Announcements

- IDE tutorial on the course website
- TA office hours on the website

Java Syntax

- Class definitions

```
public class {
    // methods
}
```

- Method definitions (inside a class)

```
// method header/signature
{
    // statements
}
```

Basic java programs

```
public class ClassName
{
    public static void main(String[] args)
    {
        // Statements
    }
}
```

Statements

- Print statement

```
System.out.println(string_expression);
```

- Variable declaration

```
type identifier;
```

- Assignment

```
variable = value;
```

- Statements in a method are executed in *sequential order* from top to bottom

Assignment

- In an assignment

  variable = *value*;

- the variable must have been declared before,

```
x = 7; // incorrect
int x;
```

- the type of the variable must match the type of the value

```
int x;
x = "7"; // incorrect
```

Sequential execution

```
public class OrderTest
{
    public static void main(String[] args)
    {
        int a;
        int b;
        a = 2;
        b = 3;
        b = 5;
        a = 8;
        System.out.println(a);
        System.out.println(b);
    }
}
```

Sequential execution

```
public class OrderTest
{
    public static void main(String[] args)
    {
        int a;
        int b;
        b = 5;
        a = 8;
        a = 2;
        b = 3;
        System.out.println(a);
        System.out.println(b);
    }
}
```

Sequential execution

```
public class OrderTest
{
    public static void main(String[] args)
    {
        int a;
        int b;
        a = 2;
        b = 5;
        a = 8;
        b = 3;
        System.out.println(a);
        System.out.println(b);
    }
}
```

Some syntactic shortcuts

- Several variables of the same type can be declared in the same variable declaration:

  type var1, var2, ..., varn ;

- Examples:

```
int a;
int b;
```

is equivalent to

```
int a, b;
```

Some syntactic shortcuts

- A variable can be initialized when declared

```
int a;
a = 2;
```

is equivalent to

```
int a = 2;
```

- But a variable cannot be redeclared, so

```
int b = 3;
int b = 2;
```

is incorrect, while the following is correct

```
int b = 3;
b = 2;
```

User Interface

- Interaction between the user and some program
- Textual UI
  - Output:

    ```
    System.out.println(string_expression);
    ```

  - Input:

    ```
    Keyboard.readInt();
    Keyboard.readString();
    ```

- Examples:

```
int n;
n = Keyboard.readInt();
```

User Interface

```
import cs1.Keyboard;
public class UserInputTest {
    public static void main(String[] args)
    {
        String name;
        int age;
        System.out.print(“Enter your name: ”);
        name = Keyboard.readString();
        System.out.print(“Enter your age: ”);
        age = Keyboard.readInt()
        System.out.println(“Your name is ” + name);
        System.out.println(“You are ” + age + “ years old”);
    }
}
```

User Interface

```
import cs1.Keyboard;
public class UserInputTest {
    public static void main(String[] args)
    {
        String name;
        int age;
        System.out.print(“Enter your name: ”);
        name = Keyboard.readString();
        System.out.print(“Enter your age: ”);
        age = Keyboard.readInt()
        System.out.println(“Your name is ” + name);
        System.out.println(“You are ” + age + “ years old”);
    }
}
```

User Interface

```
import cs1.Keyboard;
public class UserInputTest {
    public static void main(String[] args)
    {
        String name;
        int age;
        System.out.print(“Enter your name: ”);
        name = Keyboard.readString();
        System.out.print(“Enter your age: ”);
        age = Keyboard.readInt()
        System.out.println(“Your name is ” + name);
        System.out.println(“You are ” + age + “ years old”);
    }
}
```

User Interface

```
import cs1.Keyboard;
public class UserInputTest {
    public static void main(String[] args)
    {
        String name;
        int age;
        System.out.print(“Enter your name: ”);
        name = Keyboard.readString();
        System.out.print(“Enter your age: ”);
        age = Keyboard.readInt()
        System.out.println(“Your name is ” + name);
        System.out.println(“You are ” + age + “ years old”);
    }
}
```

User Interface

```
import cs1.Keyboard;
public class UserInputTest {
    public static void main(String[] args)
    {
        String name;
        int age;
        System.out.print(“Enter your name: ”);
        name = Keyboard.readString();
        System.out.print(“Enter your age: ”);
        age = Keyboard.readInt()
        System.out.println(“Your name is ” + name);
        System.out.println(“You are ” + age + “ years old”);
    }
}
```

User Interface

```
import cs1.Keyboard;
public class UserInputTest {
    public static void main(String[] args)
    {
        String name;
        int age;
        System.out.print(“Enter your name: ”);
        name = Keyboard.readString();
        System.out.print(“Enter your age: ”);
        age = Keyboard.readInt()
        System.out.println(“Your name is ” + name);
        System.out.println(“You are ” + age + “ years old”);
    }
}
```

Data types

- Each variable has a *data type*

```
String major;
int age;
```

- A data type is a set of possible values
  - `int` is the set of integers
  - `String` is the set of strings
  - `float` is the set of rational numbers written as a decimal expansion
  - `double` is the set of rational numbers as a decimal expansion, with double precision
  - `char` is the set of characters
  - `boolean` is the set {`true`, `false`}
  - `byte` is the set of bytes, writen in decimal

Data types

Data type	Possible values	Examples
int	all integers between -2^{31} and $2^{31}-1$	0, 1, 2, -3, -1729
String	all character strings enclosed in " "	"hello bye", "", "a"
float	rationals between -3.4×10^{38} and 3.4×10^{38}	0.0f, -2.3f, 111.001f
double	rationals between -1.7×10^{308} and 1.7×10^{308}	0.0, -2.3, 111.001
char	all individual characters enclosed in ' '	'a', 'b','z','7','+','A'
boolean	only true and false	true, false
byte	all integers between -128 and 127	-128, 0, 8
long	integers between -2^{63} and $2^{63}-1$	0l, 65536l, -3l
short	integers between -2^{15} and $2^{15}-1$	-3, -2, 0, 1, 4

Data types

Data type	Size	Range
boolean	8 bits (7 unused)	0 - 1
byte	8 bits	-2^7 to $2^7 - 1$
char	8 bits (ASCII), 16 bits (Unicode)	0 to 2^8 (ASCII) 0 to 2^{16} (Unicode)
short	16 bits	-2^{15} to $2^{15} - 1$
int	32 bits	-2^{31} to $2^{31} - 1$
long	64 bits	-2^{63} to $2^{63} - 1$

Real numbers

$\sqrt{2}$

$\sqrt{3}$

π

$e = 2.718...$

$\varphi = \frac{1 \pm \sqrt{5}}{2} = \begin{cases} 1.618... \\ 0.618... \end{cases}$

Assignment and data types

```
int x = 3.141592;          // Line 1
float pi = 3.141592f;      // Line 2
double e = 2.718;          // Line 3
float phi = 1.618;         // Line 4
int n = 32768;             // Line 5
int m = 32767;             // Line 6
long o = 32768l;           // Line 7
String letter = ‘A’;       // Line 8
char letter2 = ‘A’;        // Line 9
char letter3 = “B”;        // Line 10
```

Arithmetic expressions

- If a variable is of a numeric type (int, float, long, etc.) then an assignment can take the form

  variable = *arithmetic_expression*;

- where arithmetic_expression is an expression involving:
  - numbers (of the appropriate type)
  - operators (+, -, *, /, %)
  - variables (of numeric type)
  - parenthesis

- Example:

```
double grade, assignments, midterm, final;
assignments = 97.5;
midterm = 75.5;
final = 80.0;
grade = assignments * 0.25
      + midterm * 0.20
      + final * 0.55;
```

Arithmetic expressions

- Parenthesis are used to group operations:

```
double grade, assignments, midterm, final;
double a1 = 20, a2 = 19, a3 = 9, a4 = 14, a5 = 18;
assignments = a1 + a2 + a3 + a4 + a5;
midterm = 75.5;
final = 80.0;
grade = assignments * 0.25
      + midterm * 0.20
      + final * 0.55;
```

is equivalent to

```
double grade, midterm, final;
double a1 = 20, a2 = 19, a3 = 9, a4 = 14, a5 = 18;
midterm = 75.5;
final = 80.0;
grade = (a1 + a2 + a3 + a4 + a5) * 0.25
      + midterm * 0.20
      + final * 0.55;
```

Operator precedence

- Operators are evaluated depending on their precedence:

```
result = 6 + 5 * 3;
```

- If the operators did not have precedence, the expression would have as value 33
- But it's real meaning is:

```
result = 6 + (5 * 3);
```

- Which evaluates to 6 + 15 which is 21.
- Operators have "associativity":

```
result = 6 + 5 + 3 + 9;
```

- is evaluated as

```
result = ((6 + 5) + 3) + 9;
```

Operator precedence

Precedence level	Operator	Operation	Associativity
1	+ -	unary plus unary minus	right to left
2	* / %	multiplication division remainder (modulo)	left to right
3	+ - +	addition substraction string concatenation	left to right

Operator precedence

```
r = 8 / 2 / 2;
```

is evaluated as

```
r = ((8 / 2) / 2);
```

and

```
s = 12 * 2 - - 3;
```

is evaluated as

```
s = (12 * 2) - (-3);
```

and

```
t = -2 * 4 + - (a - 1);
```

is evaluated as

```
t = ((-2) * 4) + (- (a - 1));
```

Assignment

- The semantics of an assignment statement

  variable = *expression*;

- is to evaluate the expression, and the resulting value replaces the contents of the memory location of the variable.

- Evaluating an expression may involve obtaining the value of some variable:

```
int x = 4;
int y = 7;
x = y + 2;
```

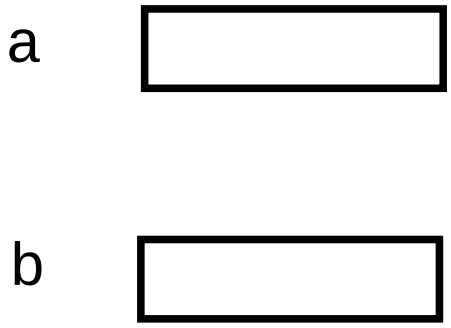
Sequential execution

```
double a, b;
a = 2.0;
b = a;
a = 3.0;
System.out.println(a);
System.out.println(b);
```

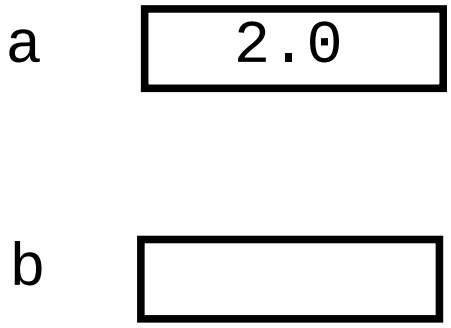
Sequential execution

```
double a, b;
a = 2.0;
b = a;
a = 3.0;
System.out.println(a);
System.out.println(b);
// Prints
// 3.0
// 2.0
```

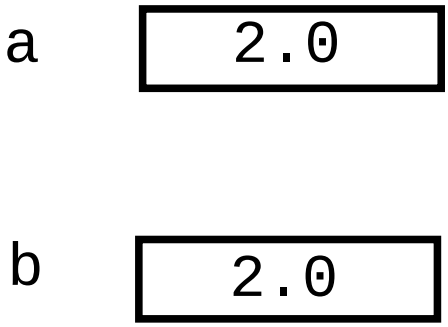
Sequential execution



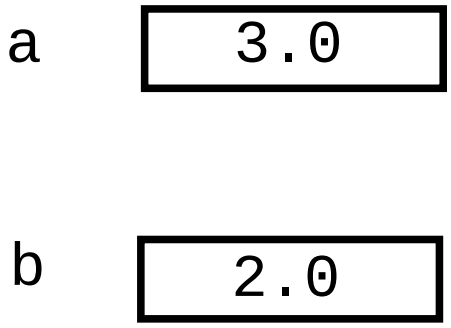
Sequential execution



Sequential execution



Sequential execution



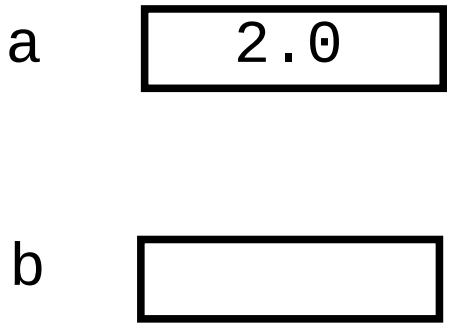
Sequential execution

```
double a, b;
a = 2.0;
a = 3.0;
b = a;
System.out.println(a);
System.out.println(b);
```

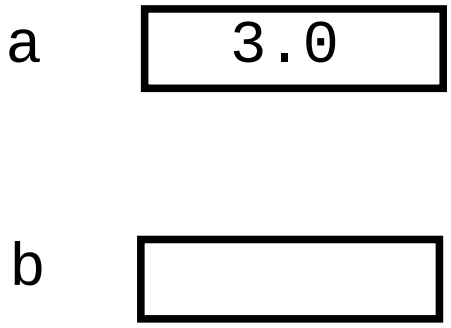
Sequential execution

```
double a, b;
a = 2.0;
a = 3.0;
b = a;
System.out.println(a);
System.out.println(b);
// Prints
// 3.0
// 3.0
//
// a and b have the same contents (the same value)
// but they are different variables
```

Sequential execution



Sequential execution



Sequential execution

Sequential execution

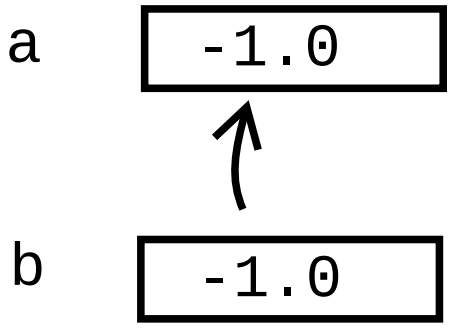
```
double a, b;
a = 2.0;
b = -1.0;
a = b;
b = a;
System.out.println(a);
System.out.println(b);
```

Sequential execution

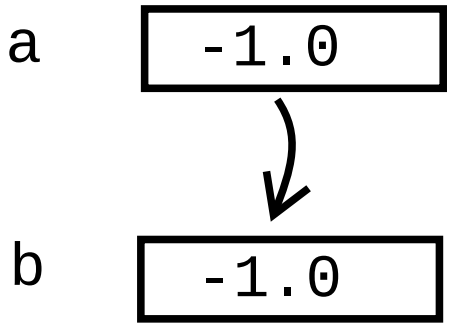
```
double a, b;
a = 2.0;
b = -1.0;
a = b;
b = a;
System.out.println(a);
System.out.println(b);
// Prints
// -1.0
// -1.0
```

Sequential execution

Sequential execution



Sequential execution



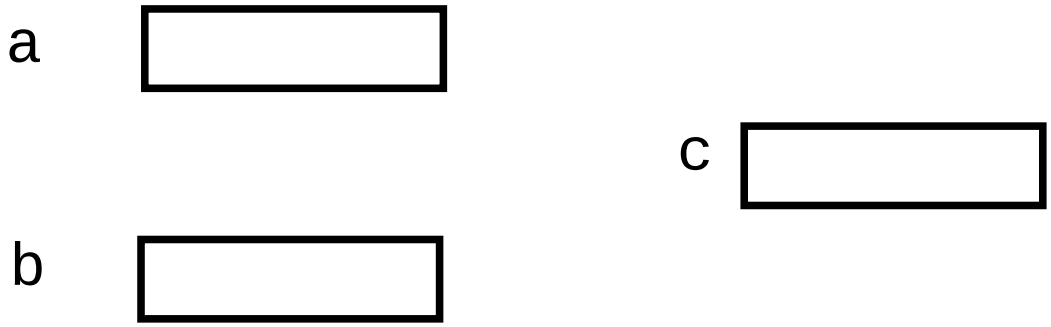
Sequential execution

```
double a, b, c;
a = 2.0;
b = -1.0;
c = a;
a = b;
b = c;
System.out.println(a);
System.out.println(b);
```

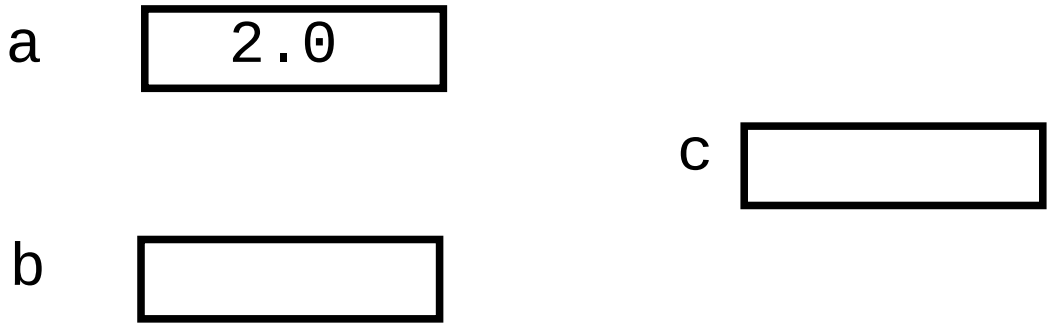
Sequential execution

```
double a, b, c;
a = 2.0;
b = -1.0;
c = a;
a = b;
b = c;
System.out.println(a);
System.out.println(b);
// Prints
// -1.0
// 2.0
// This implements a 'swap' between variables
```

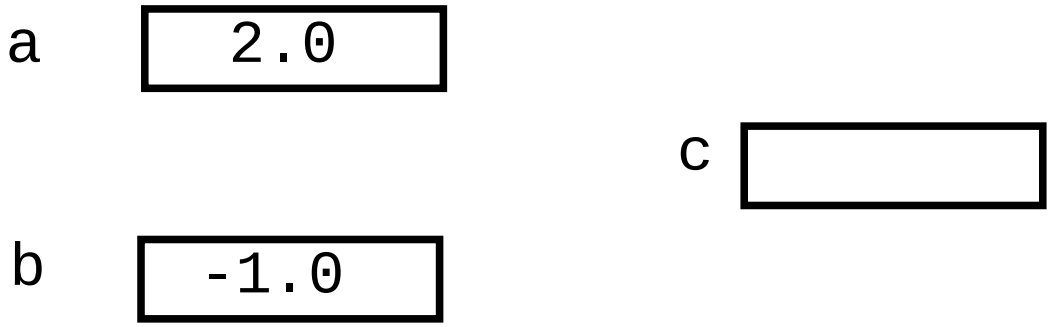
Sequential execution



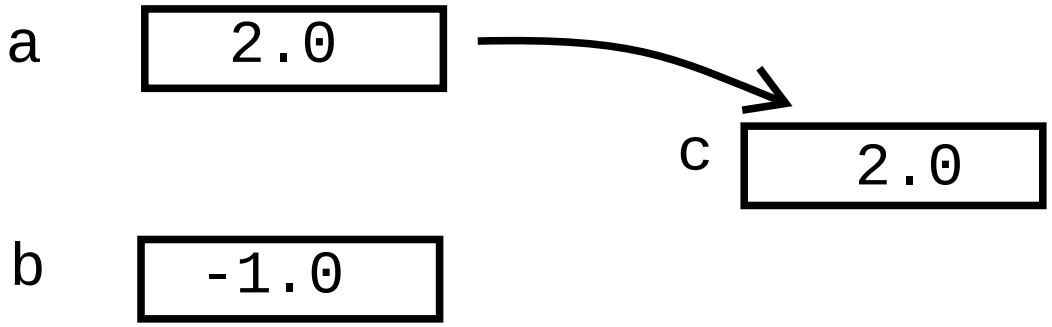
Sequential execution



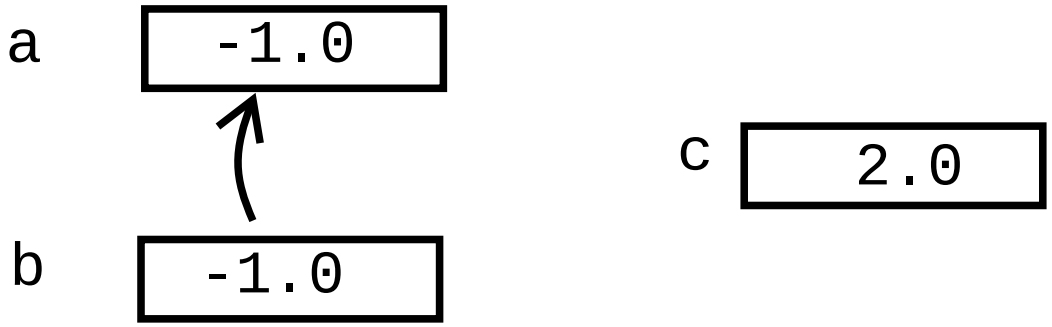
Sequential execution



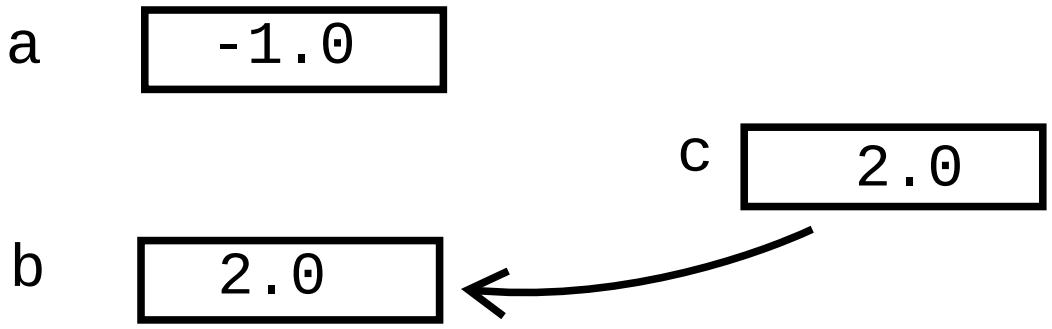
Sequential execution



Sequential execution



Sequential execution



ORDER MATTERS

More on arithmetic operations

- The remainder operator % computes the remainder of an integer division (not percentages!)
  - 8 % 2 evaluates to 0
  - 7 % 2 evaluates to 1
  - 8 % 3 evaluates to 2
  - 3 % 5 evaluates to 5
- In general, for every integers a and b, $0 \leqslant a\%b < b$
- Division has a different meaning for integers and for floats and doubles
- The division between two integers is an integer
- The division between a float or double and an integer is a float or double
  - 8 / 3 evaluates to 2
  - 8.0 / 3 evaluates to 2.666666...