# **6.S092**

Lecture 2

#### **Last Class**

- Administrative Stuff
- Primitive Data Type
- Primitive Data Structures
- Rules and Convention for naming
- Assignment to a variable
- Casting

We will go over these quickly.

#### **Primitive Data Types**

boolean

• byte 8-bit

• short 16-bit

• int 32-bit

• long 64-bit

• float 32-bit

double 64-bit

http://docs. oracle. com/javase/tutor ial/java/nutsand bolts/datatypes. html

#### **Primitive Data Structures**

- arrays
- String

#### arrays

Fixed sized immutable data structure.

Can be of primitives or Objects

#### Naming Rules

- Cannot start with a number
- Cannot use JAVA's reserved words
- Cannot start with special characters except '\$' and '\_'
- Cannot contain whitespace
- Case Sensitive

#### **Naming Conventions**

- Names should be meaningful
- Constants are all CAPS
- variable should start with a lowercase
- Use camelcase
- camelcaseLooksLikeThis

#### **JAVA Reserved Words**

```
http://docs.oracle.
com/javase/tutorial/java/nutsandbol
ts/_keywords.html
```

### Variable Assignment

(dataType) (name) = (theStuffYouWantItToBe);

dataType ∈ {int, double, String, Object, ...}
name ∈ {any valid name following the rules}
= is the operator that assigns the stuff on the right to the variable on the left.

## Variable Assignment Examples

```
String s = \text{"Hi"};
int two = 2;
double pi = 3.1415;
double e = 2.718;
```

JAVA is STATICALLY TYPED!!!

### array assignment

Use the "[]" qualifier
Two different ways to initialize

#### array example

```
int[] naturalNum = {1,2,3,4,5};
int fiveThings [] = new int[5];
String[] names = new String[65];
```

#### Casting

- Upcasting (Java may do this automatically)
  - Going from an int to a double
  - Going from a subclass to a superclass
    - We'll learn this later in the course
- Downcasting (YOU must force it)
  - Going from a double to an int
  - Going from a superclass to a subclass
    - We'll learn this later in the course

## **Casting Example**

```
int five = 5;
double sixPointFive = five + 1.5;
```

NOTE: Upcasting and String Concatenation int five = 5;

String fiveInString = "Five = " + five;

# Casting Example Cont.

```
double pi = Math.PI; // 3.141592...
int three = pi; // will NOT compile
int three = (int) pi; // will compile
```

Dog avalanche = new Dog(); Labrador lab = (Labrador) avalance; // Only work if Labrador EXTENDS Dog

#### **New Topics**

- Operators
- Commenting
- Scope
- Control Flow
  - if, if/else, if/else if, if/else if/else
- Loops
  - while, do-while, for, for each

### **New Topics Cont.**

- Printing to console
- Accepting user input
- Privacy
- Methods and Modularity
  - Naming
  - Return type and Arguments
  - The "return" statement

### **Operator - Order of Operations**

The following slides on operator follow the order of operations.

Note: Extensive list <a href="http://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.">httml</a>

### **Unary Operators**

Postfix : expr++ or expr --

Prefix : ++expr or --expr

Negative : -

Negate:!

## **Binary Operators in Order of Operation**

Multiplication - \*

Division -/

Modulo - %

Addition -+

Subtraction - -

Assignment -=

#### **Relational Operators**

Less than - <

Greater than ->

Less than or equal to -<=

Greater than or equal to - >=

Instance of - instance of

## **Equality Operators**

Equals -==

Not equal -!=

## **Logical Operators**

And - &&

Or - ||

#### **Ternary Operators**

If / Else -?:

# **Assignment**

Happens Last

# Commenting

Line Comment : Begins with a '//'
Anything after // is a comment

Block Comment: Begins with a '/\*' ends with '\*/'
Anything inside /\* and \*/ are comments

#### **JavaDocs**

JavaDocs are the description of the things you use and do in Java such as variables and methods.

#### JavaDocs Cont.

JavaDocs begin with /\*\* and end with \*/
Anything inside the /\*\* and \*/ will be
displayed if ones hovers over methods and
variables that it describes.

We will see examples of all types of commenting in the code today.

## Scope

A variable "lives" inside of the braces

Not like C/C++ where you need to allocate memory.

#### **Control Flow - if**

```
if (booleanExpression)
  statement;
if ( booleanExpression ) {
  statements;
```

#### Control Flow - if/else

```
if ( booleanExpression )
  statement;
  if ( booleanExpression ){
    statements;
else
    statement;
                                      statements;
```

#### Control Flow - if/else if

```
if ( booleanExpression )
    statement;
else if ( booleanExpression )
    statement;
```

NOTE: No limit on how many else if; you can have as many as you need. Just like the previous cases braces allow for multiple statements.

#### Control Flow - if/else if/else

```
if ( booleanExpression )
                              NOTE: No limit on
                              how many else if;
  statement;
                              you can have as
else if (booleanExpression)
                              many as you need.
                              Just like the
  statement;
                              previous cases
else
                              braces allow for
                              multiple
  statement;
                              statements.
```

#### **Loops - while**

```
while ( booleanExpression ) {
    statements;
}
```

Just like if/else structures you can also have a one line statement.

#### Loops - do-while

```
do{
    statements;
}while ( booleanExpression );
```

#### Loops - for

```
for ( int i = 0; i < 10; i++ ) {
    statements;
}</pre>
```

#### Loops - for each

ONLY works on arrays and lists. It does something **for each** of the elements.

```
int[] nums = {1,2,3,4,5,6,7,8};
for ( int numbers : nums ){
    statements;
}
```

# WARNING: Modifying in a for each

```
int [] nums = \{3,4,5\};
for (int i : nums) {
  i += 1; // same as i = i + 1;
for (int i : nums) {
                                      The first for loop
  System.out.println(i);
                                      may not do what
                                      you want it to do.
```

## **Correct Way**

```
int [] nums = \{3, 4, 5\};
for ( int i = 0; i < nums.length; <math>i++ ) {
  nums[i]++; // nums[i] = nums[i]+1
for ( int i : nums ) {
  System.out.println(i);
```

# Two things you should take for granted for now

## Printing to console

```
System.out.print();
System.out.println();
System.out.printf();
```

### **Accepting user input**

Use the Scanner class

```
Scanner in = new Scanner(System.in);
String userIn = in.nextLine();
```

### Privacy - Who can see and manipulate

public : everyone

(default) : inside package

protected: subclasses

private : only the class

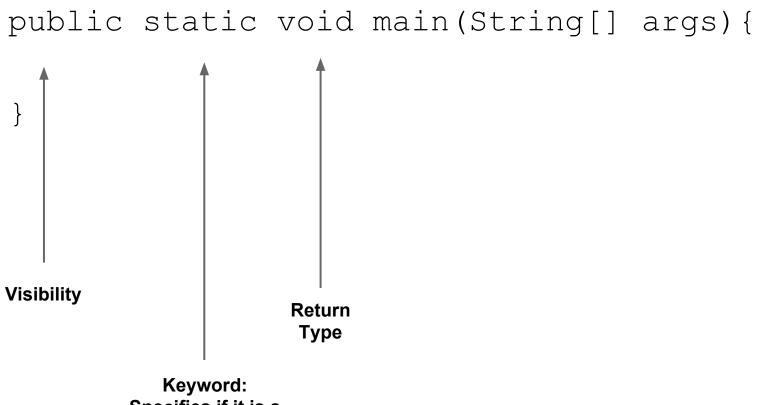
# Methods - Return Type/Naming/Args

First the basics

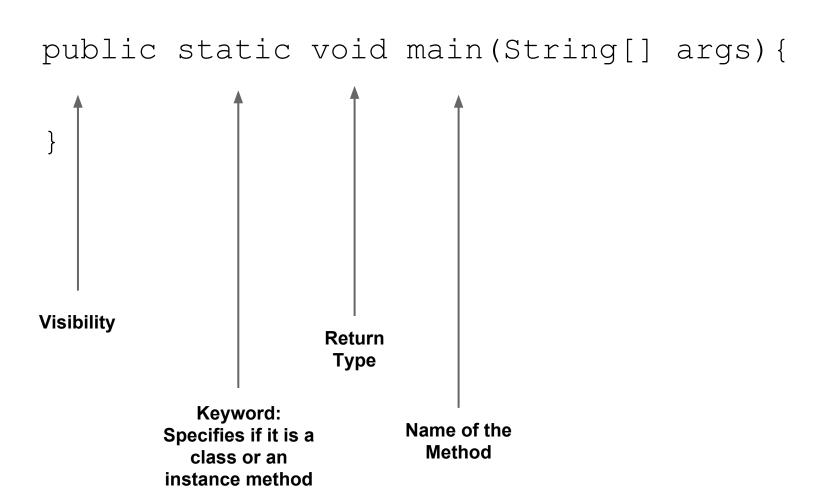
```
public static void main(String[] args) {
Visibility
```

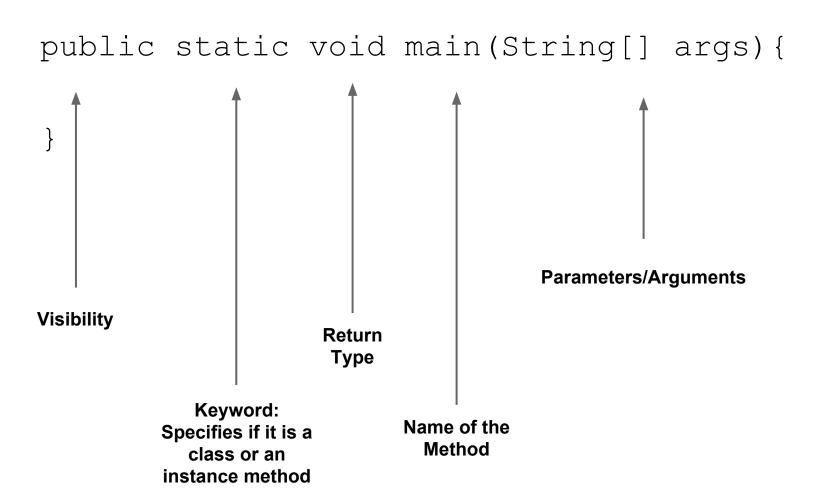
```
public static void main(String[] args) {
Visibility
            Keyword:
         Specifies if it is a
           class or an
```

instance method



Keyword:
Specifies if it is a
class or an
instance method





### Method - "return" statement

```
public int giveMeANumber() {
  return 42;
}
```

The return statement in a function returns the specified return type to where the function was called.

### **Lets CODE!**