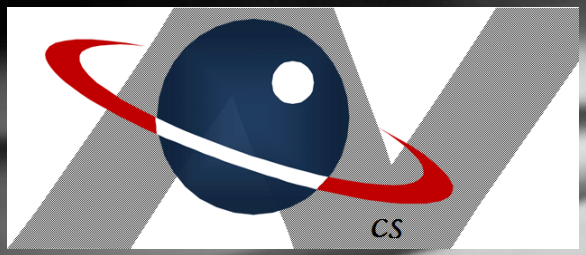
O(N) CS LESSONS

Lesson 2C – More Operations



By John B. Owen

All rights reserved

©2011, revised 2015

Table of Contents

Objectives

- The study of JAVA operations continues as we explore the concepts of reassignment, operation shortcuts, type casting, String conversion, division and modulus, and two Math class functions (pow and sqrt).

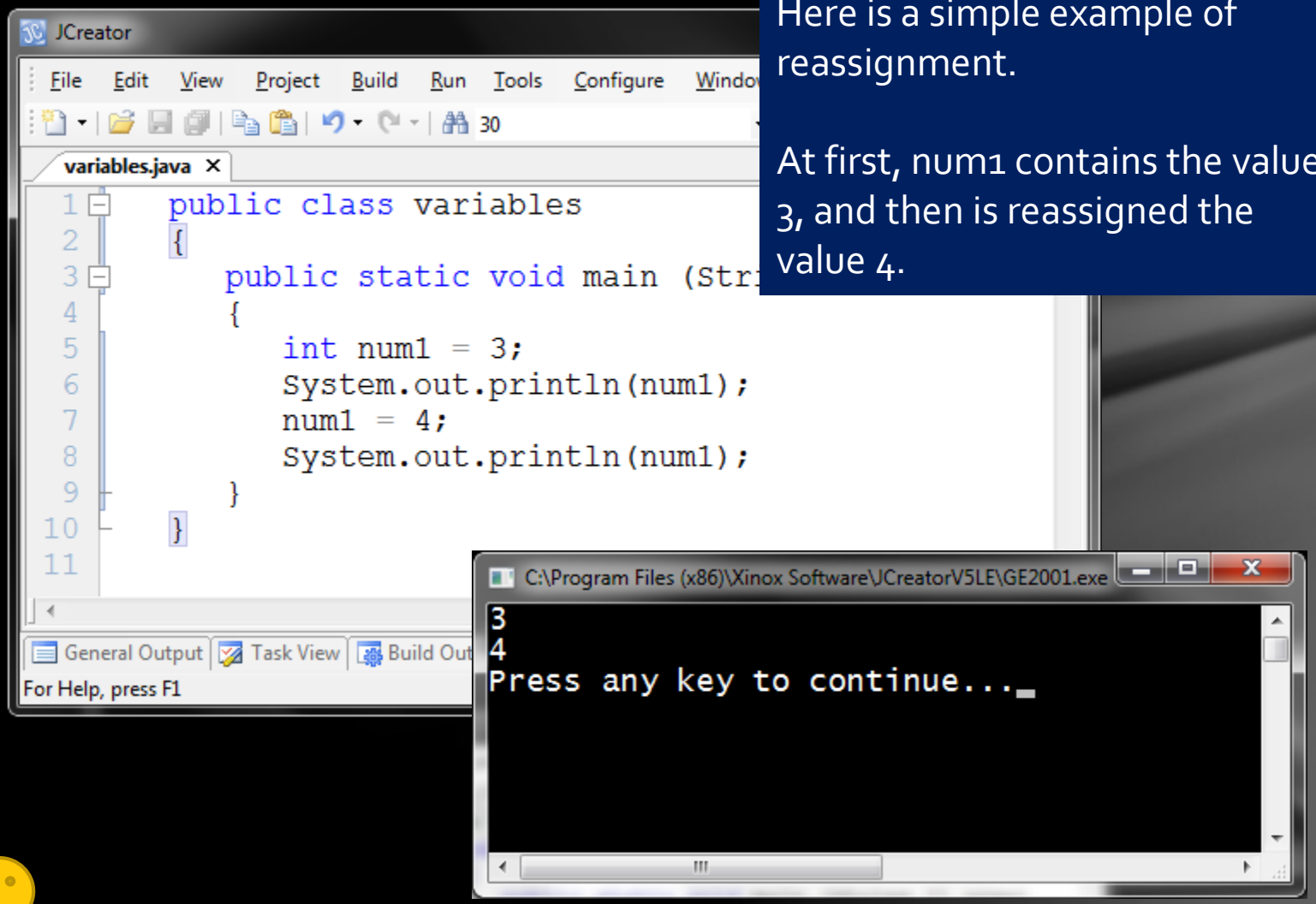
Reassignment of variables

- Variable reassignment is a major part of computer programming.
- Once variables have been initialized, they can be changed during a program, hence the name "variable".
- Constants, on the other hand, receive their first value and remain "constant" throughout the execution of the program, again, hence the name.

Reassignment of variables

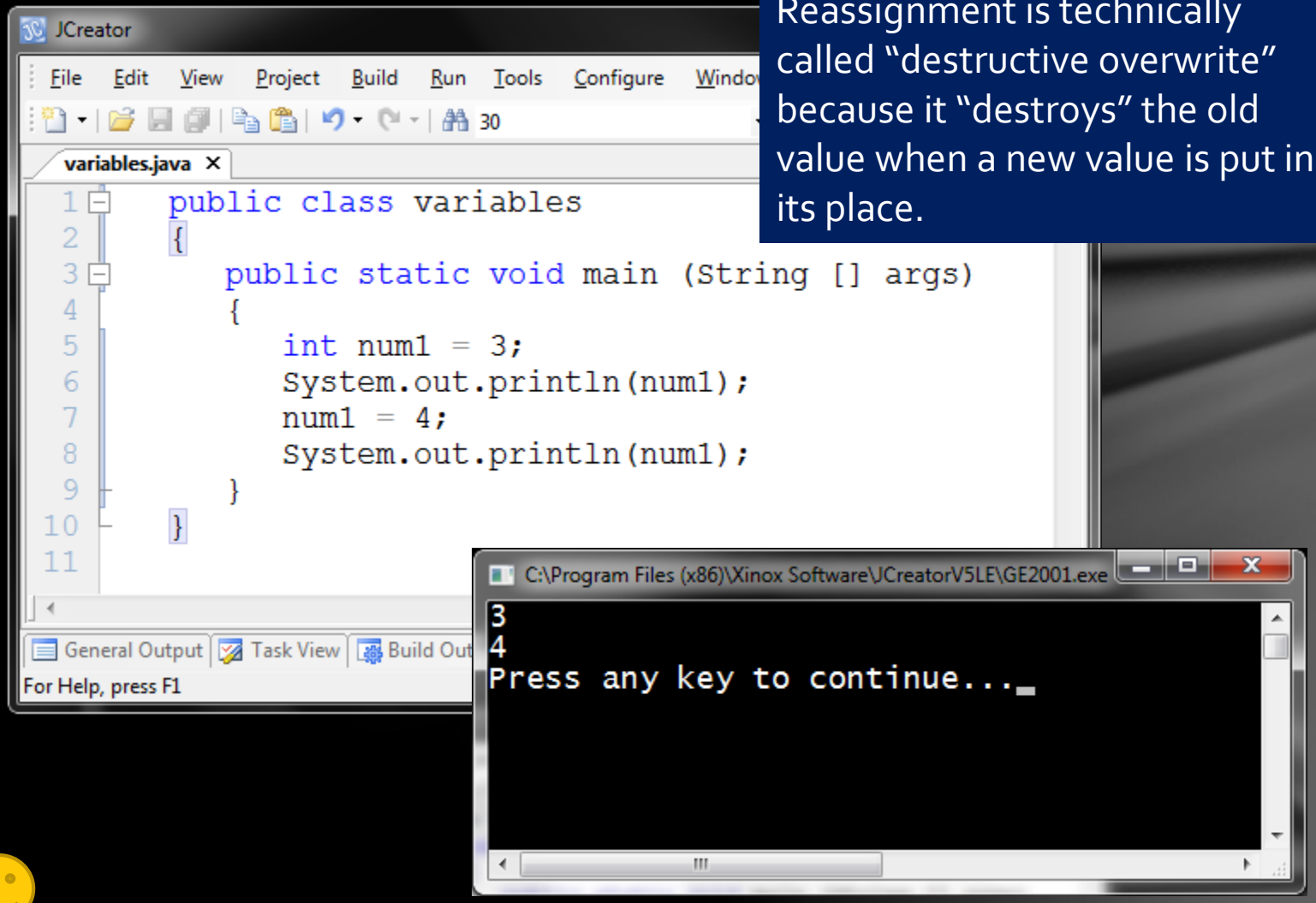
Here is a simple example of reassignment.

At first, num1 contains the value 3, and then is reassigned the value 4.



Destructive overwrite

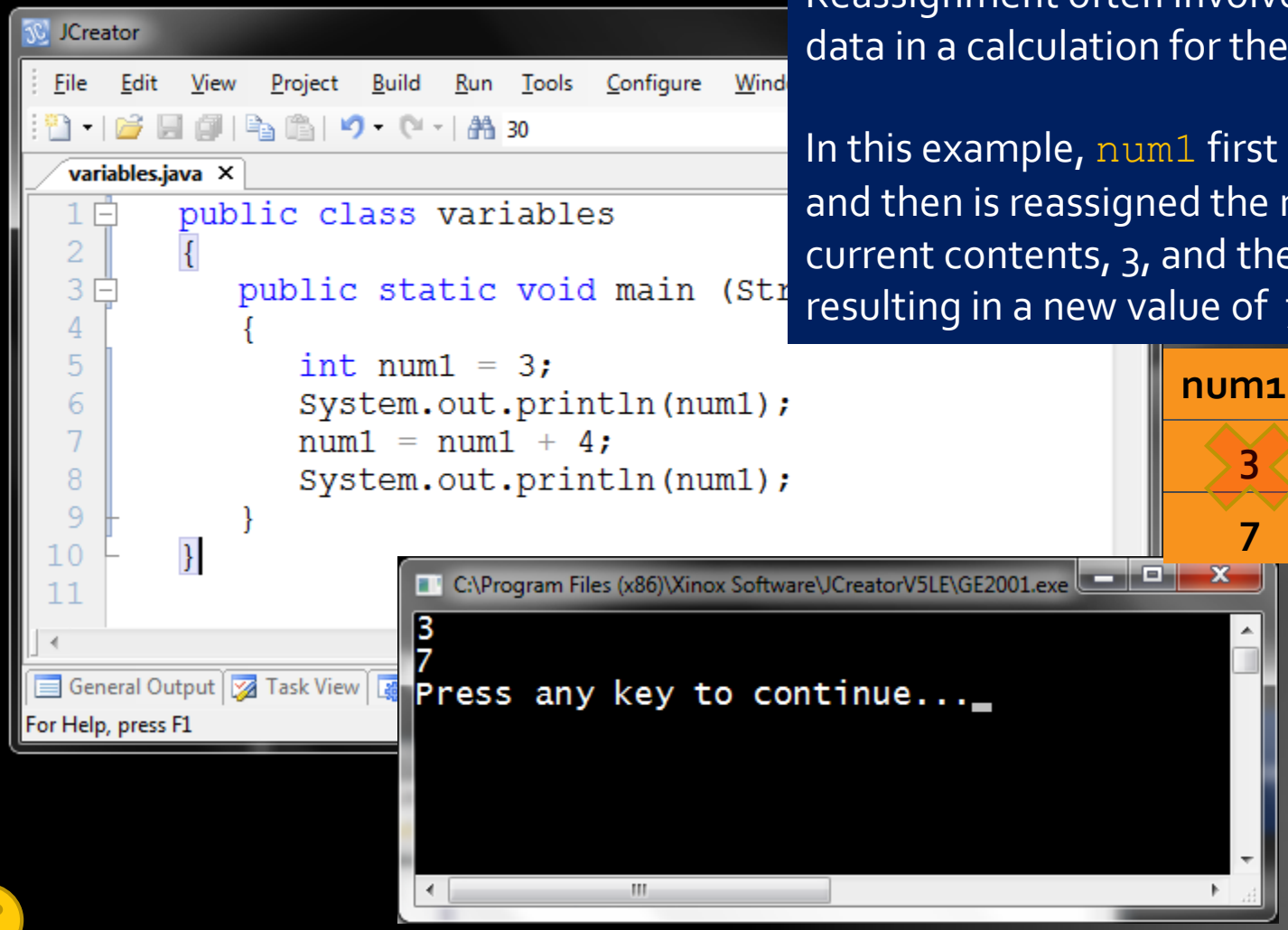
Reassignment is technically called "destructive overwrite" because it "destroys" the old value when a new value is put in its place.



Special reassignment

Reassignment often involves using the old data in a calculation for the new value.

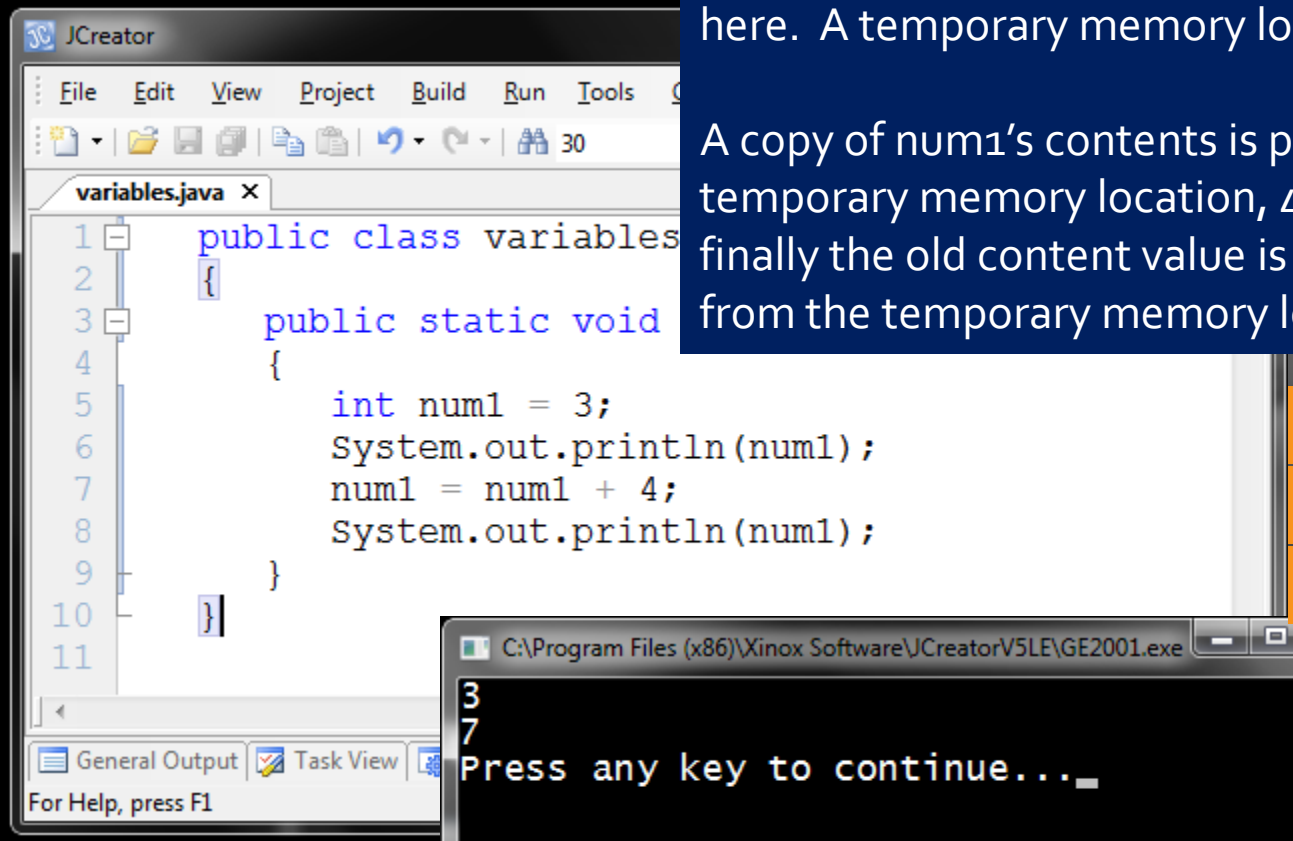
In this example, `num1` first contains a 3, and then is reassigned the result of its current contents, 3, and the literal value 4, resulting in a new value of 7

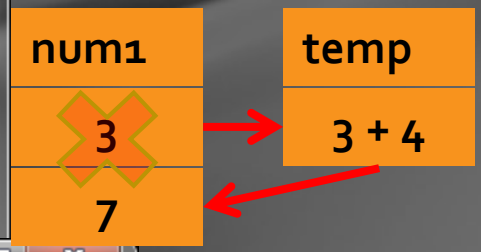


Special reassignment

The "right to left" concept we discussed earlier applies here. A temporary memory location is used to do this.

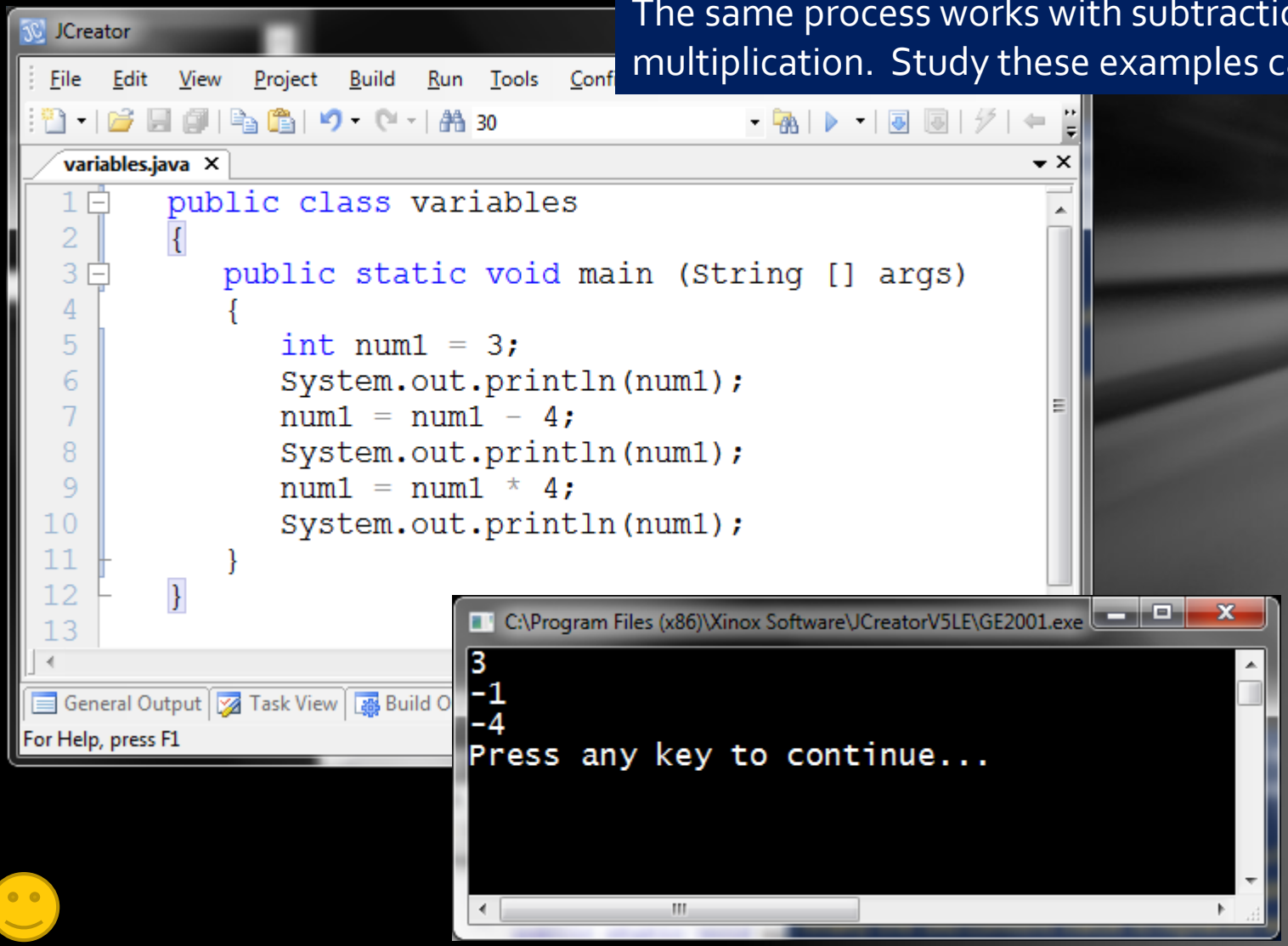
A copy of num1's contents is pasted into the temporary memory location, 4 is then added, and finally the old content value is replaced with the result from the temporary memory location.





Special reassignment

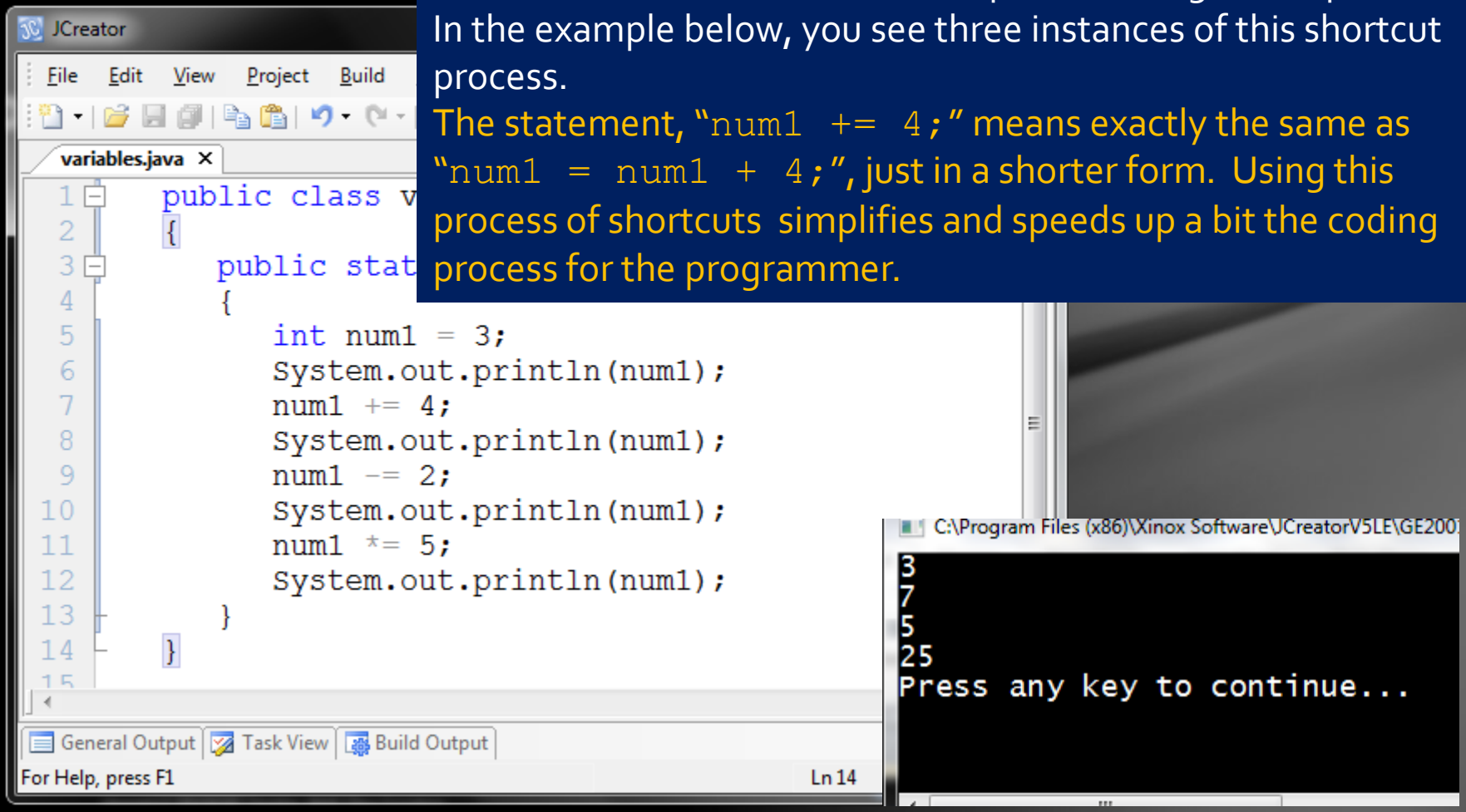
The same process works with subtraction and multiplication. Study these examples carefully.



Shortcut operations

There are shortcut forms for this special reassignment process. In the example below, you see three instances of this shortcut process.

The statement, "`num1 += 4;`" means exactly the same as "`num1 = num1 + 4;`", just in a shorter form. Using this process of shortcuts simplifies and speeds up a bit the coding process for the programmer.



Increment and Decrement

Quite often in programming constructs such as loops, it is necessary to repeatedly add or subtract by 1. This is referred to as incrementing or decrementing.

These ***unary*** operators are even shorter "extreme shortcuts" for this process, and you saw an example of this in an earlier lesson.

JCreator

File Edit View Project Buil

variables.java

```
1   public class
2   {
3       public st
4       {
5           int num1 = 3;
6           System.out.println(num1);
7           num1++;
8           System.out.println(num1);
9           num1--;
10          System.out.println(num1);
11          ++num1;
12          System.out.println(num1);
13          --num1;
14          System.out.println(num1);
15
16      }
17  }
```

General Output | Task View | Build Output

Help, press F1 Ln 17 Col 5 Char 5 NU

C:\Program Files (x86)\Xinox Software\J...

```
3
4
3
4
3
Press any key to continue...
```

Increment and Decrement

For now, there is no appreciable difference between "num1++" (post-increment) and "++num1" (pre-increment), or "num1--" (post-decrement) and "--num1" (pre-decrement). Later on, we will explore a subtle difference, but for now, as a standalone statement, they accomplish the same thing, that is, increment by one, or decrement by one.

JCreator

File Edit View Project Build

variables.java

```
1   public class
2   {
3       public st
4       {
5           int num
6           System.out.println(num1);
7           num1++;
8           System.out.println(num1);
9           num1--;
10          System.out.println(num1);
11          ++num1;
12          System.out.println(num1);
13          --num1;
14          System.out.println(num1);
15
16      }
17  }
```

General Output | Task View | Build Output

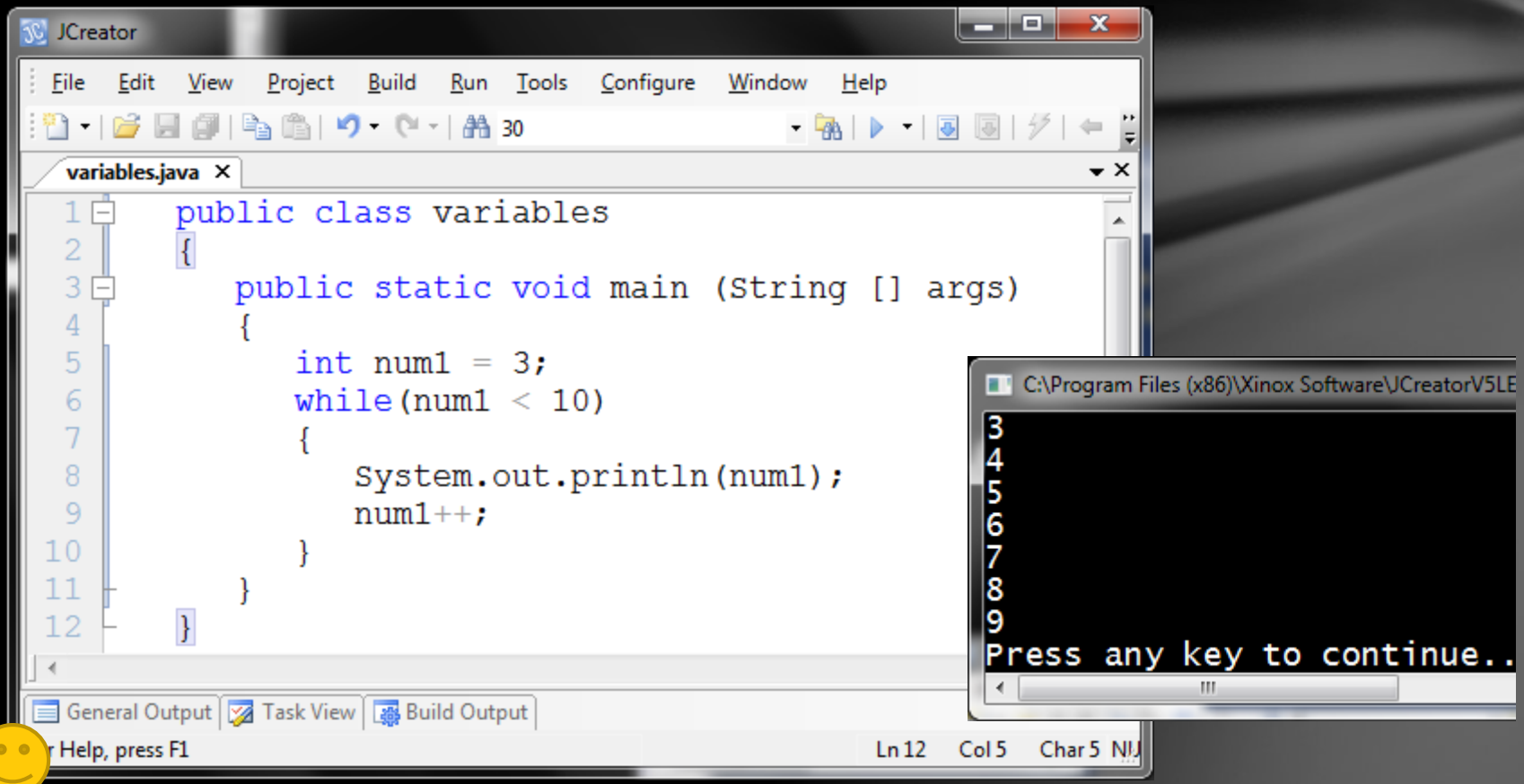
Help, press F1 Ln 17 Col 5 Char 5 NUM

C:\Program Files (x86)\Xinox Software\J...

```
3
4
3
4
3
Press any key to continue...
```

Loop example

Loops will be covered in more detail later on, but below is a quick example of how the increment statement, num1++, is used in a loop.



Type Casting with primitives

- As discussed earlier, there are some restrictions that occur when assigning values to different variable types.
- For example, you cannot assign a double value to an int variable.
- The compiler will throw an error: "possible loss of precision"

Type Casting with primitives

Below is an example of this "possible loss of precision" error. Even though we can clearly see that there will be no loss of the fractional part because it is 3.0, the compiler will not allow it.

casting.java

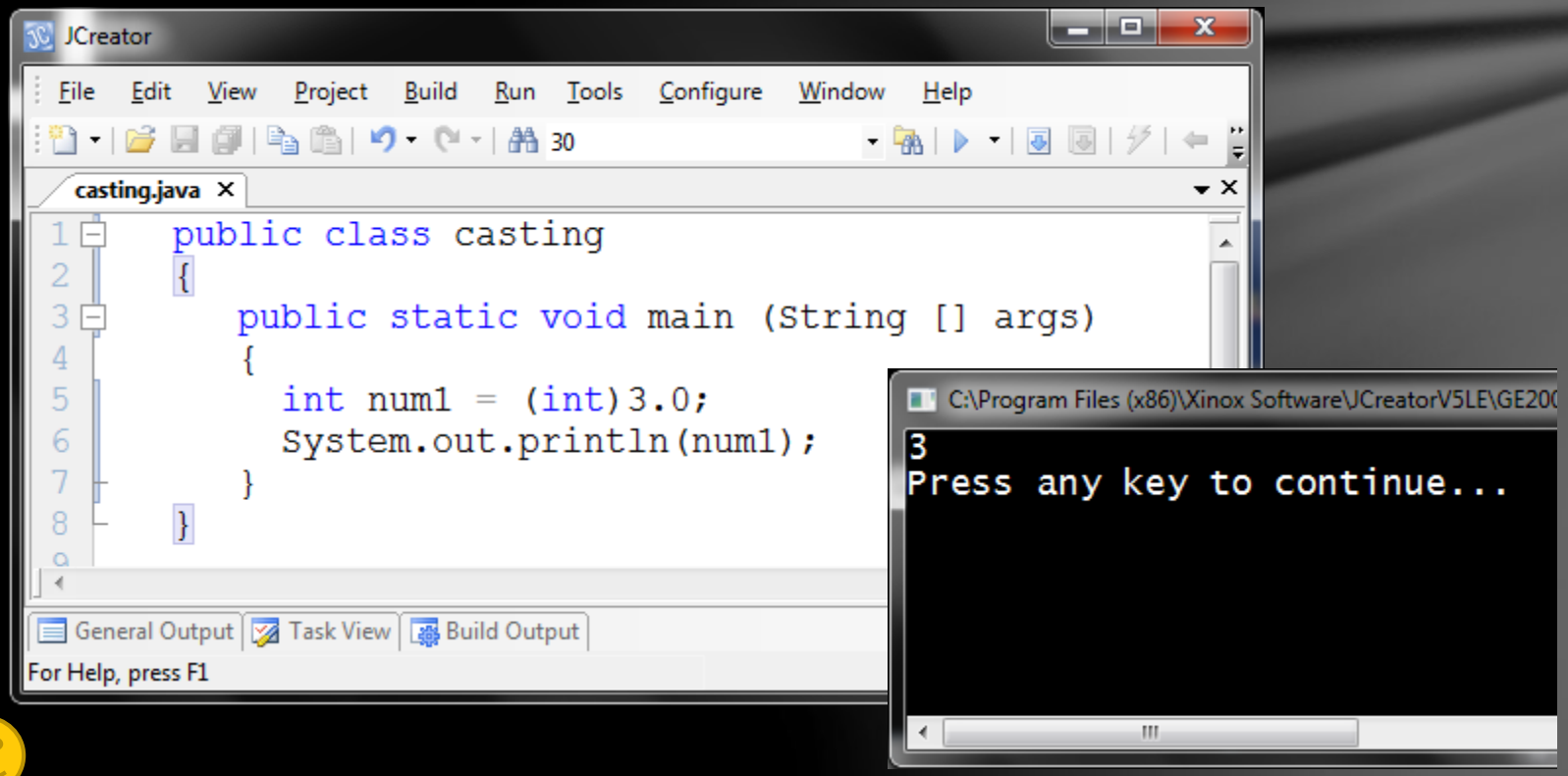
```
1   public class casting
2   {
3       public static void main (String [] args)
4       {
5           int num1 = 3.0;
6           System.out.println(num1);
7       }
8   }
9
```

Build Output

```
--------------------Configuration: <Default>--------------------
C:\Users\John Owen\Desktop\OWEN CS Lessons 2011\casting.java:5: possible
 loss of precision
found   : double
required: int
        int num1 = 3.0;
                   ^
```

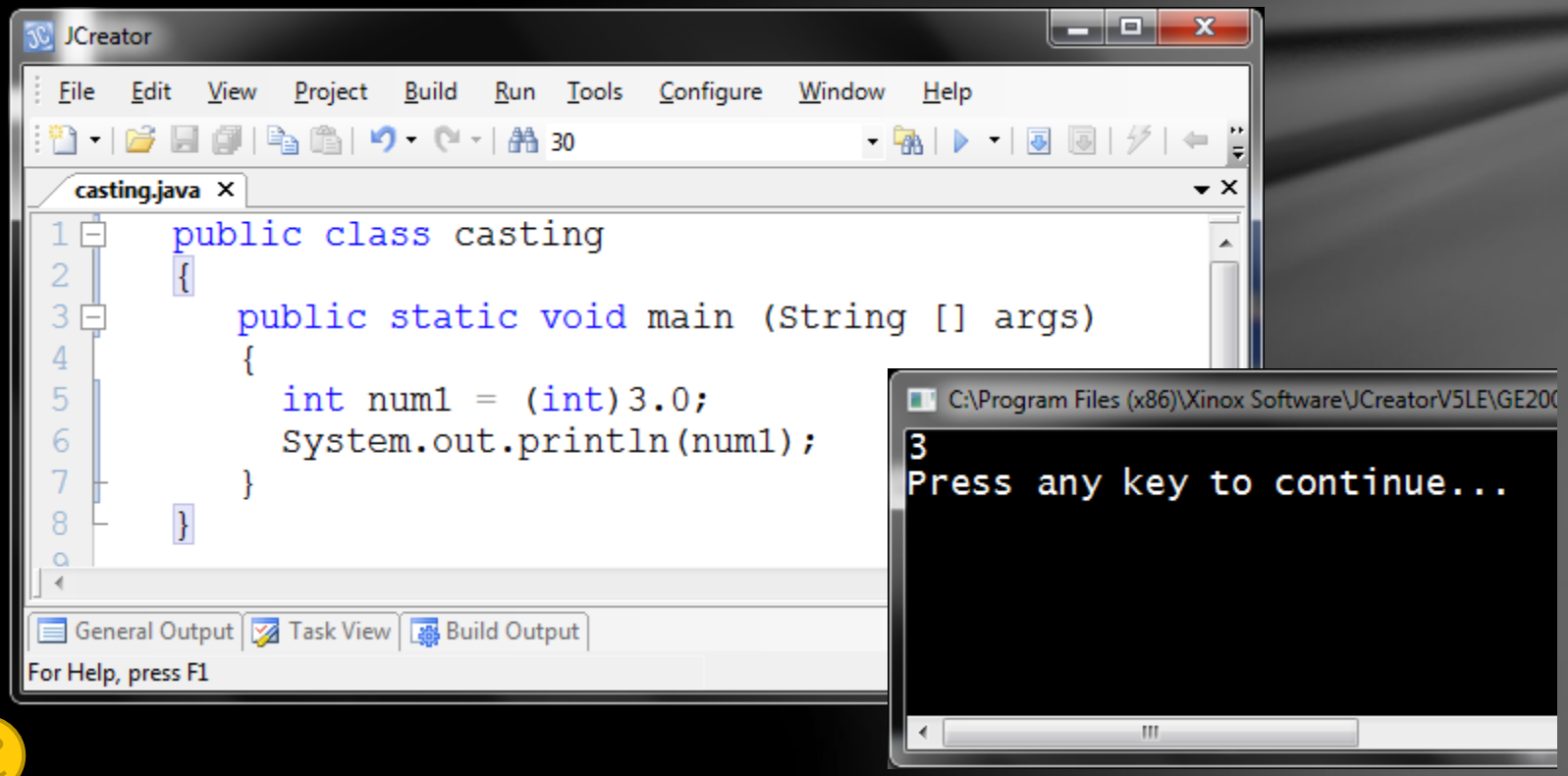
Type Casting with primitives

However, if you the programmer decide that you only want the whole number part of the value placed into the integer variable, and don't care that you lose the fractional part, you can override the compiler's ruling by ***casting*** the double as an int, like this: `(int)3.0`



Type Casting with primitives

Casting is a technique in programming that temporarily "forces" one type of data to behave like another, for the purposes of allowing unusual assignment statements, ones that the compiler doesn't normally allow.



Type Casting with primitives

In this example, a char cannot receive an int, for one primary reason: int values can possibly be negative, and a char always has positive values. Even though a char is a member of the integer family, it cannot directly receive ANY of the integer data types without casting.

casting.java

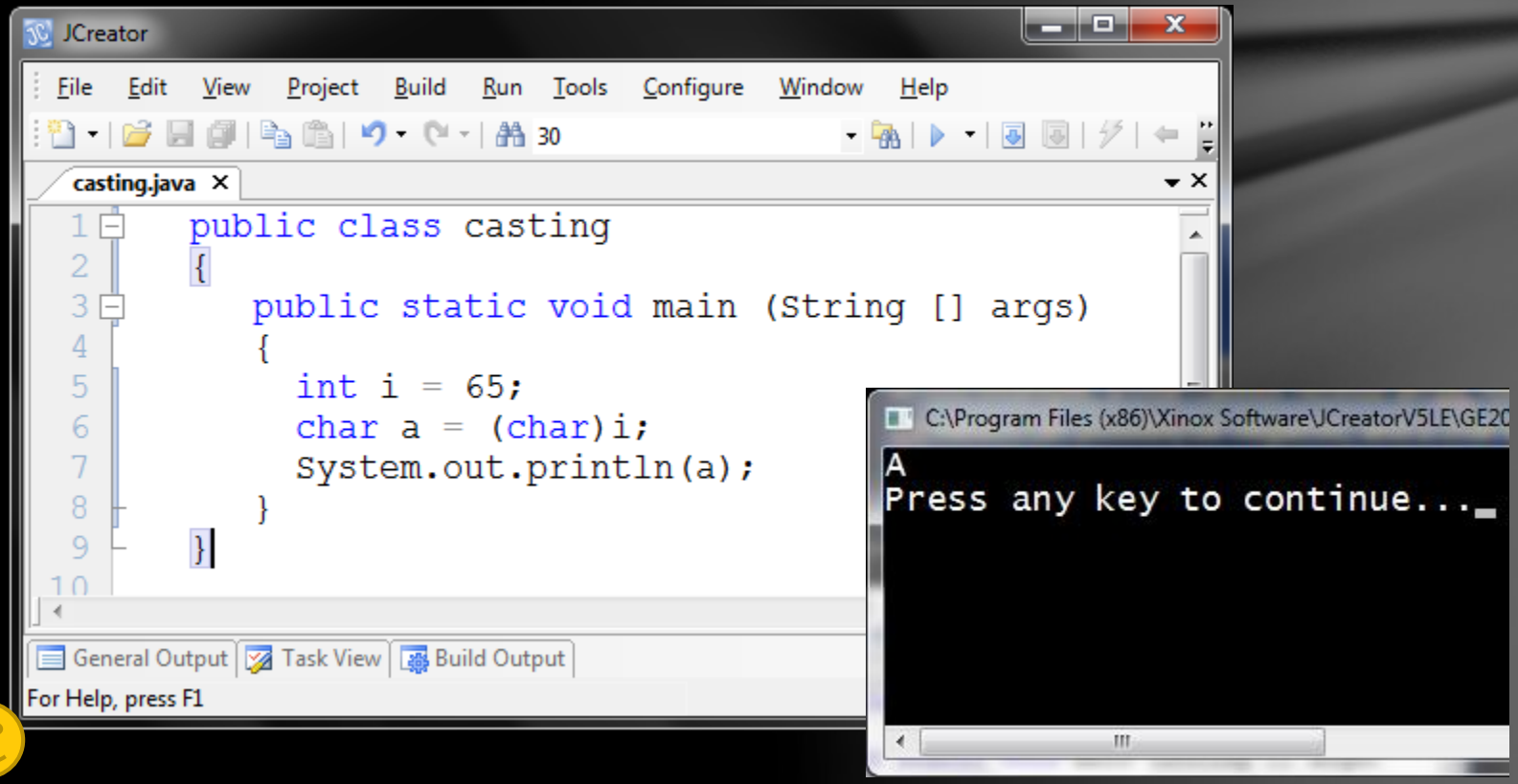
```
1   public cl
2   {
3      public static void main (String [] args)
4      {
5        int i = 65;
6        char a = i;
7        System.out.println(a);
8      }
9   }
```

Build Output

```
--------------------Configuration: <Default>--------------------
C:\Users\John Owen\Desktop\OWEN CS Lessons 2011\casting.java:6: possible
 loss of precision
found   : int
required: char
        char a = i;
                 ^
1 error
```

Type Casting with primitives

With a char cast, the int to char assignment statement is allowed, and everybody is happy!

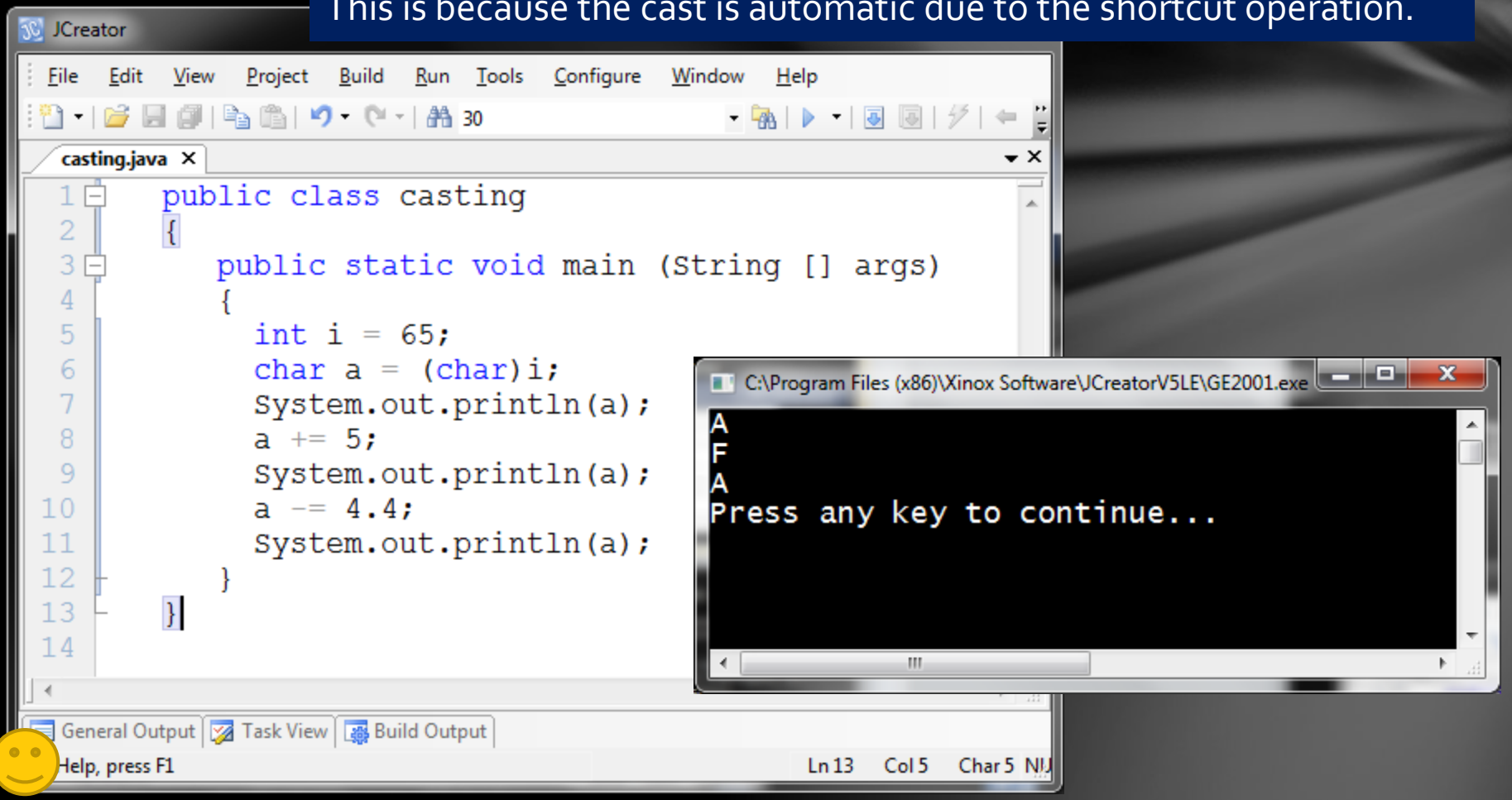


Automatic Type Casting

- Earlier we talked about the shortcut operators, +=, -=, and *=.
- One interesting feature of these shortcuts is automatic casting.
- See some examples of this on the next slide.

Automatic Type Casting

Here you can see that the char variable easily handles increments and decrements by an int and by a double, without an explicit cast. This is because the cast is automatic due to the shortcut operation.



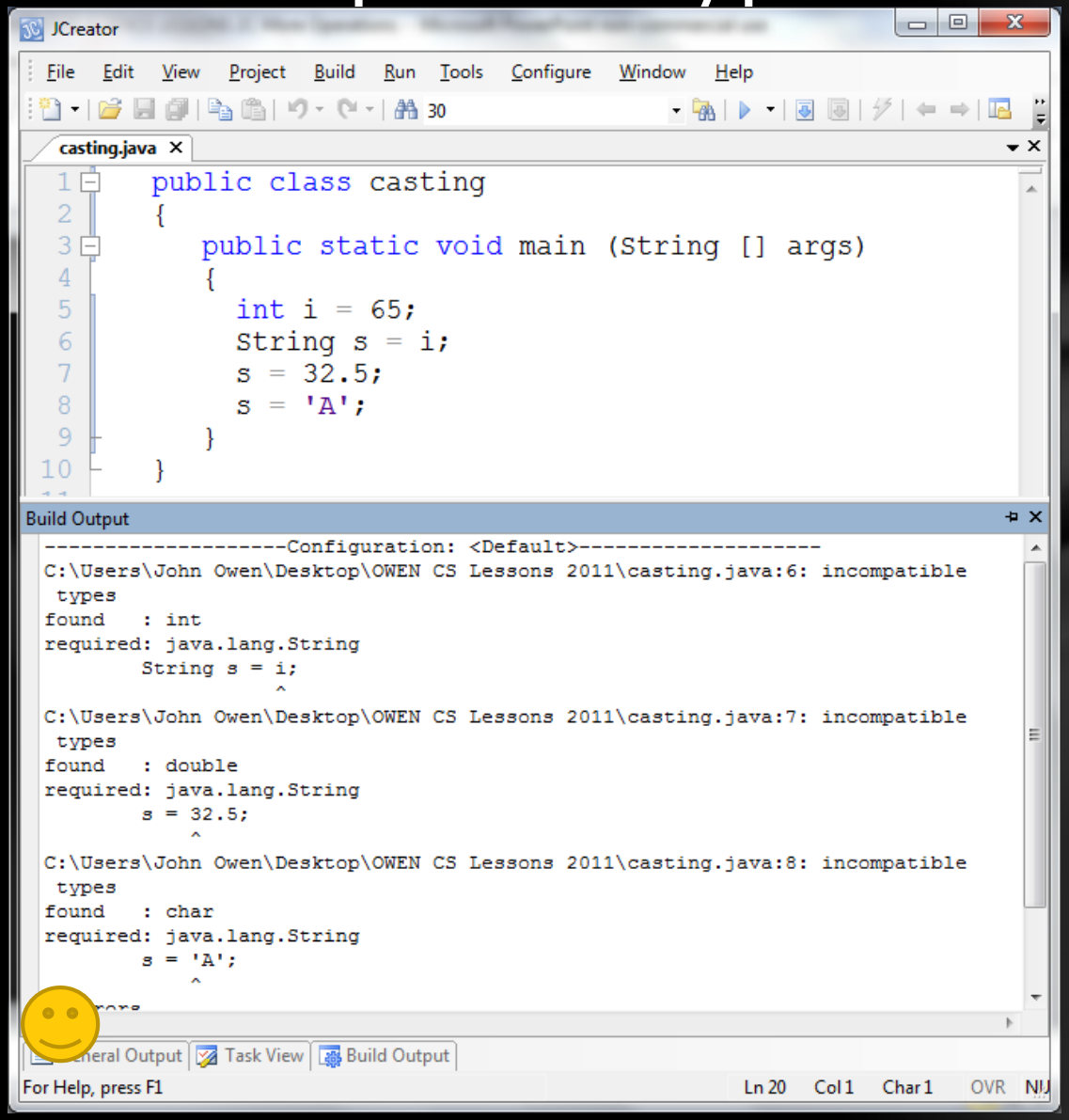
String Conversion

- Casting does not work with the String object, either converting to a String, or from a String.
- Other processes must be used, such as:
  - empty string concatenation
  - charAt
  - parsing

Empty string concatenation

- Since normal casting will not work with Strings, empty string concatenation must be used in order to accomplish this.
- See some examples on the next slide.

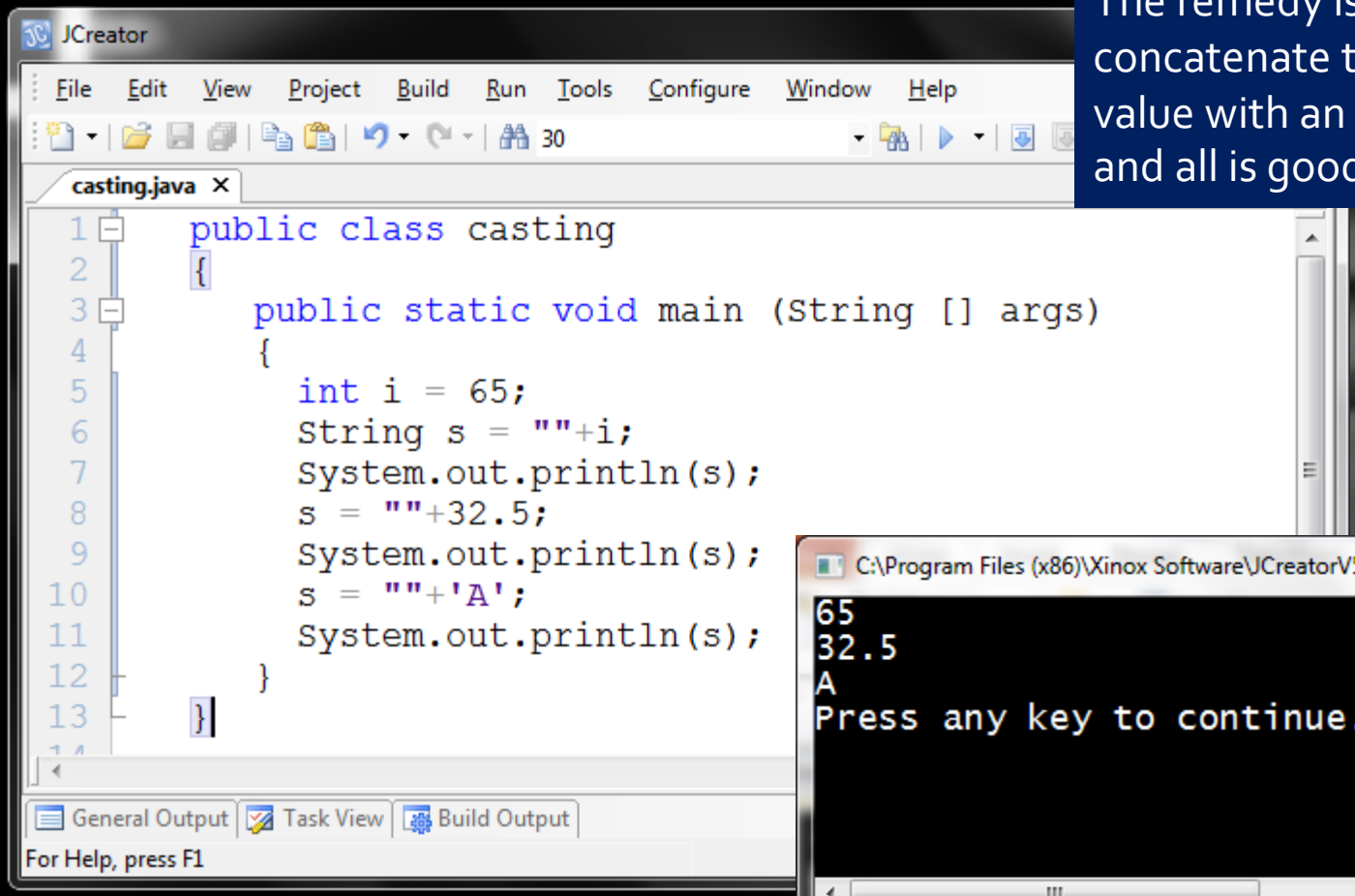
"Incompatible types"



Here you can see several errors caused by an attempt to assign a non-String value to a String variable, with the error message, "incompatible types", instead of "possible loss of precision".

“Incompatible types” remedy

The remedy is easy: concatenate the incompatible value with an empty string, and all is good with the world!

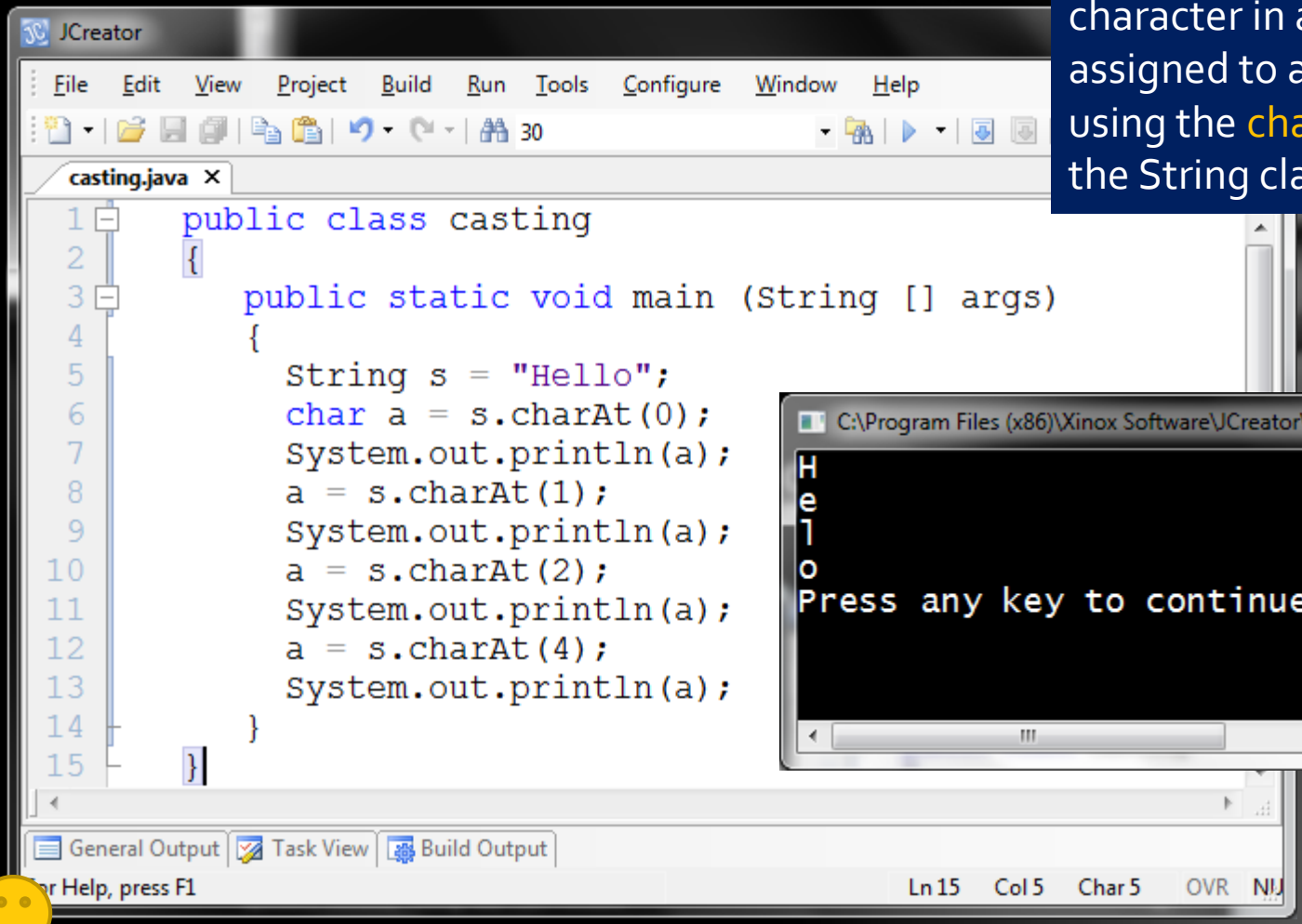


charAt

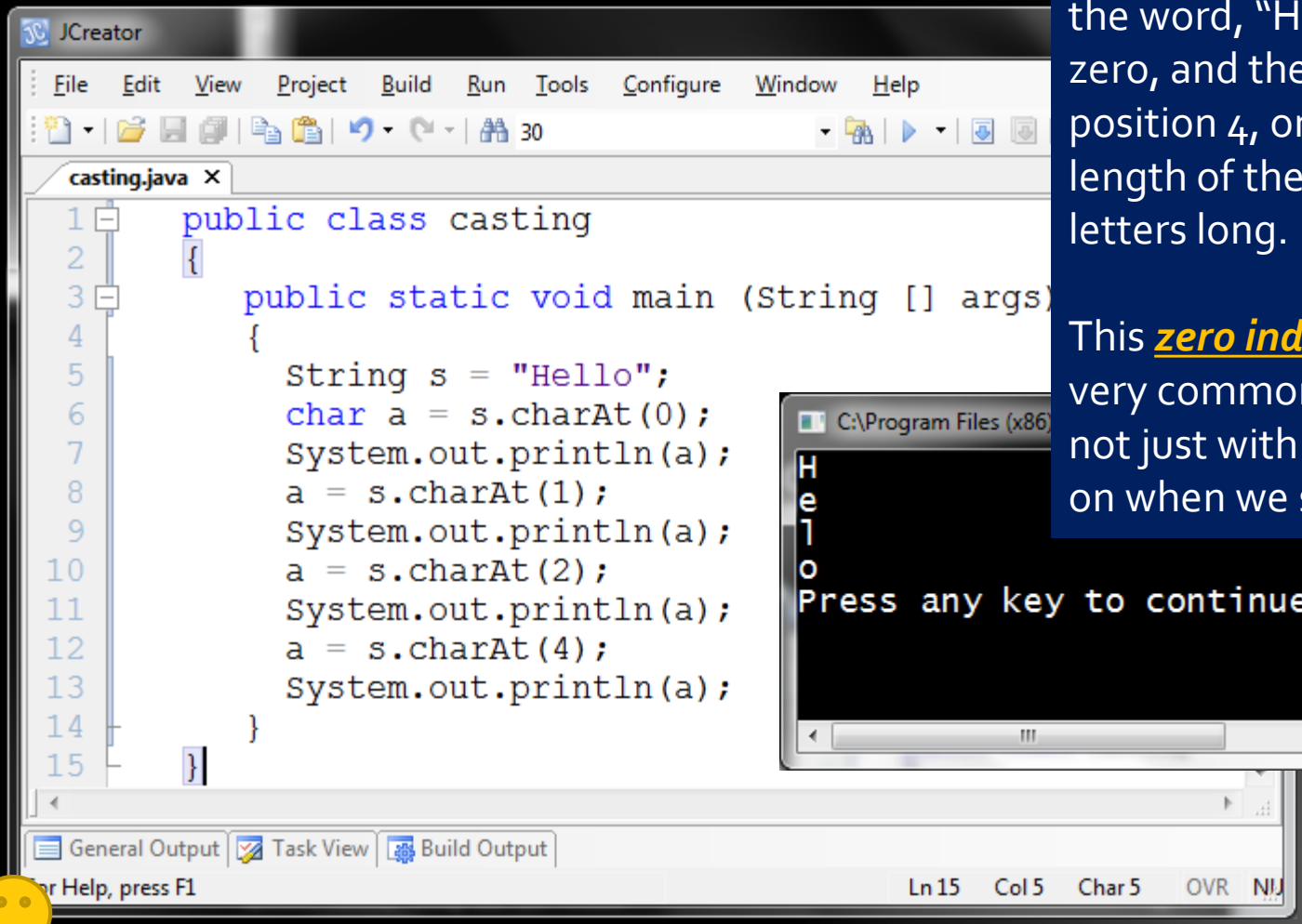
- It is also possible to convert from Strings to characters.
- Since a String is actually an "array" of characters, each digit in a String is a character (more about arrays later).
- The way to extract that character into it's own variable is quite easy…
- `char a = "Hello".charAt(0);`

String to char conversion

Here you see how each character in a String can be assigned to a char variable using the charAt method of the String class.



Zero indexing



Notice that the first letter of the word, "Hello", is at position zero, and the last letter is at position 4, one less than the length of the word, which is 5 letters long.

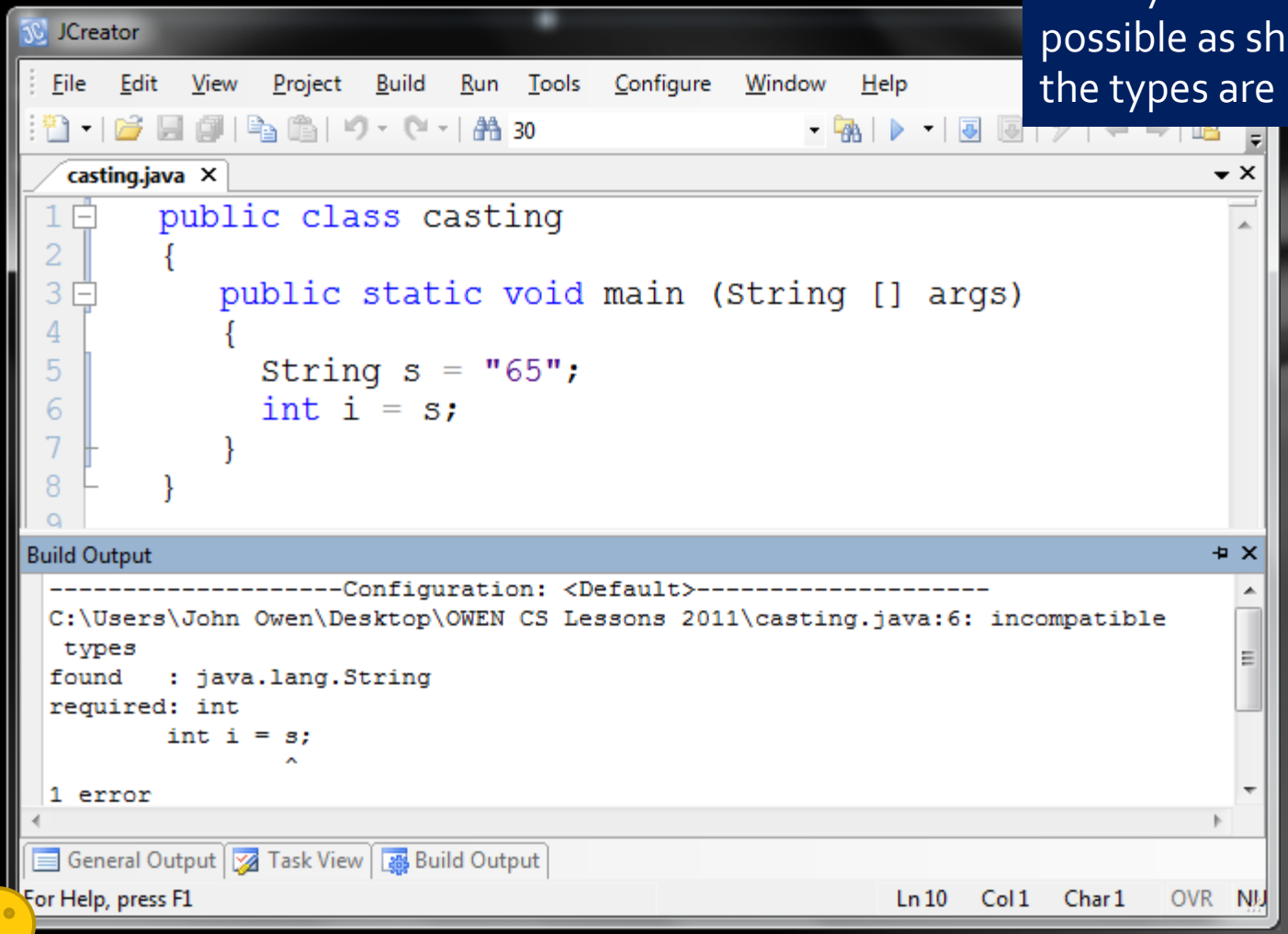
This ***zero indexing*** concept is very common in programming, not just with Strings, but later on when we study arrays.

Parsing

- The conversion of Strings that "look" like integers or decimals is also possible.
- It requires the use of some special Integer and Double class methods, but still is an easy prospect.
- See the next slide for examples.

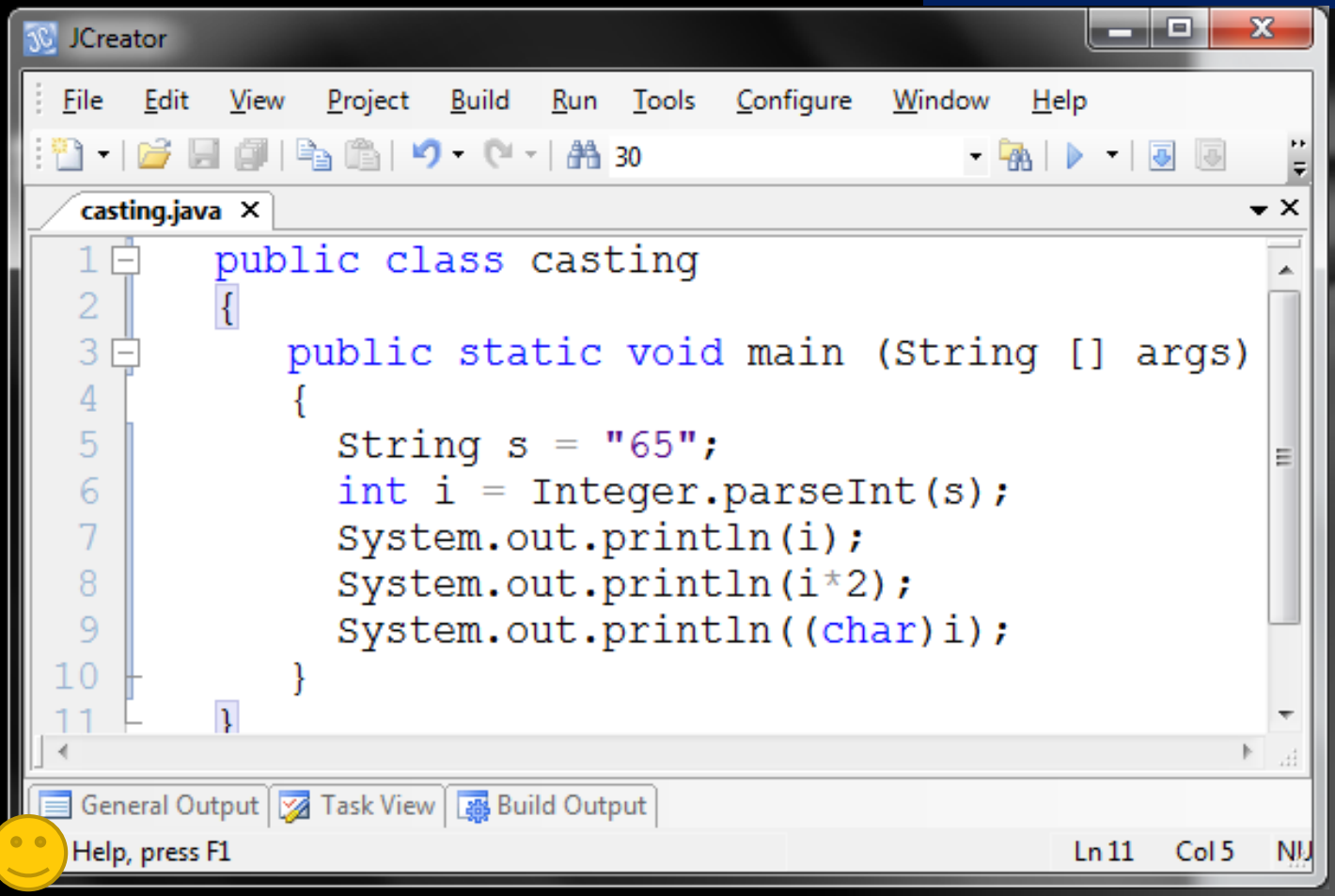
String to integer conversion

Clearly this conversion is not possible as shown below, since the types are incompatible.



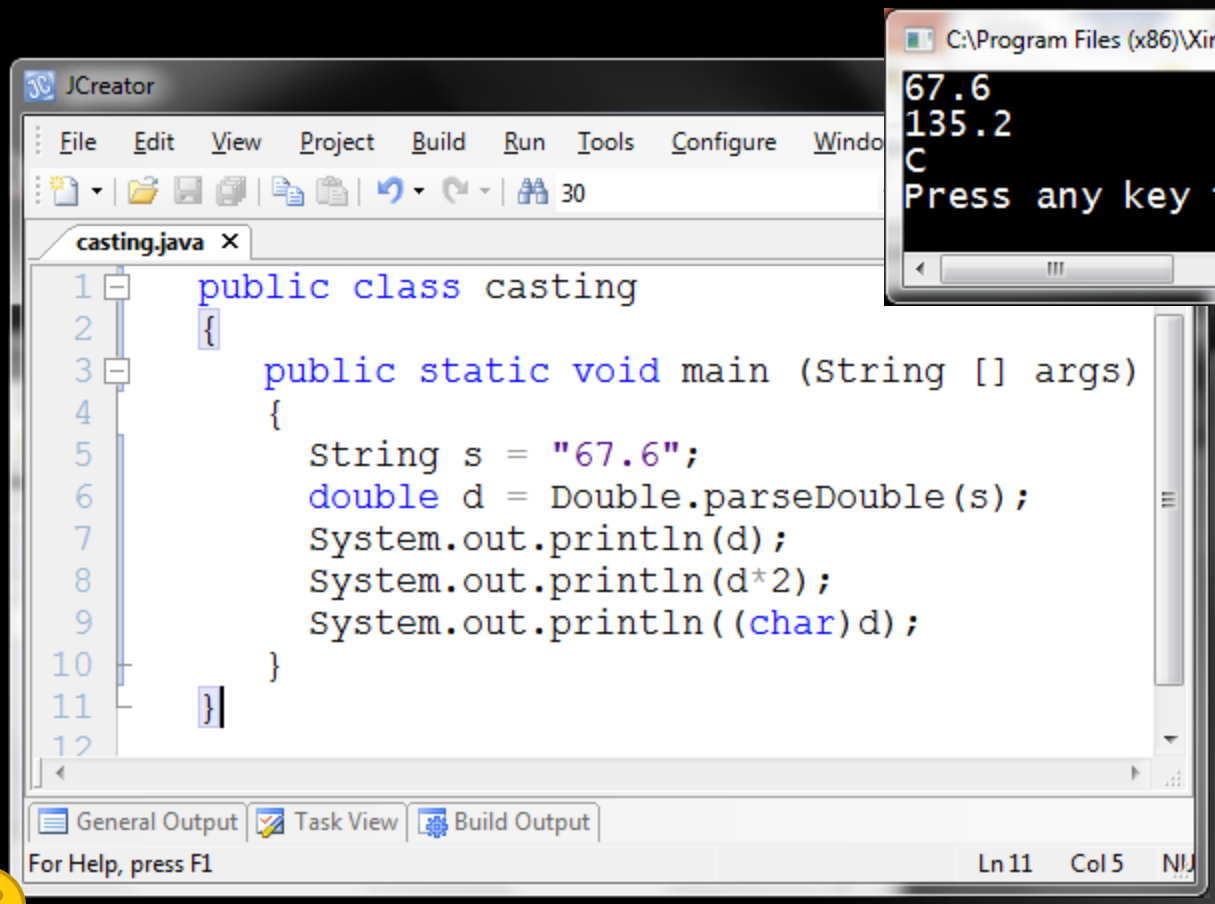
Integer.parseInt

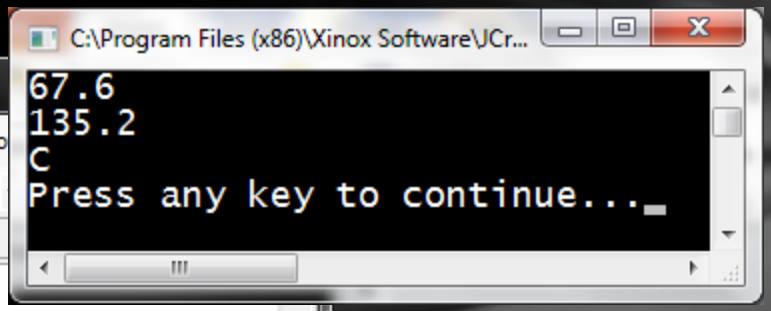
The string "65" is parsed into an integer value by the Integer.parseInt method, as you can see with the example below



Double.parseDouble

The parsing process also works with double values, as shown by the program example below.





Division and Modulus

- Earlier you learned the +, -, and * operators, along with special short cut operators, +=, -=, and *=.
- Now let's discuss the two remaining operators: division (/) and modulus (%)

Division

- The division operator – the "forward" slash (/) – does exactly as you would expect; it divides two numeric values.
- But there's a catch…

Integer Division

- When two integers are involved, such as `12/3`, the answer produced is an integer, or `4` as you would expect.
- The catch is that any remainder is discarded, so `13/3`, would also result in the value `4`, as would `14/3`.
- `15/3`, `16/3`, and `17/3` would all result in `5`.

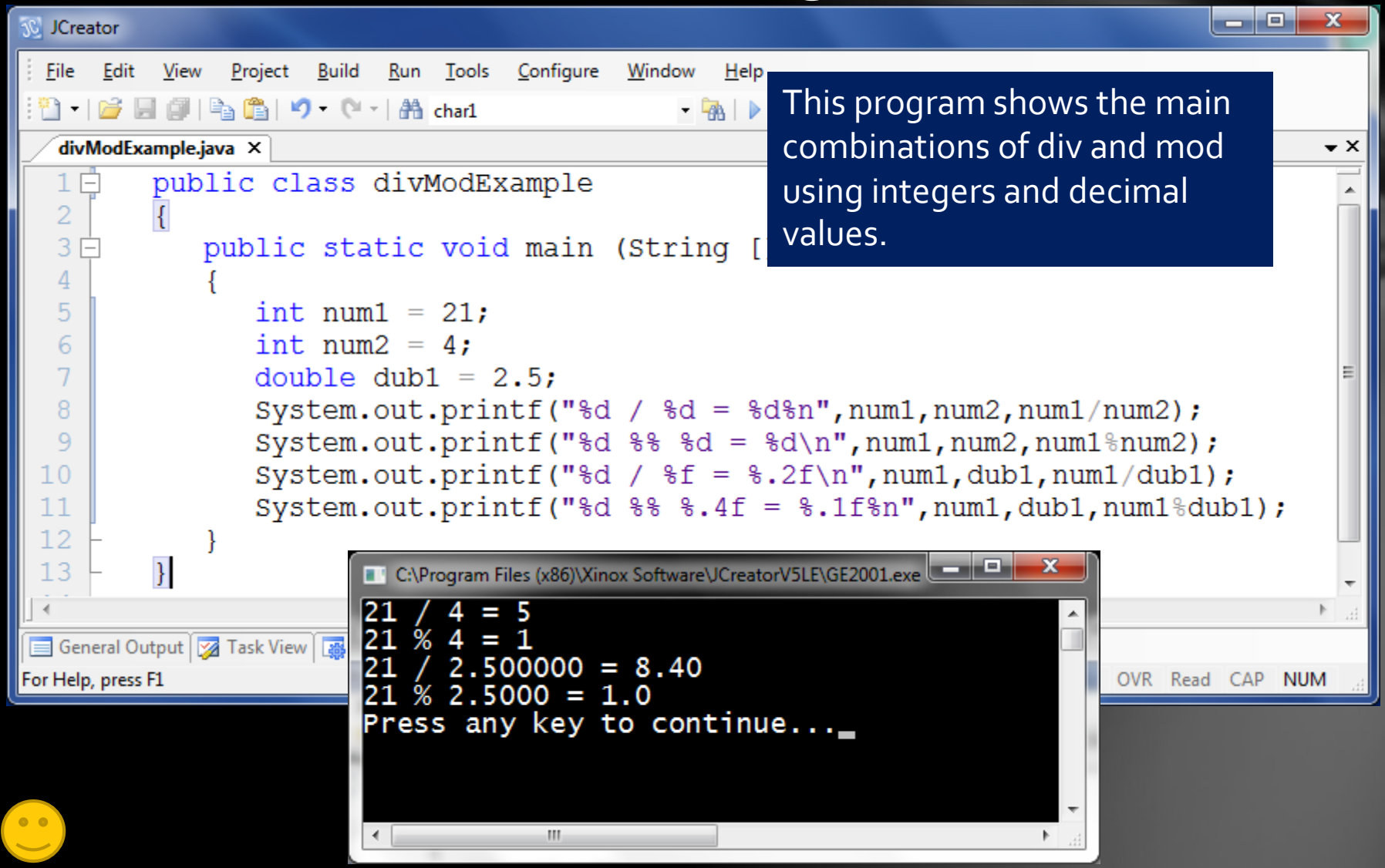
Decimal Division

- When ***<u>at least one</u>*** of the two values is a decimal, such as `12.0/3`, `12/3.0`, or `12.0/3.0`, the answer produced is a decimal, or `4.0`.
- The fractional part is kept as a part of the value.
- `12.0/5`, `12/5.0`, or `12.0/5.0` would all result in `2.4`

Modulus (%)

- The modulus operator, or "mod" for short, is a new one to most of you, but a very important one in computer science.
- Many situations in programming rely on finding the remainder value of a division calculation, which modulus does.

Division and modulus program example



This program shows the main combinations of div and mod using integers and decimal values.

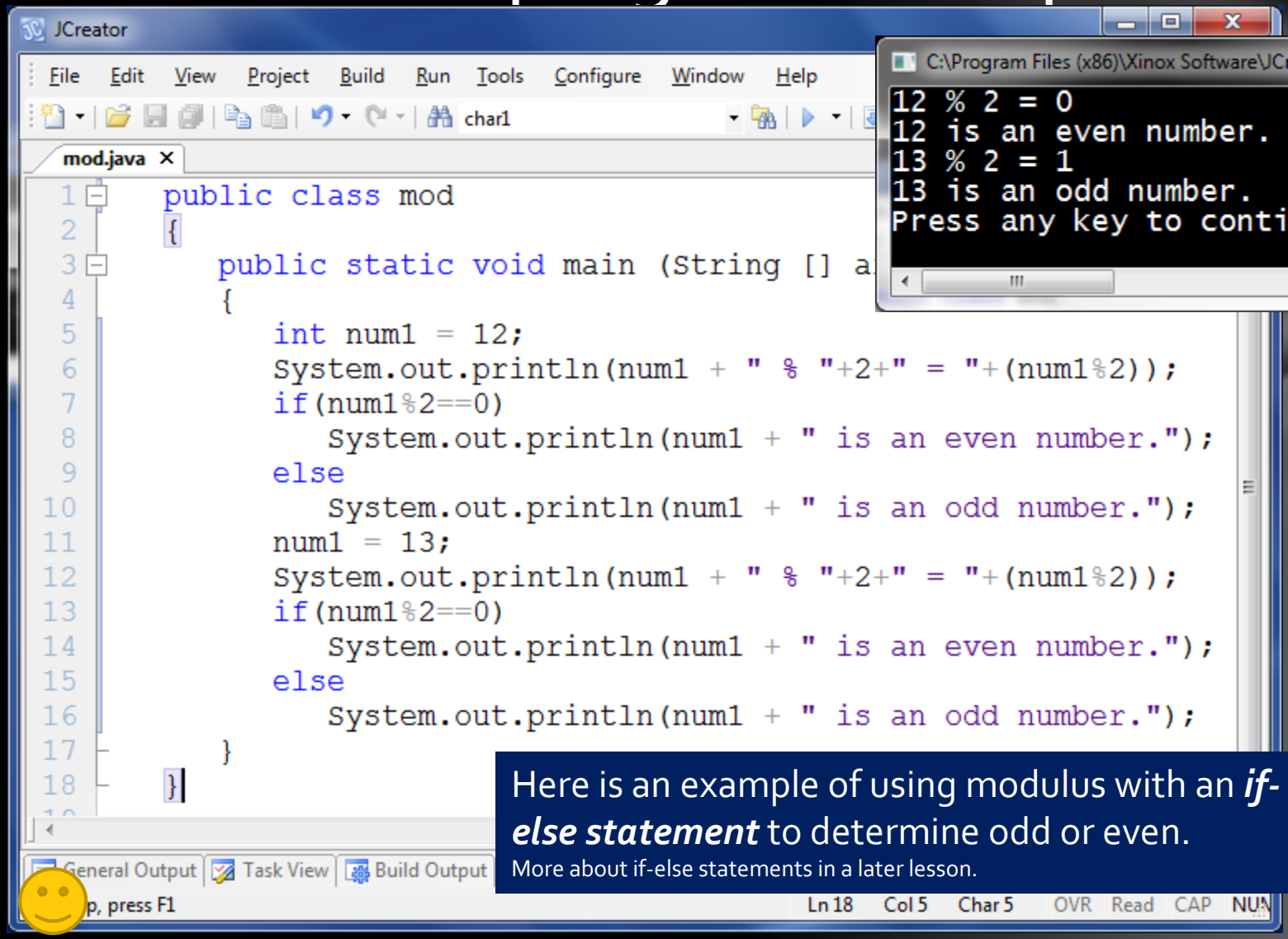
Odd or even

- One very simple example of using modulus is determining if a value is odd or even.
- Trained mathematicians like you know this fact almost intuitively, but the computer needs a modulus calculation to do this.
- If you "mod" `12` by `2`, you get a value of zero for an answer, since the remainder of `12` divided by `2` is zero.
- The zero remainder for "mod by 2" means the value is even.

Odd or even

- If you "mod" `13` by `2`, you get a value of `1` for an answer, since the remainder of `13` divided by `2` is `1`.
- The `1` remainder in this case means the tested value is odd.
- In a later lesson, you will use an *if* statement to do this test, as shown in the program example on the next slide.

Odd or even program example



Here is an example of using modulus with an ***if-else statement*** to determine odd or even.

More about if-else statements in a later lesson.

Other div/mod examples

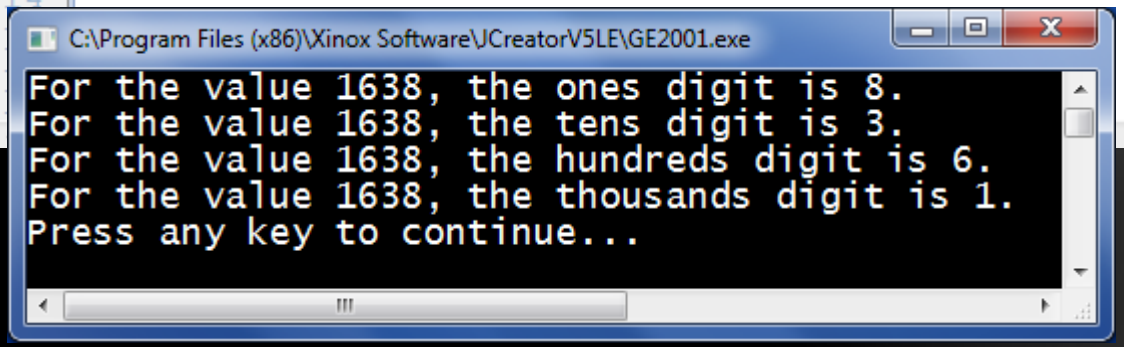
- Two more classic examples of using div and mod in combination are shown on the next few slides.
  - Isolating place values
  - Making change

Isolating place values

divmodCombo.java | Math.html

```
1   public class divmodCombo
2   {
3       public static void main
4       {
5           int num = 1638;
6           System.out.printf("For the value %d, the ones digit is %d.\n",
7                             num,num%10);
8           System.out.printf("For the value %d, the tens digit is %d.%n",
9                             num,num%100/10);
10          System.out.printf("For the value %d, the hundreds digit is %d.\n",
11                            num,num%1000/100);
12          System.out.printf("For the value %d, the thousands digit is %d.%n",
13                            num,num/1000);
14
```

Study the expressions carefully for the middle two statements, especially how the combination of mod and div produce the desired result.

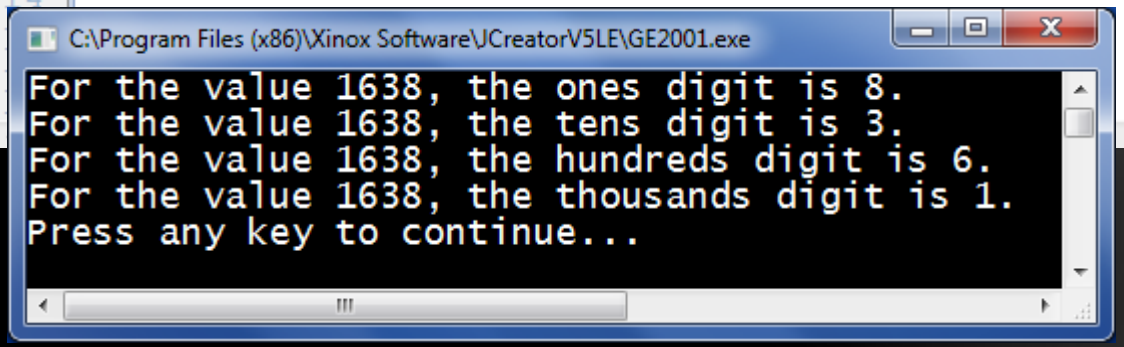


Isolating place values

divmodCombo.java | Math.html

```
1   public class divmodCombo
2   {
3       public static void main
4       {
5           int num = 1638;
6           System.out.printf("For the value %d, the ones digit is %d.\n",
7                             num,num%10);
8           System.out.printf("For the value %d, the tens digit is %d.%n",
9                             num,num%100/10);
10          System.out.printf("For the value %d, the hundreds digit is %d.\n",
11                            num,num%1000/100);
12          System.out.printf("For the value %d, the thousands digit is %d.%n",
13                            num,num/1000);
14
```

In the second output shown below, `1638 % 100` produces `38`, which then divided by `10` produces `3`, the value of the tens digit.



Making change

This program demonstrates how to separate the various money denominations. You will further extend this example in a later lab, calculating the coin denominations as well, i.e., quarters, dimes, nickels and pennies.

divmodCombo.java | Math.html

```
1   public class d
2   {
3      public static void main (String [] args)
4      {
5         int pennies = 3738;
6         System.out.printf("%d pennies converts to:\n",pennies);
7         System.out.printf("\t%d  ten dollar bill(s),%n",
8                           pennies/1000);
9         System.out.printf("\t%d five dollar bill(s),%n",
10                          pennies%1000/500);
11        System.out.printf("\t%d  one dollar bill(s).\n",
12                          pennies%500/100);
13        System.out.println("\tand some change.");
14     }
15  }
```

Math class functions

- The Math class constants PI and E were introduced in an earlier lesson and are used quite often in calculations.
- There are also Math class methods, or functions, that are very useful in mathematic problem solving.
- We will learn two of them in this lesson:
  - pow
  - sqrt

Math.pow

- Since there is no exponent operator in JAVA, we use the ***Math.pow*** function to do this.
- This function requires two parameters, the base and the exponent.
- For the power expression `2^3`, the function call is `Math.pow(2,3)`, resulting in `8.0`
- See some examples on the next slide.

Math.pow

Below are several examples using the Math.pow function, including square root (exponent of `0.5`) and inverse (exponent of `-1`). Notice carefully that the result of the ***pow*** function is ALWAYS a double value.

mathClassFunctions.java

```
1   public class mathClass
2   {
3       public static void
4       {
5           int num1 = 2;
6           int num2 = 4;
7           double dub1= Math.pow(num1,3);
8           System.out.printf("%d ^ %d = %f%n",num1,3,dub1);
9           dub1= Math.pow(num1,num2);
10          System.out.printf("%d ^ %d = %.2f%n",num1,num2,dub1);
11          dub1= Math.pow(2.5,num1);
12          System.out.printf("%.3f ^ %d = %.4f%n",2.5,num1,dub1);
13          dub1= Math.pow(100,0.5);
14          System.out.printf("The square root of %d is %.1f%n",100,dub1);
15          dub1= Math.pow(10,-1);
16          System.out.printf("The inverse of %d is %.1f%n",10,dub1);
17      }
18  }
```

```
C:\Program Files (x86)\Xinox Software\JCreato...
2 ^ 3 = 8.000000
2 ^ 4 = 16.00
2.500 ^ 2 = 6.2500
The square root of 100 is 10.0
The inverse of 10 is 0.1
Press any key to continue...
```

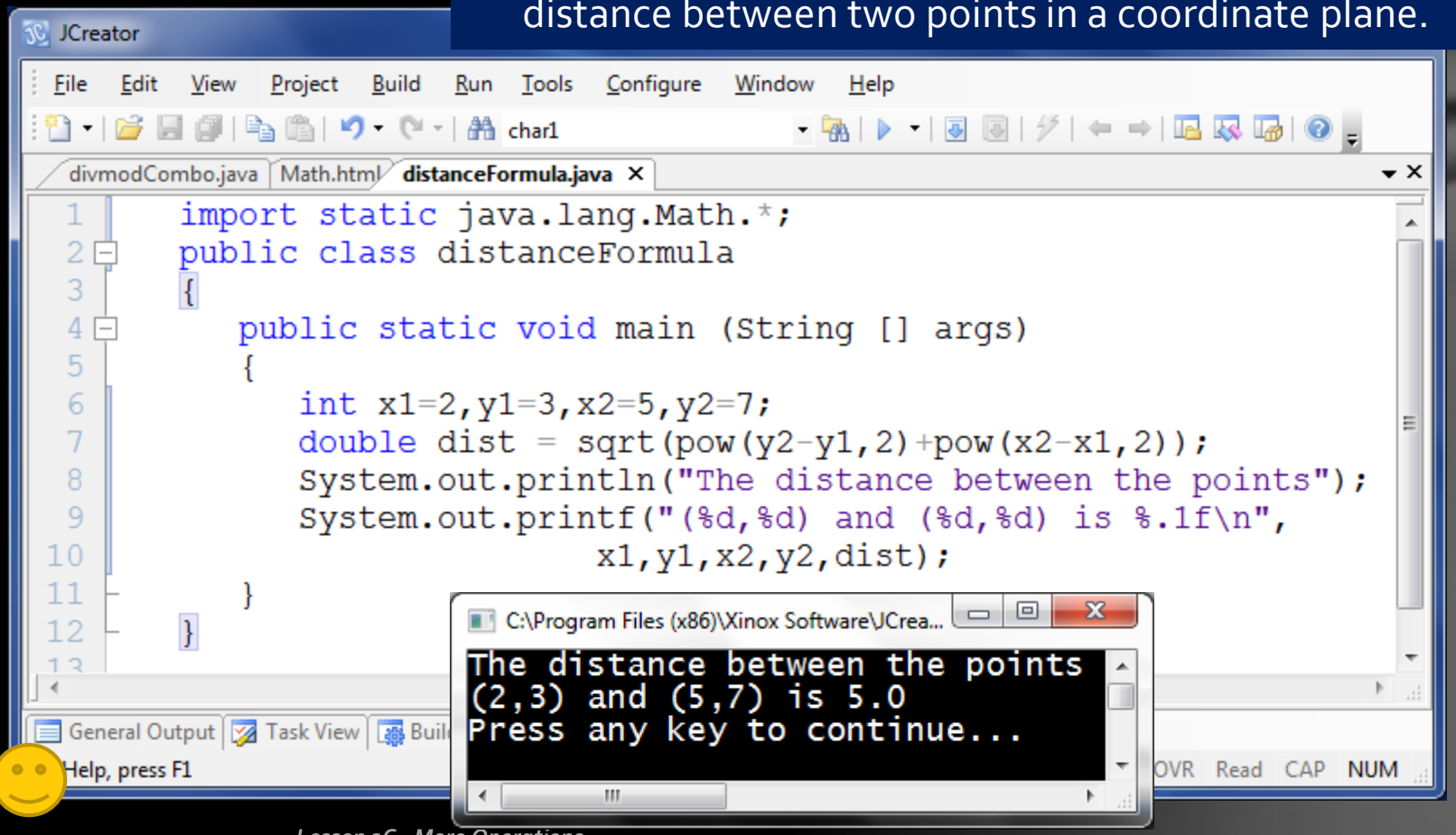
Math.sqrt

- Another very useful Math class method calculates square root.
- On the next slide you will see the distance formula calculation, which uses a combination of sqrt and pow.
- You will also see a new static import statement which allows use of the Math class functions WITHOUT having to say "Math" each time.

Math.sqrt and use of static import

Notice two new things here:

- The static import statement at the top
- The combination use of sqrt and pow to calculate the distance between two points in a coordinate plane.



Math class functions

There several more Math class functions that will be introduced in a later lesson. These include:

- abs
- ceil
- floor
- sin
- cos
- tan
- and more….

Lesson Summary

- This lesson discussed more JAVA operations regarding the concepts of reassignment, operation shortcuts, type casting, String conversion, division and modulus, and two Math class functions (pow and sqrt).

Lab 2C-1

WAP to assign two integer values to two variables *num1* and *num2*. Then calculate and assign to variables the sum, difference, product, integer quotient, decimal quotient, integer modulus value, and decimal modulus value for the two integers. Indent output 20 spaces and format output exactly as shown below. Make the program run for two sets of values.

Data:

```
12  5
256  17
```

See output on next slide

Lab 2C-1

Output:

```
****.****.****.****.*
                    12 + 5 = 17
                    12 - 5 = 7
                    12 * 5 = 60
                    12 / 5 = 2
                    12 / 5.0 = 2.40
                    12 % 5 = 2
                    12 % 5.0 = 2.00
****.****.****.****.*
                    256 + 17 = 273
                    256 - 17 = 239
                    256 * 17 = 4352
                    256 / 17 = 15
                    256 / 17.0 = 15.06
                    256 % 17 = 1
                    256 % 17.0 = 1.00
```

Lab 2C-2

WAP to output the area of a circle with a radius of 1.7.

- **Use Math.PI and Math.pow in your calculation.**
- **Format each value to 4 decimal places.**
- **The radius should be defined as a double variable.**
- **No literal values should appear in the output statement.**
- **Indent 15 spaces.**
- **Do a second circle with 10.25 as the radius value.**

```
****.****.****.****.****.
               1.7000 * 1.7000 * 3.1416 = 9.0792
               10.2500 * 10.2500 * 3.1416 = 330.0636
```

Lab 2C-3

WAP to calculate the side of a square given the area.

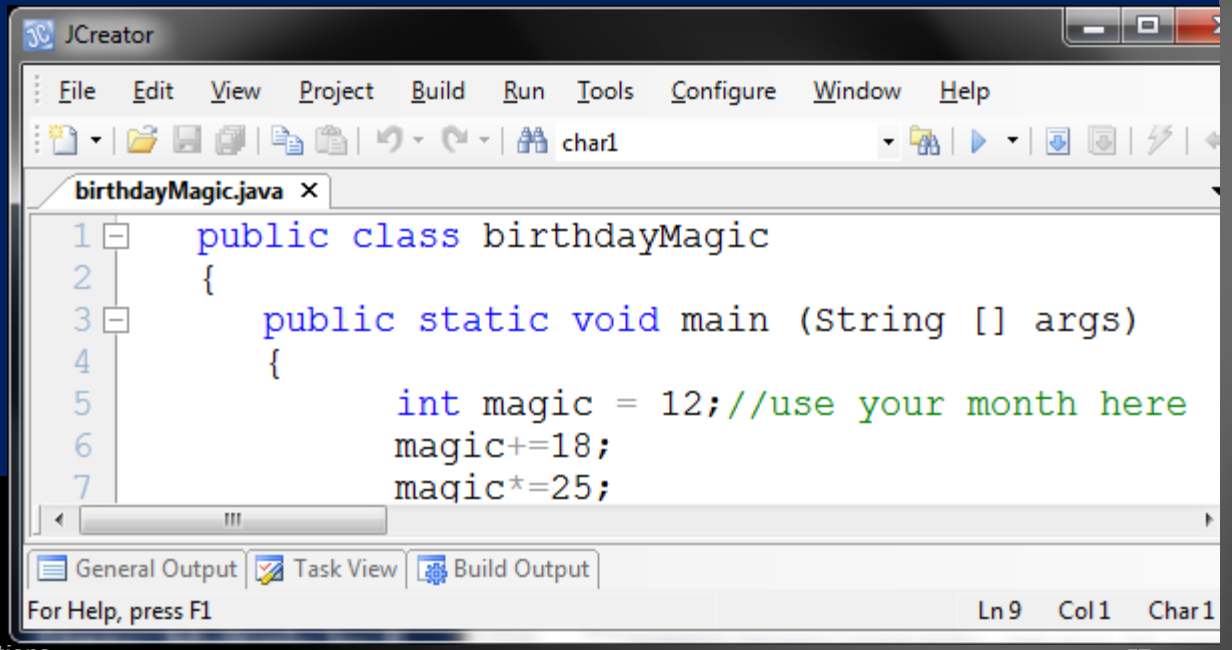
- **Define an integer variable named *area* with a value of 144.**
- **Use the Math.sqrt method in your calculation and assign the answer to a variable called *side*.**
- **Output the answer to two decimal places.**
- **No literal values should actually appear in the output statement, only variables.**
- **Do a second output with the value of *area* as 200.**
- **Indent 5 spaces.**

```
****.*
     The side length of a square with an area of 144 is 12.00
     The side length of a square with an area of 200 is 14.14
```

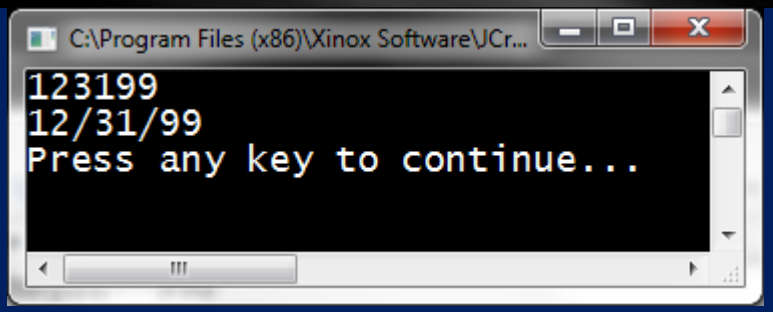
Lab 2C-4

Write a program to do the classic "Birthday Magic" calculation, which correctly calculates your birthday as a large integer value. The steps are listed below. Use the shortcut operations you learned in this lesson to accomplish this task. Below is the first part of the program to help get you started.

Step1: Add 18 to your birth month.
Step2: Multiply by 25.
Step3: Subtract 333.
Step4: Multiply by 8.
step5: Subtract 554.
step6: Divide by 2.



Lab 2C-4



step7: Add your birth date.
step8: Multiply by 5.
step9: Add 692.
step10: Multiply by 20.
step11: Add only the last two digits of your birth year.
step12: Subtract 32940 to get your birthday!

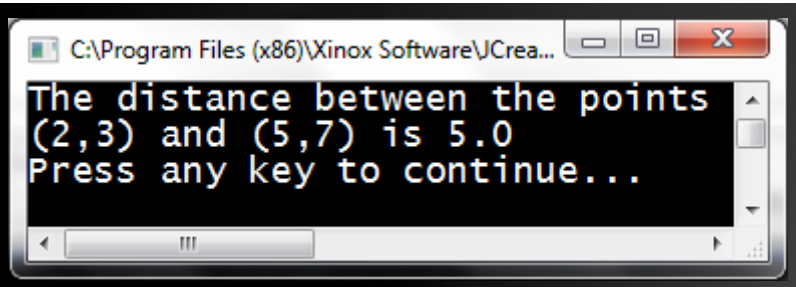
Output the actual value calculated, and then use div and mod to output it in "slash" format, as shown above.

Lab 2C-5

Implement the distance formula program example shown earlier in the lesson on page 49 (portion shown below). Use the points (2,3) and (5,7) as shown in the example, then choose two points of your own that are in opposite quadrants of the coordinate plane, either I and III, or II an IV.

divmodCombo.java | Math.html | distanceFormula.java

```
1   import static java.lang.Math.*;
2   public class distanceFormula
3   {
4      public static void main (String [] args)
5      {
6         int x1=2,y1=3,x2=5,y2=7;
7         double dist = sqrt(pow(y2-y1,2)+pow(x2-x1,2));
```



Lab 2C-6

WAP to calculate your NAAC, that is, your "name-age average character", the ASCII character that represents the value of the average of the ASCII values of all the letters of your first name, plus your age. Here are the steps you must use:

- **Assign your name to a String**
- **Assign your age to a String (not an int)**
- **Use the charAt String method to assign each letter of your name to a char variable.**
- **Parse your age into an int variable using the Integer.parseInt method**

Lab 2C-6

```
C:\Program Files (x86)\Xinox Software\JCreatorV5LE\GE2001.exe
Neo = N + e + o
Age = 17
Neo+17 = 78 + 101 + 111 + 17
Total = 307
Average = 76.8
"Neo"s NAAC is "L", the ASCII value 76
Press any key to continue...
```

NAAC – continued…

- **Total all the ASCII values of your name letters, add your age, and divide to find the decimal average (not integer).**
- **Finally, convert that decimal average into your NAAC, or your "name-age average character".**
- **Output all the values in the format shown above, the complete output for "Neo", age 17.** Your output will be different, of course, unless your name is Neo and your age is 17!
- **Below is the program started for you, showing the code for the first two lines of output.**

```
{
    public static void main (String [] args)
    {
        String name = "Neo";
        String age = "17";
        char c1 = name.charAt(0);
        char c2 = name.charAt(1);
        char c3 = name.charAt(2);
        System.out.printf("%s = %s + %s + %s\n",name,c1,c2,c3);
        System.out.printf("Age = %s\n",age);
```

CONGRATULATIONS!

- You now have a fairly complete understanding of the operations used in JAVA, with only a few subtle nuances yet to discuss.
- *Lesson 3A will introduce keyboard input, allowing programs to be interactive and dynamic.*

Thanks, and have fun!

To order supplementary materials for all the lessons in this package, including lesson examples, lab solutions, quizzes, tests, and unit reviews, visit the O(N)CS Lessons website, or contact me at

John B. Owen
captainjbo@gmail.com