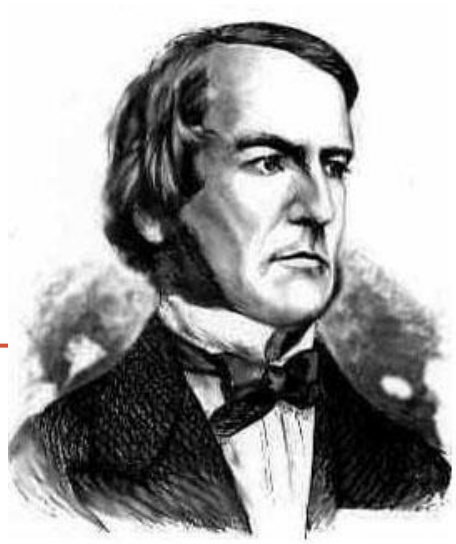
BASIC COMPUTATION



x

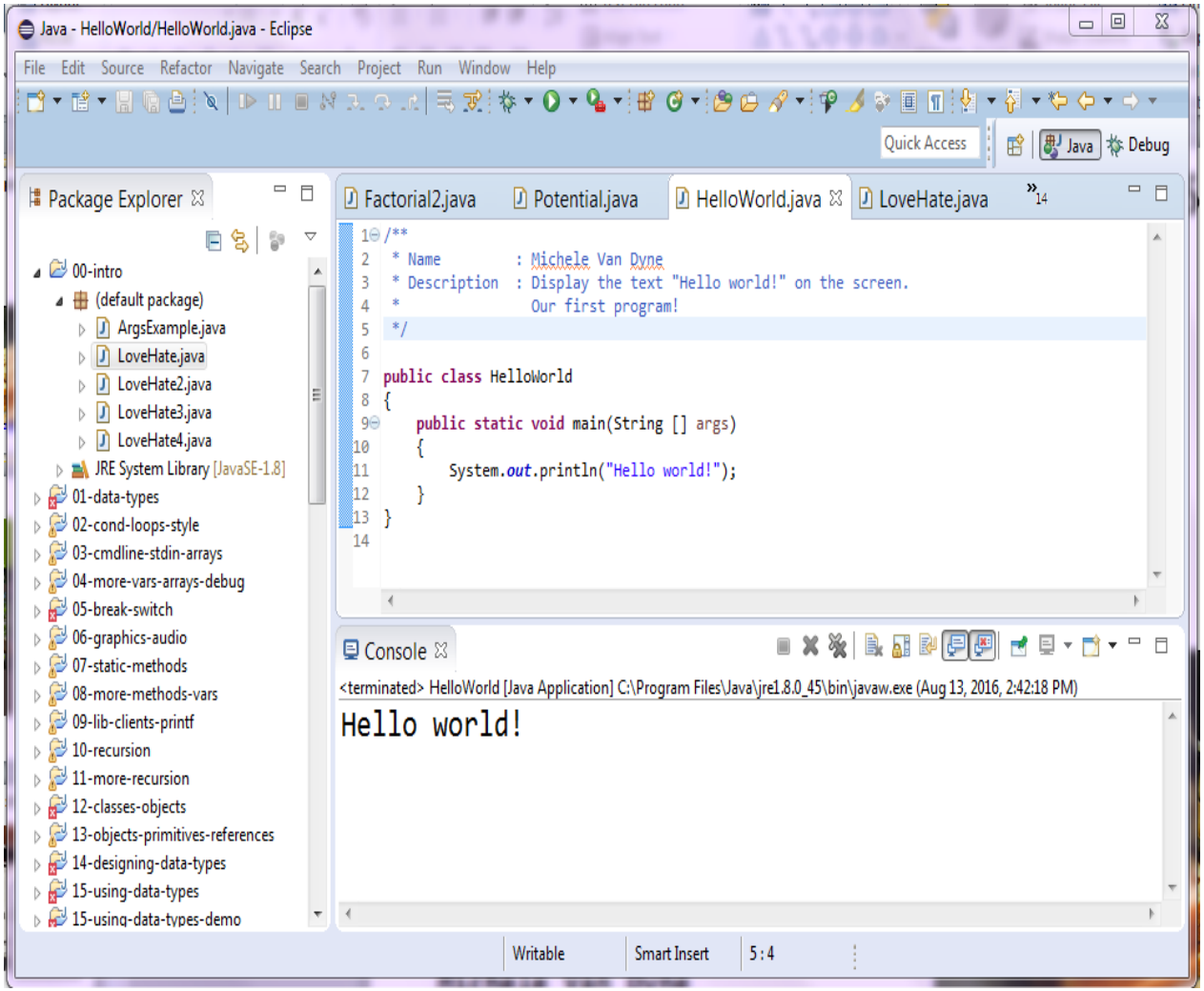
```
public static void main(String [] args)
```

Outline

- Using Eclipse
- Data Types
  - Variables
  - Primitive and Class Data Types
- Expressions
  - Declaration
  - Assignment

Eclipse

- Eclipse IDE (Integrated Development Environment)



Comments

- A comment can begin with //.
  - Everything after these symbols and to the end of the line is treated as a comment and is ignored by the compiler.
  - `double radius; //in centimeters`
- A comment can begin with `/*` and end with `*/`
  - Everything between these symbols is treated as a comment and is ignored by the compiler.
- A *javadoc* comment, begins with `/**` and ends with

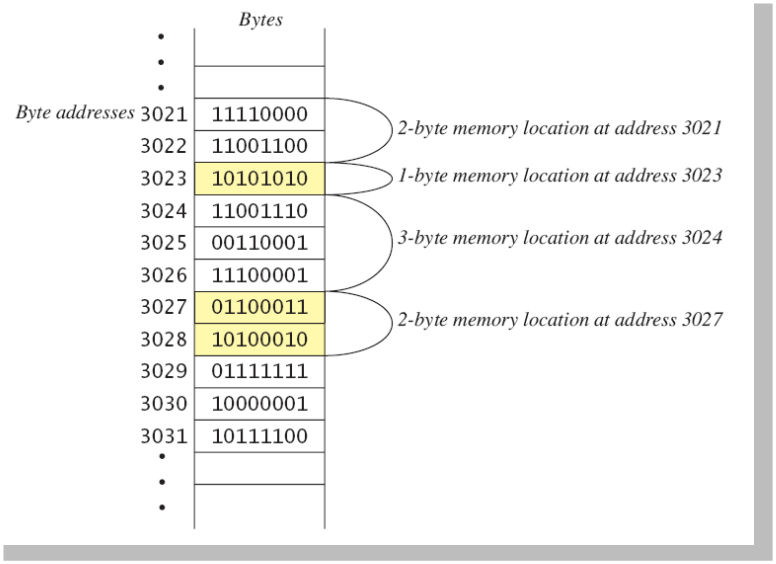
```
/**
This program should only
be used on alternate Thursdays,
except during leap years, when it s
only be used on alternate Tuesdays.
*/
```

Try It!

- Open Eclipse
- Browse to the directory you want to save your work in
  - Flash drive if you have one, D:*YourName*\Workspace if you don't
- Select File – New – Java Project, name it Activity01
- Under Project Layout, select "Use project folder as root…" then click Finish
- Right click on Activity01, select New, then Class, name it HelloWorld
  - If you click the box "public static void main…" Eclipse will create the main method for you
- Copy the HelloWorld program into the editor
  - Type it in OR
  - Copy and paste it from the slides (you'll need to edit the quote marks) OR
  - Copy and paste it from the website
  - NOTE: If you selected the "public static void main…" in the previous step, you only need to add one line of code
- Save and Run your program, see if it prints out "Hello world!"
- Make sure your name is in the comment section at the top of the program (not my name)
- Open Moodle, go to CSCI 135, Section 01
- Open the Activity01 dropbox for today
- Drag and drop your program file to the Moodle dropbox
  - You can drag from the Eclipse file menu right into Moodle
  - Wait until the blue line goes away and you see an icon for your file
- You get: 1 point if you turn in something, 2 points if you turn in something that is correct.

Variables

- *Variables* store data such as numbers and letters.
  - Think of them as places to store data.
  - They are implemented as memory locations.
- The data stored in a variable is called its *value.*
  - The value is stored in the memory location.
- Its value can be changed.



Some Definitions

```
int a;
```

Declaration statement

"I'm going to need an integer and let's call it a"

REMEMBER: in Java you are *required* to declare a variable before using it!

```
a = 10;
```

Variable name

"Whenever I say a, I mean the value stored in a"

Literal

"I want the value 10"

= in CS
is not the same
as
= in math!

```
int b;
b = 7;
```

Assignment statement

"Variable b gets the literal value 7"

```
int c = a + b;
```

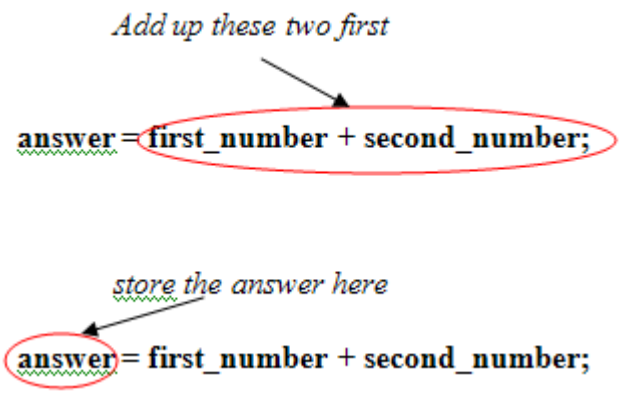
Combined declaration and assignment

"Make me an integer variable called c and assign it the value obtained by adding together a and b"

Assignment Evaluation

- The expression on the right-hand side of the assignment operator (=) is evaluated first.
- The result is used to set the value of the variable on the left-hand side of the assignment operator.

```
score = numberOfCards +  handicap;
eggsPerBasket = eggsPerBasket - 2;
```



Data Types

- A *primitive type* is used for simple, non-decomposable values such as an individual number or individual character.
  - `int, double,` and `char` are primitive types.
- A *class type* is used for a class of objects and has both data and methods.
  - `"Java is fun"` is a value of class type `String`

OFFICIAL

DEFINITION

- A ***data type*** is a set of values and the legal operations on those values.

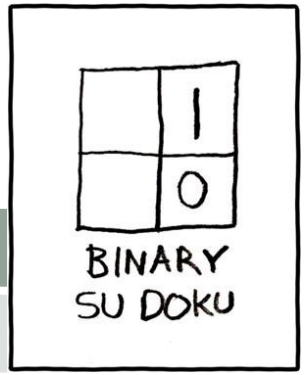
Variables and Data Types

- Variables
  - Stores information your program needs
  - Each has a unique name
  - Each has a specific type

Java built-in type	what it stores	example values	operations
`int`	integer values	`42` `1234`	add, subtract, multiply, divide, remainder
`double`	floating-point values	`9.95` `3.0e8`	add, subtract, multiply, divide
`char`	characters	`'a', 'b', '!'`	compare
`String`	sequence of characters	`"Hello world!"` `"I love this!"`	concatenate
`boolean`	truth values	`true` `false`	and, or, not

Primitive Type Sizes

- Primitive types
  - Just a block of memory in your computer
  - Size of block measured in bits (number of 0s or 1s)
  - Integers:



type	bits	example
`byte`	8	0110 1110
`short`	16	0110 1110 1101 1101
`int`	32	0101 1001 0000 0001 0111 1101 0110 0010
`long`	64	1101 0011 1001 0001 1101 0101 1010 0101 0111 1010 0011 1010 1011 1100 1111 1111

Creating a Primitive Variable

```
byte x;
```

x

x

```
01101010101010010101010111010101010101010101011101010
00010101010101011111010101010101010101010010011010101
01001010101010110000000001010101010111010101000101010
10101111010101010101010101010100100110101010101001010101
01110101010101010101010111010101000101010101010111101 0
10101010101010101010010011010101010100101010101110101010
10101010101011110101010001010101010101111010101010101
01010100100110101010101001010101011101010101010101010101
```

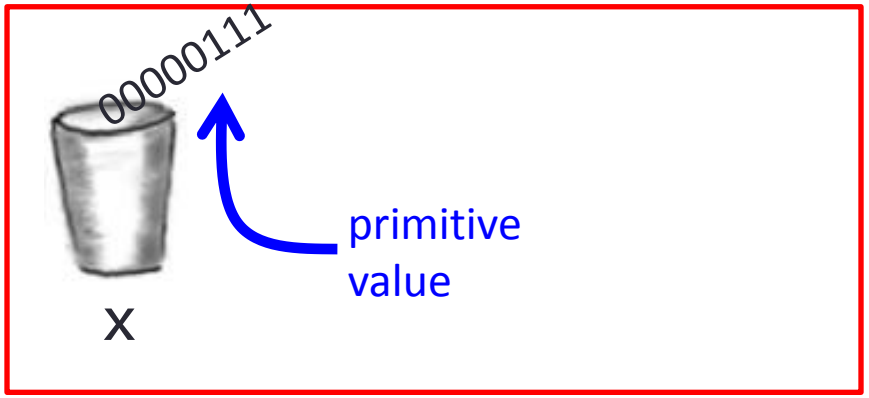
Initializing Variables

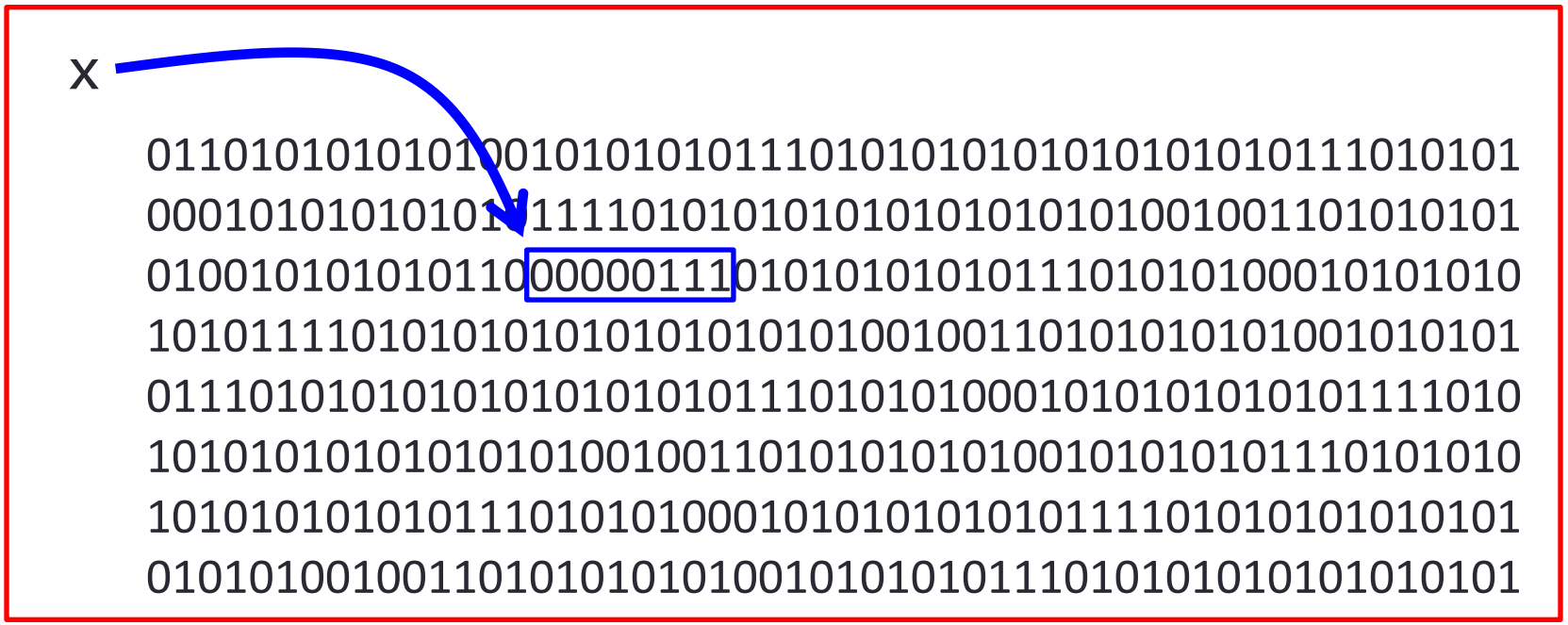
- A variable that has been declared, but not yet given a value is said to be *uninitialized.*
- Uninitialized class variables have the value **`null`**.
- Uninitialized primitive variables may have a default value.
  - It's good practice ***not*** to rely on a default value.
- To protect against an uninitialized variable (and to keep the compiler happy), assign a value at the time the variable is declared.
- Examples:

```
int count = 0;
char grade = 'A';
```

Creating and Initializing a Primitive Variable

```
byte x = 7;
```





Integers

- `int` data type
  - An integer value between -2^{31} and $+2^{31}-1$
    - Between -2,147,483,648 and 2,147,483,647
  - Operations:

add	subtract	multiply	divide	remainder
`+`	`-`	`*`	`/`	`%`

Watch out for this!
/ is integer division if both sides are integers!

example	result	comment
`10 + 7`	`17`	
`10 - 7`	`3`	
`10 * 7`	`70`	
`10 / 7`	`1`	integer division, no fractional part
`10 % 7`	`3`	remainder after dividing by 7
`10 / 0`		runtime error, you can't divide an integer by 0!

Increment and Decrement Operators

- Used to increase (or decrease) the value of a variable by 1
- Easy to use, important to recognize
- The increment operator

  `count++` or **`++count`**
- The decrement operator

  `count--` or **`--count`**

Equivalent operations:

```
count++;
++count;
count = count + 1;

count--;
--count;
count = count - 1;
```

Floating-Point Numbers

- `double` data type
  - Floating-point number (as specified by IEEE 754)
  - Operations:

add	subtract	multiply	divide
+	-	*	/

example	result
9.95 + 2.99	12.94
1.0 - 2.0	-1.0
1.0 / 2.0	0.5
1.0 / 3.0	0.3333333333333333
1.0 / 0.0	Infinity
0.0 / 123.45	0.0
0.0 / 0.0	NaN

e Notation

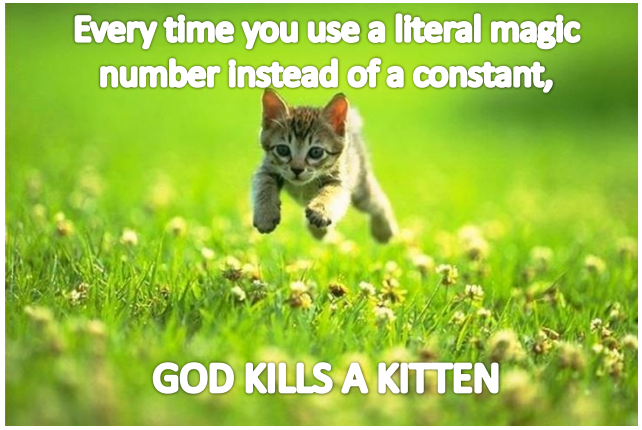
- e notation is also called *scientific notation* or *floating-point notation.*
- Examples
  - `865000000.0` can be written as `8.65e8`
  - `0.000483` can be written as `4.83e-4`
- The number in front of the `e` does not need to contain a decimal point.

Imprecision in Floating-Point Numbers

- Floating-point numbers often are only approximations since they are stored with a finite number of bits.
- Hence `1.0/3.0` is slightly less than 1/3.
- `1.0/3.0 + 1.0/3.0 + 1.0/3.0` is less than 1.

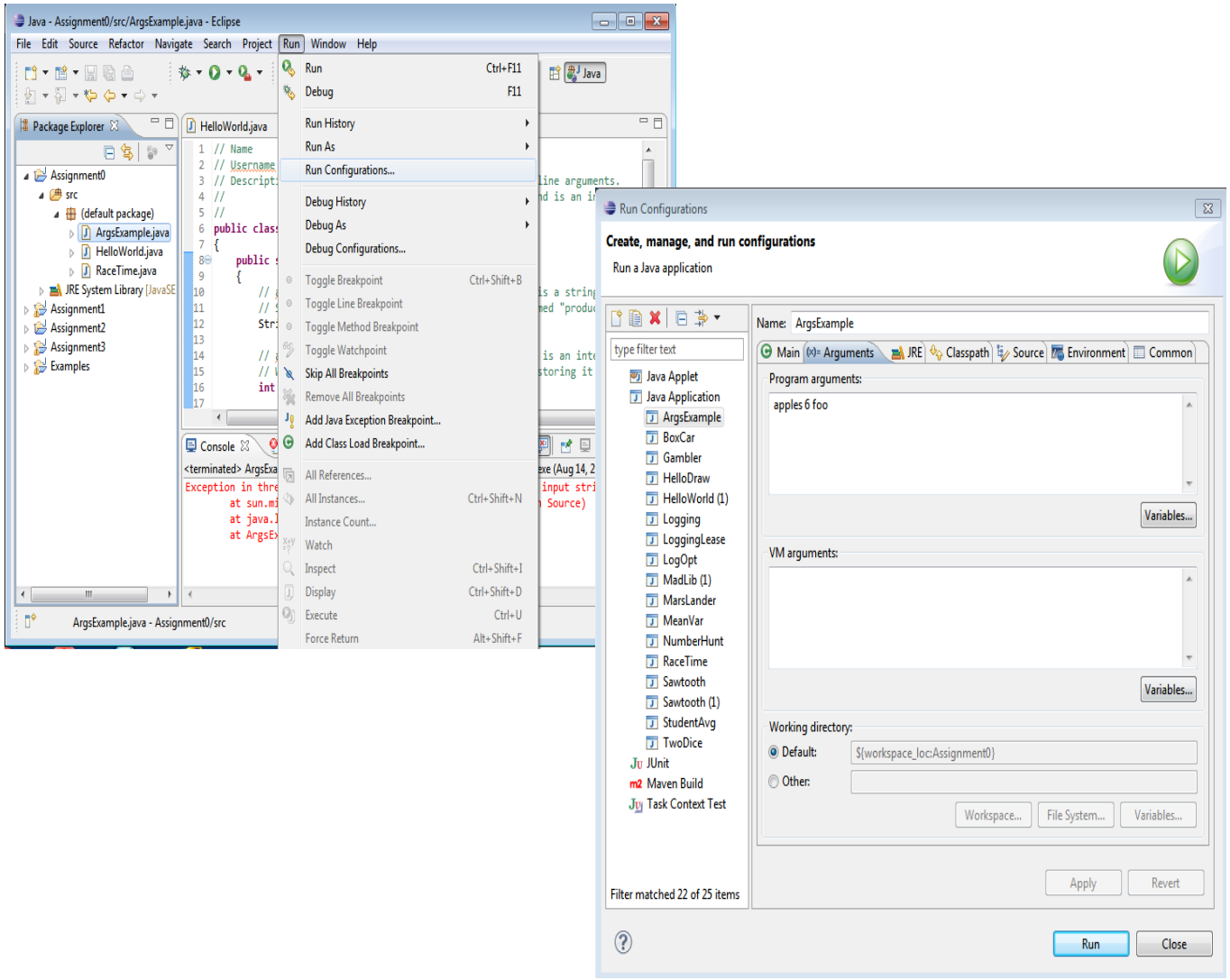
Constants

- Literal expressions such as `2, 3.7,` or `'y'` are called *constants*.
- Integer constants can be preceded by a **+** or **–** sign, but cannot contain commas.
- Floating-point constants can be written
  - With digits after a decimal point or
  - Using *e notation.*
- Java provides mechanism to …
  - Define a variable
  - Initialize it
  - Fix the value so it cannot be changed




```
public static final double PI = 3.14159;
```

Command line args in Eclipse



Summary

- Using Eclipse
- Data Types
  - Variables
  - Primitive and Class Data Types
- Expressions
  - Declaration
  - Assignment