word” boundaries with an uppercase letter
    - Restrict the remaining characters to digits and lowercase letters
  - Classes
    - Start with an uppercase letter
    - Otherwise same as variables and methods
  - See the CMSC 202 course website

Variable Types

Primitive Type

- Declared to be of basic type
  - e.g. float, double, char, int
- Variables hold actual data

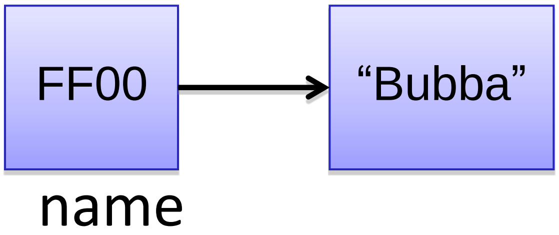
```
int x = 25;
```



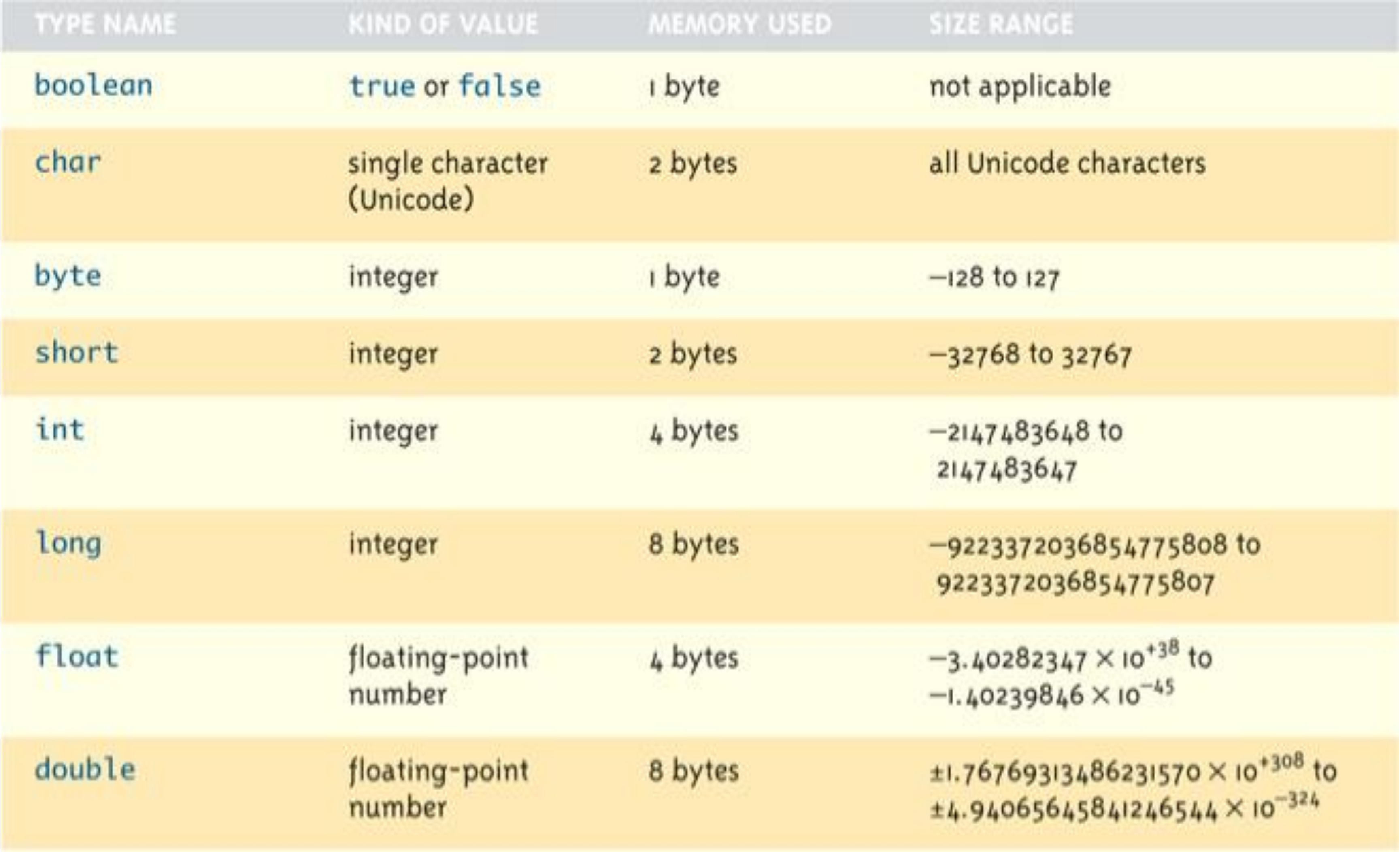
Reference Type

- Declared to be of class type
  - e.g. String, MyClass, Integer
- Variables hold addresses to dynamically allocated memory space
  - We will discuss this in more detail later

```
String name = "Bubba";
```



Primitive Types



TYPE NAME	KIND OF VALUE	MEMORY USED	SIZE RANGE
boolean	true or false	1 byte	not applicable
char	single character (Unicode)	2 bytes	all Unicode characters
byte	integer	1 byte	−128 to 127
short	integer	2 bytes	−32768 to 32767
int	integer	4 bytes	−2147483648 to 2147483647
long	integer	8 bytes	−9223372036854775808 to 9223372036854775807
float	floating-point number	4 bytes	$-3.40282347 \times 10^{+38}$ to $-1.40239846 \times 10^{-45}$
double	floating-point number	8 bytes	$\pm 1.76769313486231570 \times 10^{+308}$ to $\pm 4.94065645841246544 \times 10^{-324}$

Primitive Types

- All primitive type variables store the information inside of the variable

```
int x = 25;
```

  - x contains the value 25
  - There are no additional steps required to access the contents of x

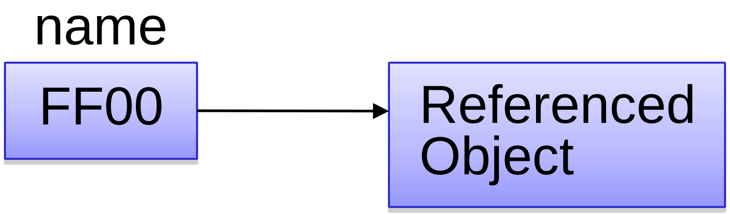
- Default Values
  - Java automatically initializes all declared primitive variables to a default value that is equivalent to 0.
    - Integer and floating point types are set to 0.
    - The character type is set to the ‘\u0000’ Unicode character (null).
    - The boolean type is set to false.

Reference Types

- Reference type variables must be created dynamically and are generally in the form

```
ReferencedType name = new ReferencedType();
```

- The "new" keyword creates an instance of a class.
- It returns an address to the newly created object on the heap.
- Typically the address is assigned into a variable (e.g. "name").
- The instance can then be referenced using the variable name.
- Members and methods can be accessed using dot notation.



Arrays

- Arrays are referenced objects that hold a fixed number of **homogeneous** data (i.e. data of the same type).
- These elements appear in **contiguous** memory.
- General form:

```
<type>[] <variable name>;
```

- Sample declarations:

```
int[] scores;

float[] grades;
```

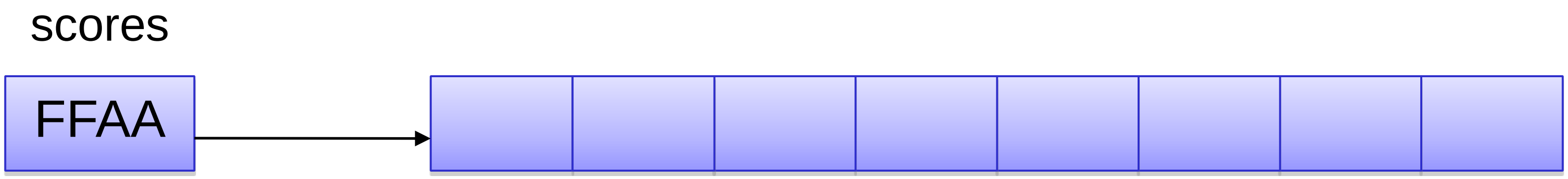
- What does each variable contain at this point?

scores
grades

Arrays

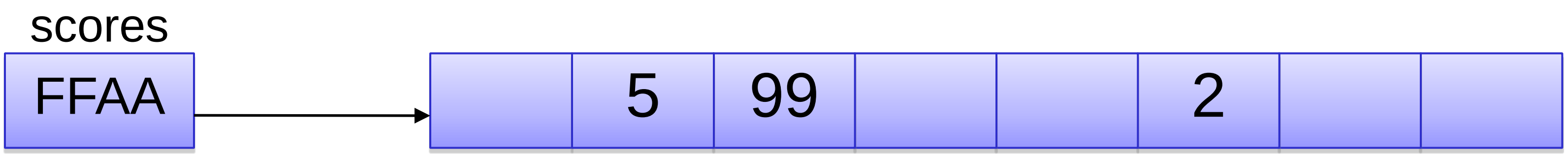
- Initializing an array requires the usage of the keyword “new” to create the space on the heap to hold the elements

```
type[] variable_name = new type[number_of_elements];
int[] scores = new int[8];
```



  - Java initializes all elements of the array to the default value for that type
  - The size of an array can be obtained by accessing the **length** member variable (e.g. scores.length).
  - An array of size 8 will have what for indices?

Arrays



- We can access any element in the array using array_name[index]
  - scores[1] will return what value?
  - scores[0] = 82;
    - Assigns 82 to index 0 of the array

- How does accessing with array_name [index] really work?
  - FFAA is the address of the first element of the array.
  - Since all elements of an array are of a common type, we know that each element will consume the same amount of space.
  - Using that knowledge, we can compute the location (offset) of the element within the array.

    scores[2] → FFAA + size of (type)*index
  - Luckily, Java handles all this for you!

Multi-Dimensional Arrays

- Really should be considered an array of arrays (and potentially of arrays, and so forth)
- You can declare multi-dimensional arrays just like single dimensional arrays.
- The general form:

```
type[][] array_name = new type[ rows ][ columns ];
```

- Example:

```
char [][] ticTacToeBoard = new char[3][3];
```

- Use the same access syntax as single dimensional arrays.
- What statement will place an O in the upper right corner?

	X	

Printing to the Screen

- Formatted output

```
System.out.printf("Printing integer %d%n",5);
System.out.printf("%d %c %d", 1, 'a', 2);
```

- Place holders can be added to represent variables to be output in the format string.
  - %d, %c, %f, %s – What does each stand for?
  - Every place holder that appears inside the output string must have a matching value separated by a comma.
- Add proceeding white space characters and precision to variables printed.

```
System.out.printf("2 points of precision %10.2d", 89.999);
```

  - “Two points of precision ____90.00” ← no newline character
- Other special formatting
  - %n – platform independent newline character
  - \t – horizontal tab

Printing to the Screen (con’t)

- Unformatted output
  - General formats:
    - System.out.print(…)   leaves cursor on same line
    - System.out.println(…)   cursor moves to next line
  - Example:

```
System.out.print(“Hello”);
System.out.print(“ there”);
System.out.println(“Hello”);
System.out.println(“ there”);
```

Output:

```
Hello thereHello
 there
```

Binary Operators

- What is a binary operator?
  - An operator that has two operands

    <operand> <operator> <operand>

  - Arithmetic Operators

    +   -   *   /   %

  - Relational Operators

    <   >   ==   <=   >=

  - Logical Operators

    && ||

Relational Operators

- In Java, all relational operators evaluate to a boolean value of either **true** or **false** .

```
x = 5;
y = 6;
```

  - x > y will always evaluate to **false** .

- Java has a ternary operator – the general form is:

  (conditional expression) ? true case : false case ;

- For example:

```
System.out.println(( x > y ) ? "X is greater" : "Y is greater");
```

Unary Operators

- Unary operators only have one operand.

  !   ++   --

  ++ and -- are the **increment** and **decrement** operators
  x++   **a post-increment** (postfix) operation
  ++x   **a pre-increment** (prefix) operation

  - What is the difference between these segments?

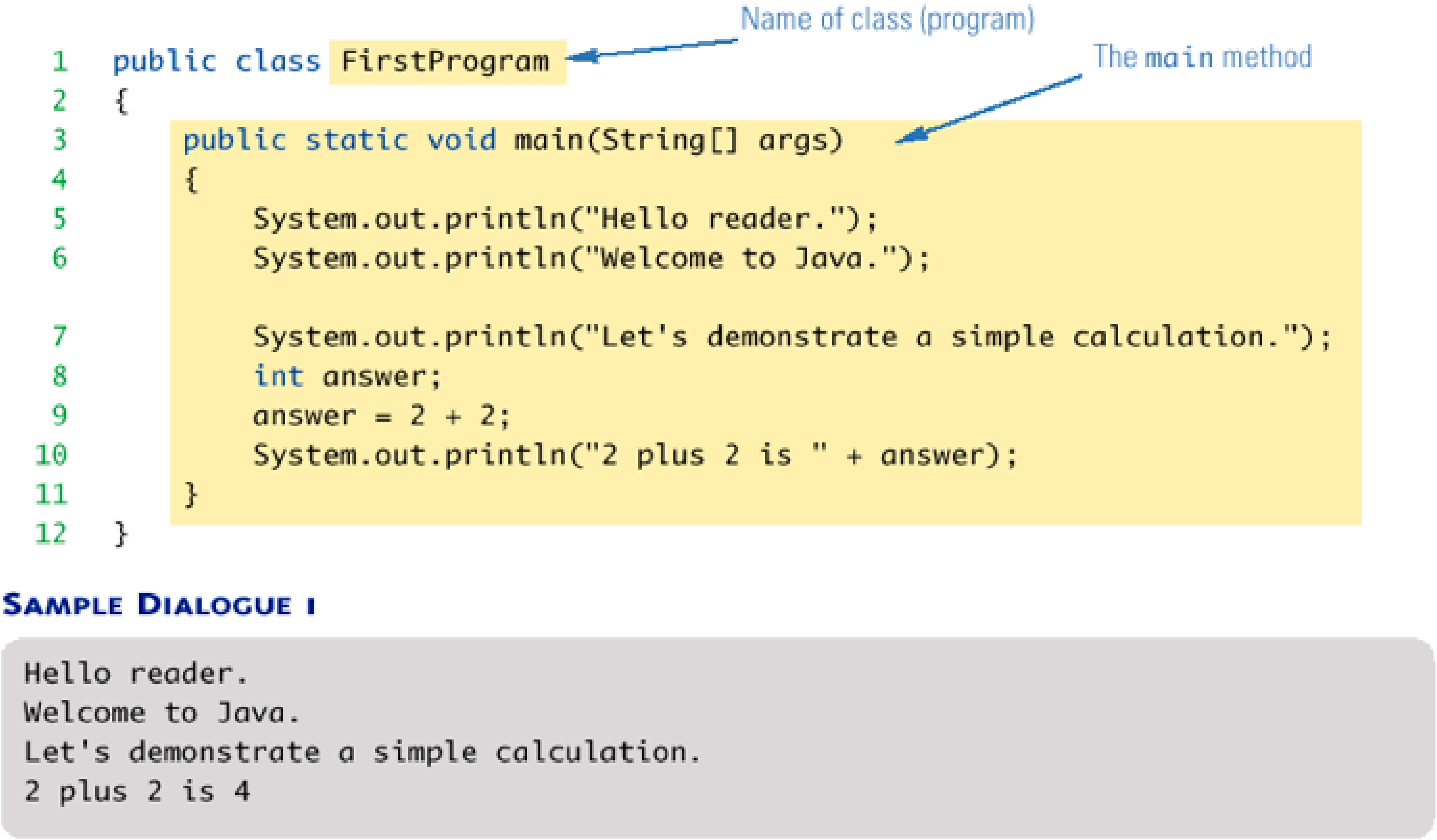
```
x = 5;
System.out.printf("x's value %d%n", x++);

x = 5;
System.out.printf("x's value %d%n", ++x);
```

Precedence

- Order of operator application to operands:
  - Postfix operators: ++ -- (right to left)
  - Unary operators: + - ++ -- ! (right to left)
  - * / % (left to right)
  - + - (left to right)
  - < > <= >=
  - == !=
  - &&
  - ||
  - ? :
  - Assignment operator: = (right to left)

A Sample Java Program



```java
public class FirstProgram
{
    public static void main(String[] args)
    {
        System.out.println("Hello reader.");
        System.out.println("Welcome to Java.");

        System.out.println("Let's demonstrate a simple calculation.");
        int answer;
        answer = 2 + 2;
        System.out.println("2 plus 2 is " + answer);
    }
}
```

Sample Dialogue 1

```
Hello reader.
Welcome to Java.
Let's demonstrate a simple calculation.
2 plus 2 is 4
```