

6.092: Intro to Java

**2: More types, Methods,
Conditionals**

Outline

- Lecture 1 Review
- Methods
- Conditionals

Types

Kinds of values that can be stored and manipulated.

boolean: Truth value (**true** or **false**).

int: Integer (0, 1, -47).

double: Real number (3.14, 1.0, -2.1).

String: Text (“hello”, “example”).

Variables

Named location that stores a value

Example:

```
String a = "a";  
String b = "letter b";  
a = "letter a";  
String c = a + " and " + b;
```

Operators

Symbols that perform simple computations

Assignment: =

Addition: +

Subtraction: -

Multiplication: *

Division: /

Conversion by casting

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

```
double a = 2;   // a = 2.0 (Implicit)
```

```
int a = 18.7;   // ERROR
```

```
int a = (int)18.7; // a = 18
```

```
double a = 2/3; // a = 0.0
```

```
double a = (double)2/3; // a = 0.6666...
```

Order of Operations

Precedence like math, left to right

Right hand side of = evaluated first

Parenthesis increase precedence

```
double x = 3 / 2 + 1; // x = 2.0
```

```
double y = 3 / (2 + 1); // y = 1.0
```

```
class GravityCalculator {  
    public static void main(String[] args) {  
        double gravity = -9.81;  
        double initialVelocity = 0.0;  
        double fallingTime = 10.0;  
        double initialPosition = 0.0;  
        double finalPosition = .5 * gravity * fallingTime *  
            fallingTime;  
        finalPosition = finalPosition +  
            initialVelocity * fallingTime;  
        finalPosition = finalPosition + initialPosition;  
        System.out.println("An object's position after " +  
            fallingTime + " seconds is " +  
            finalPosition + " m.");  
    }  
}
```


Questions from last lecture?

Outline

- Lecture 1 Review
- **Methods**
- Conditionals

Methods

```
public static void main(String[] arguments)
```

```
{
```

```
    System.out.println("hi");
```

```
}
```

Adding Methods

```
public static void NAME() {  
    STATEMENTS  
}
```

To call a method:

```
NAME ( ) ;
```

```
class Methods {
```

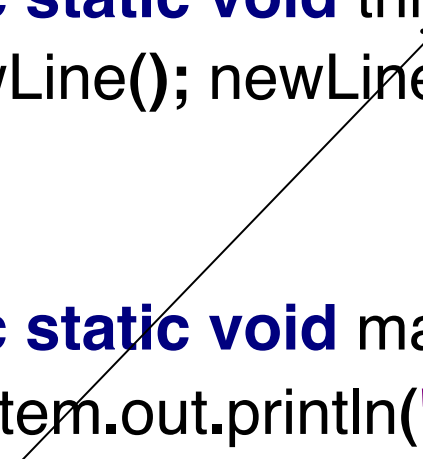
```
    public static void newLine() {  
        System.out.println("");  
    }
```

```
    public static void threeLines() {  
        newLine(); newLine(); newLine();  
    }
```

```
    public static void main(String[] arguments) {  
        System.out.println("Line 1");  
        threeLines();  
        System.out.println("Line 2");  
    }
```

```
}
```

```
class Methods {  
    public static void newLine() {  
        System.out.println("");  
    }  
  
    public static void threeLines() {  
        newLine(); newLine(); newLine();  
    }  
  
    public static void main(String[] arguments) {  
        System.out.println("Line 1");  
        threeLines();  
        System.out.println("Line 2");  
    }  
}
```



```
class Methods {  
    public static void newLine() {  
        System.out.println("");  
    }  
  
    public static void threeLines() {  
        newLine(); newLine(); newLine();  
    }  
  
    public static void main(String[] arguments) {  
        System.out.println("Line 1");  
        threeLines();  
        System.out.println("Line 2");  
    }  
}
```

The diagram illustrates the flow of method calls in the provided Java code. Arrows indicate the following sequence: 1. An arrow from the `main` method's body to the `threeLines` method. 2. An arrow from the first `newLine()` call inside `threeLines` to the `newLine` method. 3. An arrow from the second `newLine()` call inside `threeLines` to the `newLine` method. 4. An arrow from the third `newLine()` call inside `threeLines` to the `newLine` method.

Parameters

```
public static void NAME(TYPE NAME) {  
    STATEMENTS  
}
```

To call:

```
NAME ( EXPRESSION ) ;
```



```
class Square {  
    public static void printSquare(int x) {  
        System.out.println(x*x);  
    }  
  
    public static void main(String[] arguments) {  
        int value = 2;  
        printSquare(value);  
        printSquare(3);  
        printSquare(value*2);  
    }  
}
```

```
class Square2 {  
    public static void printSquare(int x) {  
        System.out.println(x*x);  
    }  
  
    public static void main(String[] arguments) {  
        printSquare("hello");  
        printSquare(5.5);  
    }  
}
```

What's wrong here?

```
class Square3 {  
    public static void printSquare(double x) {  
        System.out.println(x*x);  
    }  
  
    public static void main(String[] arguments) {  
        printSquare(5);  
    }  
}
```

What's wrong here?

Multiple Parameters

```
[...] NAME(TYPE NAME, TYPE NAME) {  
    STATEMENTS  
}
```

To call:

```
NAME (arg1, arg2) ;
```

```
class Multiply {  
    public static void times (double a, double b) {  
        System.out.println(a * b);  
    }  
  
    public static void main(String[] arguments) {  
        times (2, 2);  
        times (3, 4);  
    }  
}
```

Return Values

```
public static TYPE NAME() {  
    STATEMENTS  
    return EXPRESSION;  
}
```

`void` means “no type”

```
class Square3 {  
    public static void printSquare(double x) {  
        System.out.println(x*x);  
    }  
  
    public static void main(String[] arguments) {  
        printSquare(5);  
    }  
}
```

```
class Square4 {  
    public static double square(double x) {  
        return x*x;  
    }  
  
    public static void main(String[] arguments) {  
        System.out.println(square(5));  
        System.out.println(square(2) + 1);  
    }  
}
```



```
class Square4 {  
    public static double square(double x) {  
        return x*x;  
    }  
  
    public static void main(String[] arguments) {  
        System.out.println(square(5)); // println(25)  
        System.out.println(square(2) + 1);  
    }  
}
```

```
class Square4 {  
    public static double square(double x) {  
        return x*x;  
    }  
  
    public static void main(String[] arguments) {  
        System.out.println(square(5)); // println(25)  
        System.out.println(square(2) + 1); // println(4+1)  
    }  
}
```

Methods: Building Blocks

- Big programs are built out of small methods
- Methods can be individually developed, tested and reused
- User of method does not need to know how it works
- In Computer Science, this is called “*abstraction*”

Mathematical Functions

`Math.sin(x)`

`Math.cos(Math.PI / 2)`

`Math.pow(2, 3)`

`Math.log(Math.log(x + y))`

Outline

- Lecture 1 Review
- Methods
- **Conditionals**

if statement

```
if (CONDITION) {  
    STATEMENTS  
}
```

```
public static void test(int x) {  
    if (x > 5) {  
        System.out.println(x + " is > 5");  
    }  
}
```

```
public static void main(String[] arguments) {  
    test(6);  
    test(5);  
    test(4);  
}
```

Comparison operators

$x > y$: x is greater than y

$x < y$: x is less than y

$x \geq y$: x is greater than or equal to x

$x \leq y$: x is less than or equal to y

$x == y$: x equals y

(equality: $==$, assignment: $=$)

Negation Operator

“!” flips true/false

```
if (x >= y) {
```

```
...
```

```
}
```

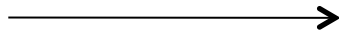
Negation Operator

“!” flips true/false

```
if (x >= y) {
```

```
...
```

```
}
```



```
if (!(x < y)) {
```

```
...
```

```
}
```

Boolean operators

&&: logical AND

||: logical OR

```
if (x > 6) {  
    if (x < 9) {  
        ...  
    }  
}
```

Boolean operators

&&: logical AND

||: logical OR

if (x > 6) { if (x < 9) { ... } }	→	if (x > 6 && x < 9) { ... }
---	---	------------------------------------

else

```
if (CONDITION) {  
    STATEMENTS  
} else {  
    STATEMENTS  
}
```

```
public static void test(int x) {  
    if (x > 5) {  
        System.out.println(x + " is > 5");  
    } else {  
        System.out.println(x + " is not > 5");  
    }  
}
```

```
public static void main(String[] arguments) {  
    test(6);  
    test(5);  
    test(4);  
}
```

else if

```
if (CONDITION) {  
    STATEMENTS  
} else if (CONDITION) {  
    STATEMENTS  
} else if (CONDITION) {  
    STATEMENTS  
} else {  
    STATEMENTS  
}
```

```
public static void test(int x) {  
    if (x > 5) {  
        System.out.println(x + " is > 5");  
    } else if (x == 5) {  
        System.out.println(x + " equals 5");  
    } else {  
        System.out.println(x + " is < 5");  
    }  
}
```

```
public static void main(String[] arguments) {  
    test(6);  
    test(5);  
    test(4);  
}
```


Variable Scope

Variables live in the block ({}) where they are defined (**scope**)

Method parameters are like defining a new variable in the method

```
class SquareChange {  
    public static void beforeAfterSquare(int x) {  
        System.out.println("before x = " + x);  
        x = x * x;  
        System.out.println("after x = " + x);  
    }  
  
    public static void main(String[] arguments) {  
        int x = 5;  
        System.out.println("main x = " + x);  
        beforeAfterSquare(x);  
        System.out.println("main x = " + x);  
    }  
}
```

```
class Scope {  
    public static void main(String[] arguments) {  
        int x = 5;  
        if (x == 5) {  
            int x = 6;  
            int y = 72;  
            System.out.println("x = " + x + " y = " + y);  
        }  
        System.out.println("x = " + x + " y = " + y);  
    }  
}
```

Questions?

Assignment: FooCorporation

Method to print pay based on base pay and hours worked

Overtime: More than 40 hours, paid 1.5 times base pay

Minimum Wage: \$8.00/hour

Maximum Work: 60 hours a week

Reminder

- Write **your own** code
- Homework due Tuesday 3pm on Stellar.

**finalPosition = finalPosition +
initialVelocity * fallingTime;
finalPosition = finalPosition + initialPosition;**

OR

**finalPosition += initialVelocity * fallingTime;
finalPosition += initialPosition;**

Division

Division (“/”) operates differently on integers and on doubles!

Example:

```
double a = 5.0/2.0; // a = 2.5
```

```
int b = 4/2; // b = 2
```

```
int c = 5/2; // c = 2
```

```
double d = 5/2; // d = 2.0
```


Conversion by method

int to String:

```
String five = 5; // ERROR!
```

```
String five = Integer.toString (5);
```

```
String five = "" + 5; // five = "5"
```

String to int:

```
int foo = "18"; // ERROR!
```

```
int foo = Integer.parseInt ("18");
```

Comparison operators

- Do NOT call `==` on doubles! EVER.

```
double a = Math.cos (Math.PI / 2);  
double b = 0.0;
```

`a = 6.123233995736766E-17`

`a == b` will return FALSE!