CSE 1325

Week of 08/29/2022

Instructor : Donna French

Comments

- Comments are used to provide information to the human reader of your programs.
- The compiler does not process comments at all.
- Use / / to comment an entire line
- Use /* */ to comment blocks of code.

Converting Floating Point Numbers to Integer

If you try to assign a floating point number to an integer, you will get an error

```
public static void main(String[] args)
{
      double x = 1;
      int y = x;
}
```

```
Compiling 1 source file to
C:\Users\Donna\Desktop\UTA\CSE1310\Programs\Arithmetic\build\classes
C:\Users\Donna\Desktop\UTA\CSE1310\Programs\Arithmetic\src\arithmetic
\Arithmetic.java:19: error: incompatible types: possible lossy
conversion from double to int
          int y = x;
1 error
BUILD FAILED (total time: 1 second)
```

Converting Floating Point Numbers to Integer

```
public static void main(String[] args)
{
    double x = 1;
    int y = x;
}
```

The compiler disallows this assignment because it is potentially dangerous

- The fractional part is lost
- The magnitude may be too large (the largest integer is about 2 billion but a floating point number can be much larger)

Converting Floating Point Numbers to Integer

The cast operator is used to convert a floating point value to an integer.

Write the cast operator before the expression that you want to convert and enclose the cast in `()`

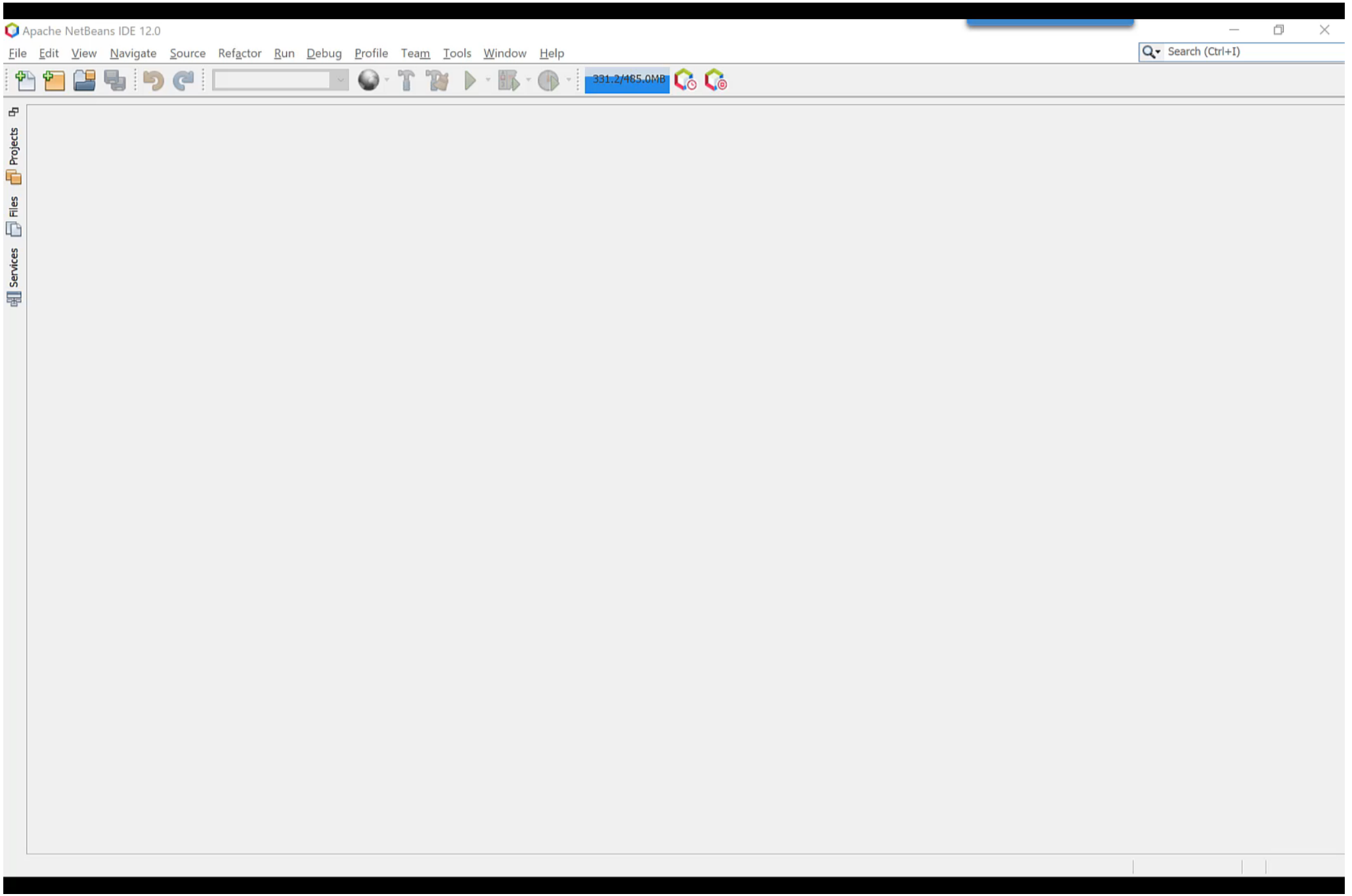
```
public static void main(String[] args)
{
      double x = 1;
      int y = (int)x;
}
```

Common coding error

`int y = int (x);`

placing the `()` around the variable instead of the cast

This placement of the `()` turns the statement into a method call


Apache NetBeans IDE 12.0
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
331.2/465.0MB
Projects
Files
Services

Converting Floating Point Numbers to Integer

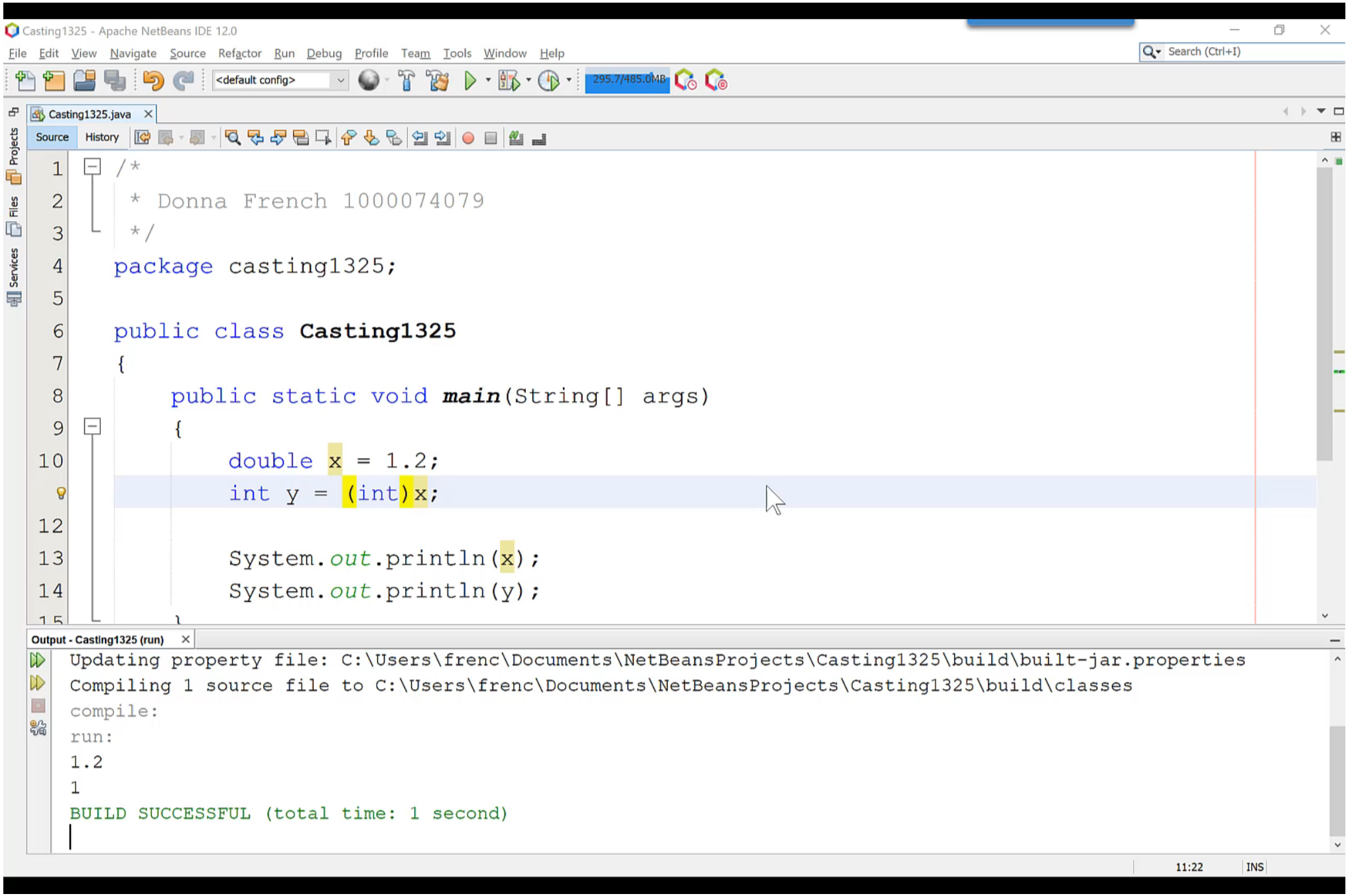
The cast operator discards the fractional part of the floating point value.

This may not be the desired outcome.

If you actually need to round the number (rather than discard), then use the math method `round`.

Please note that the method `round` follows the standard rule of rounding down from anything below 5 and rounding up for anything 5 and above.

1.4999 rounds down to 1 and 1.5 rounds up to 2


Casting1325 - Apache NetBeans IDE 12.0
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
295.7/485.0MB
Casting1325.java
Source History
Projects
Files
Services
1 /*
2 * Donna French 1000074079
3 */
4 package casting1325;
5
6 public class Casting1325
7 {
8 public static void main(String[] args)
9 {
10 double x = 1.2;
int y = (int)x;
12
13 System.out.println(x);
14 System.out.println(y);
Output - Casting1325 (run)
Updating property file: C:\Users\frenc\Documents\NetBeansProjects\Casting1325\build\built-jar.properties
Compiling 1 source file to C:\Users\frenc\Documents\NetBeansProjects\Casting1325\build\classes
compile:
run:
1.2
1
BUILD SUCCESSFUL (total time: 1 second)
11:22
INS

```
 public static void main(String[] args)
 {
    double x = 1.7;

    int y = (int) Math.round(x);

    System.out.println(x);
    System.out.println(y);
}
```

`Math.ceil(x)`

Rounds `x` up to the nearest integer

`Math.floor(x)`

Rounds `x` down to the nearest integer

`Math.round(x)`

`round` follows the standard rule of rounding down from anything below 5 and rounding up for anything 5 and above

It does this by adding 0.5 to the number and then taking the `floor()` of the result.

Input and Output

- Up to now, we have been hardcoding values into our program.

Hard coding

Hard coding is the software development practice of embedding data directly into the source code of a program or other executable object, as opposed to obtaining the data from external sources or generating it at run-time. Wikipedia

- To make our programs more flexible and need fewer modifications, we need our programs to be able to accept input.

Input and Output

Rather than hardcode a value, compile/run the program, modify the program again to change the value, compile/run the program (rinse and repeat)….

We want to **prompt** for the different values of input

```
int Dog = 0;

System.out.print("Please enter a value for Dog ");
```

Input and Output

It would have been nice if the creators of Java had just added

```
System.in.getln(Dog);
```

or at least something that simple.

They did not.

They were so focused on GUIs that they did not pay much attention to keyboard input. `System.in` does exist but has a minimal set of features and must be combined with other classes to be useful.

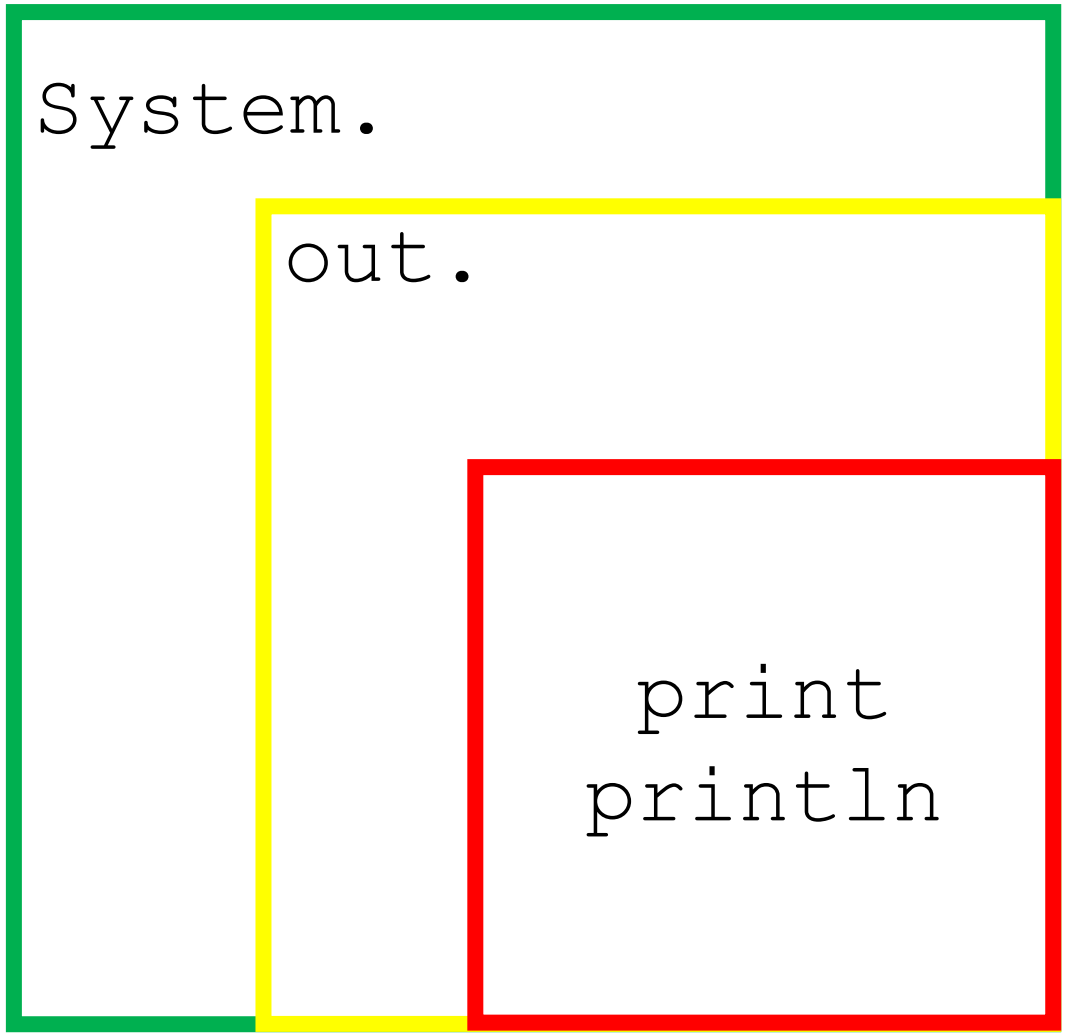
Input and Output

```
import java.util.Scanner;

public class HelloWorld
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
    }
}
```

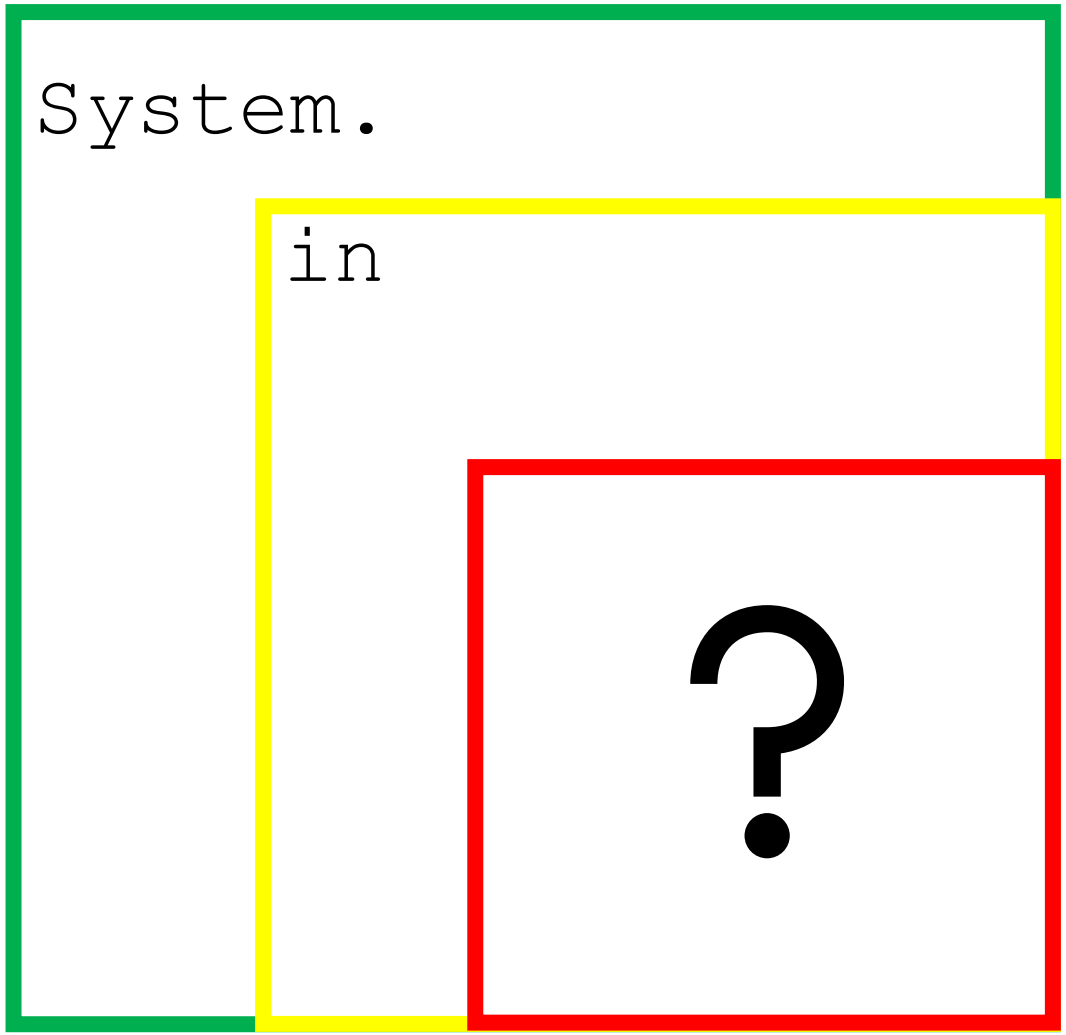
`java.util` is a package
The `Scanner` class belongs to `java.util`
Importing it pulls its code into your code

Instantiates an object called `in` of class type `Scanner`

System.out.print



System.in



`System.out` vs `System.in`

`System.out` uses the standard output stream

- Your screen
- Already open and ready to accept output data
- Just need to know how to format the print…

`System.in` uses the standard input stream

- Your keyboard
- Already open and ready to accept input data
- Lots of different types of input – how to tell them apart?

  If I type "123", did I mean the number 123 or the string 123?

Scanner

```
Scanner in = new Scanner(System.in);
```

This creates a new object called `in`.

This object has methods that allow us to take in different types of input.

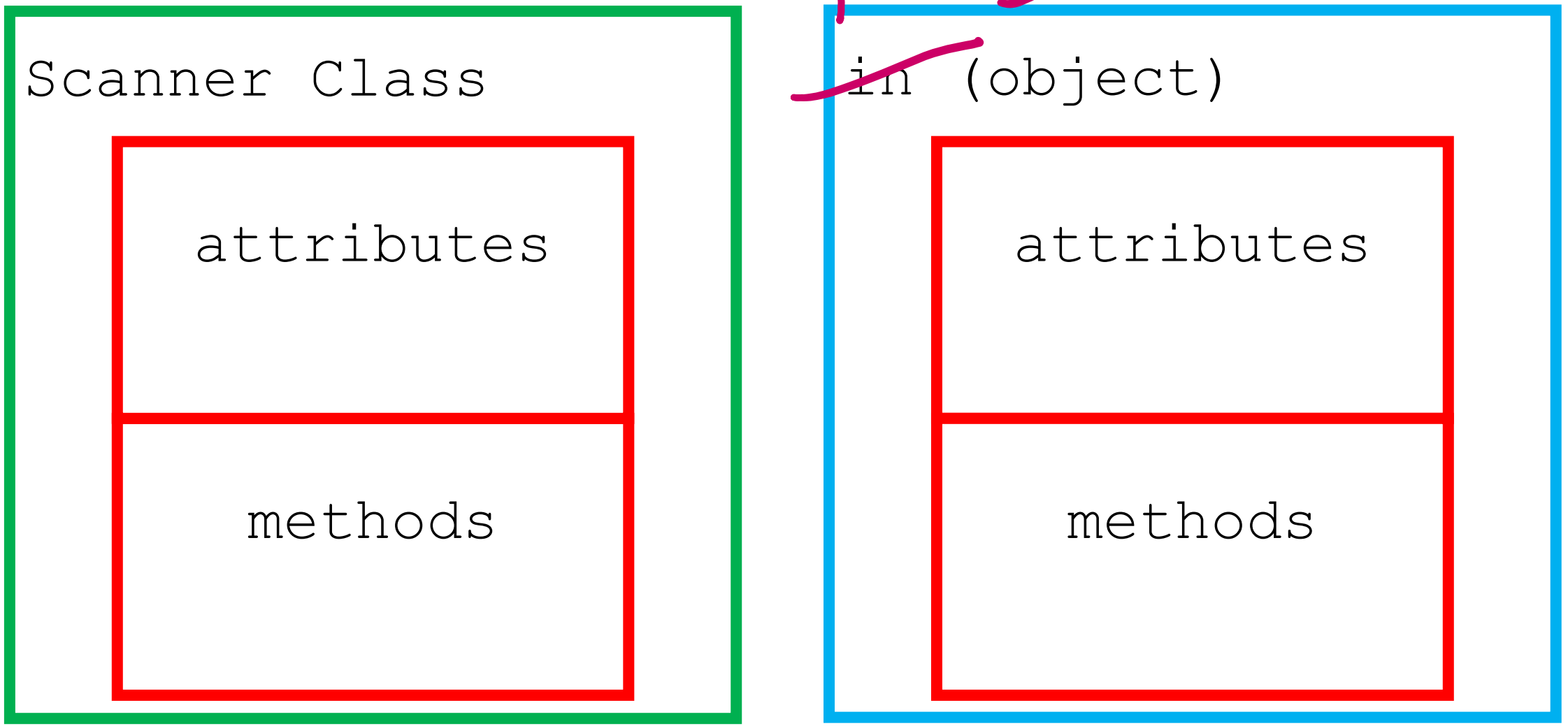
Not required to use the object name of `in`.

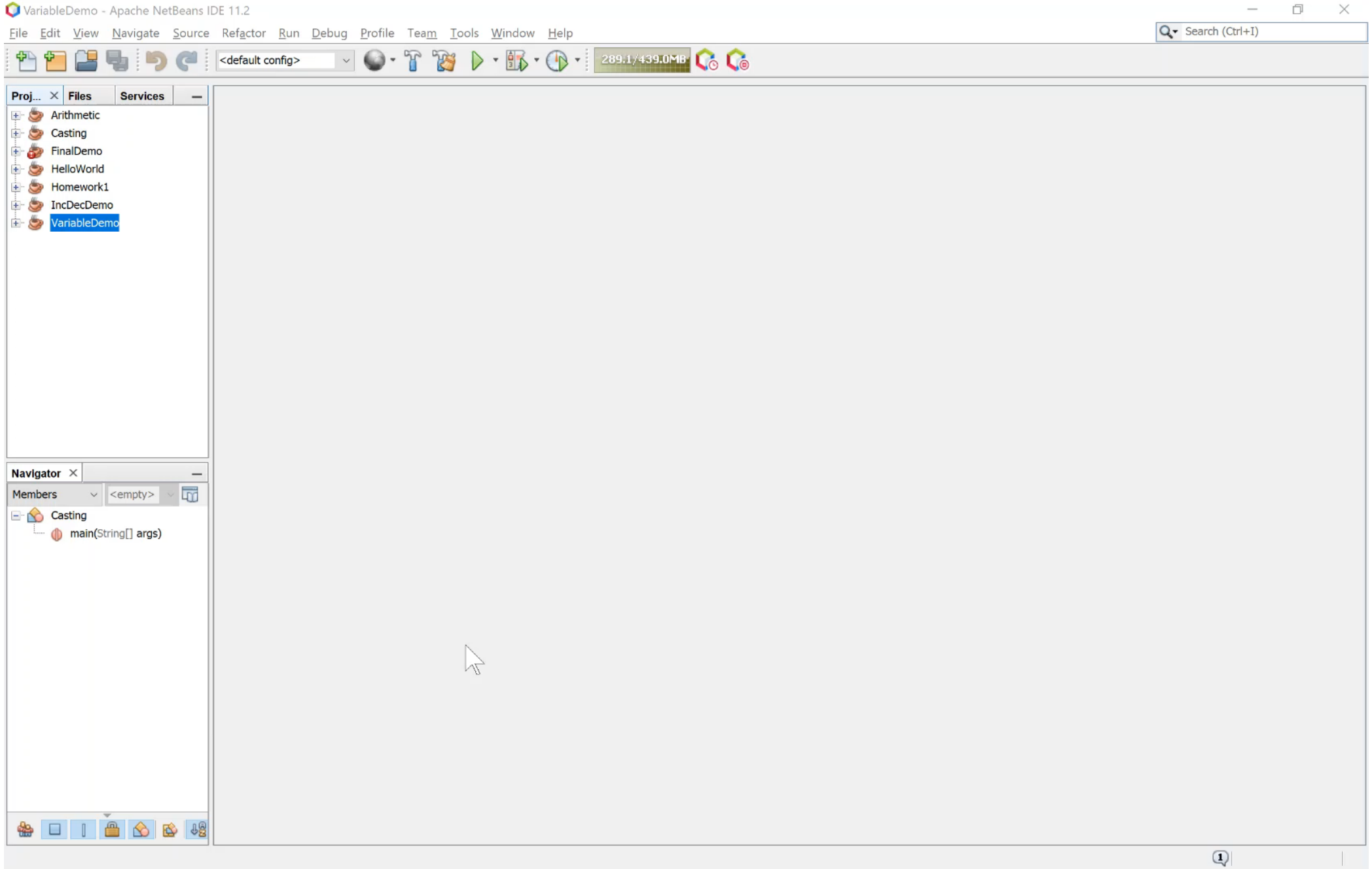
```
Scanner Frog = new Scanner(System.in);
```

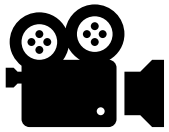
```
Scanner in = new Scanner(System.in);
```

Frog

Frog




VariableDemo - Apache NetBeans IDE 11.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
289.1/439.0MB
Proj... Files Services
Arithmetic
Casting
FinalDemo
HelloWorld
Homework1
IncDecDemo
VariableDemo
Navigator
Members
<empty>
Casting
main(String[] args)



Arithmetic.java | HelloWorld.java | VariableDemo.java | InputOutput.java

Source | History

```
5   package inputoutput;
6   import java.util.Scanner;
7
8   public class InputOutput
9   {
10      public static void main(String[] args)
11      {
12      }
13
14  }
15
```

inputoutput.InputOutput > main

Output - InputOutput (run)

```
import java.util.Scanner;

public class InputOutput
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        int Dog = 0;
        System.out.print("Enter a value for Dog ");
        Dog = in.nextInt();
        System.out.println("The value of Dog is " + Dog);
    }
}
```

Formatting and Alignment

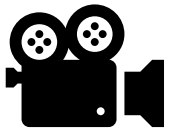
```
package strings;
import java.util.Scanner;
public class Strings
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        do
        {
            if (last >= 0)
            {
            }
            else
            {
            }
        }
        while (first >= 0);
    }
}
```

Input and Output

When the `nextInt` method is called, the program waits until the user types in a **number** and presses ENTER.

After the user supplies the input, the number is placed into the variable and the program continues.

What happens if you enter something other than a number (not an integer)?



Source History

```
5   package inputoutput;
6   import java.util.Scanner;
7
8   public class InputOutput
9   {
10      public static void main(String[] args)
11      {
12          Scanner in = new Scanner(System.in);
            int Dog = 0;
14          System.out.print("Enter a value for Dog ");
15          Dog = in.nextInt();
16          System.out.println("The value of Dog is " + Dog);
17      }
18
19  }
```

inputoutput.InputOutput > main > in

Output - InputOutput (run)

Input and Outp

```
int Dog = 0;
Dog = in.nextInt
```

If the user e ther than an integer at a `nextInt()` prompt, then an exception e the program crashes.

```
Exception in thread "main" java.util.InputMismatchExcepti
        at java.base/java.util.Scanner.throwFor(Scann
        at java.base/java.util.Scanner.nex Sca
        at java.base/java.util.Scanner.ne                    258)
        at java.base/java.util.Scanner.n                    java:2212)
        at icq2.ICQ2.main(ICQ2.java:15)
C:\Users\frenc\Documents\NetBeansProject Q2\nbproject\build-
impl.xml:1330: The following error occurred while executing this line:
C:\Users\frenc\Documents\NetBeansProjects\ICQ2\nbproject\build-
impl.xml:936: Java returned: 1
BUILD FAILED (total time: 6 seconds)
```

Exception
Program compiles and runs but crashes when executing a step

Line 15 in our program
`Dog = in.nextInt();`

```java
import java.util.Scanner;

public class ICQ2
{
    public static void main(String[] args)
    {
        System.out.println("I'm alive!!");
        System.out.println("I'm going to divide by zero!!");
        int Dog = 0;
        Dog = Dog/0;
        System.out.println("Did I survive??");
```

icq2.ICQ2 > main >

Input and Output

To read floating point numbers (decimals), use the `nextDouble` method.

Change

```
int MyVar = in.nextInt();
```

To

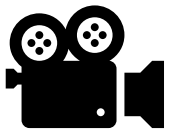
```
double MyVar = in.nextDouble();
```

Source History

```
5   package inputoutput;
6   import java.util.Scanner;
7
8   public class InputOutput
9   {
10      public static void main(String[] args)
11      {
12          Scanner in = new Scanner(System.in);
            int Dog = 0;
14          System.out.print("Enter a value for Dog ");
15          Dog = in.nextInt();
16          System.out.println("The value of Dog is " + Dog);
17      }
18
19  }
```

inputoutput.InputOutput > main > Dog

Output - InputOutput (run)



Formatted Output

We have been using `print` and `println` to print text.

```
System.out.print("Hello");      // Prints "Hello" only
System.out.println("Bye");      // Prints "Bye" and does a carriage return
```

We have also been using `print` and `println` to print variables.

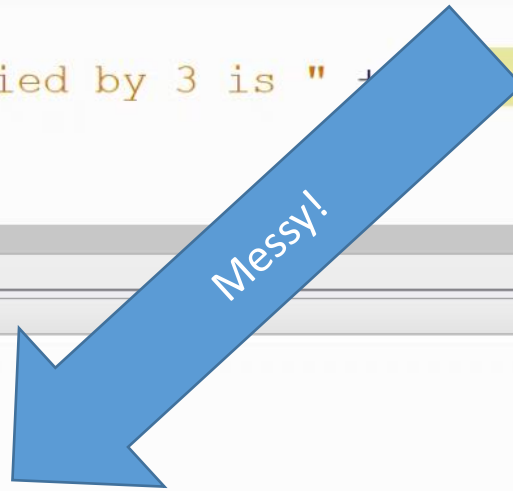
```
double myFraction = 4.3;
System.out.println("4.3 multiplied by 3 is " + myFraction*3);
```

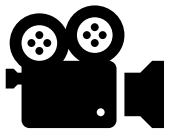
```java
package inputoutput;
import java.util.Scanner;

public class InputOutput
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.print("Hello");      // Prints "Hello" only
        System.out.println("Bye");      // Prints "Bye" and does a carriage re

        double myFraction = 4.3;
        System.out.print("4.3 multiplied by 3 is " + myFraction*3);

    }
```

inputoutput.InputOutput > main

Output - InputOutput (run)

```
compile:
run:
HelloBye
4.3 multiplied by 3 is 12.899999999999999BUILD SUCCESSFUL (total time:
```





Source History

```
5   package inputoutput;
6   import java.util.Scanner;
7
8   public class InputOutput
9   {
10      public static void main(String[] args)
11      {
12          Scanner in = new Scanner(System.in);
13          System.out.print("How much is a 6 pack of Coke? ");
14          double cost = in.nextDouble();
            System.out.print("So each Coke costs " + cost/6);
16
17      }
18
19  }
```

inputoutput.InputOutput > main

Output - InputOutput (run)

Formatted Output

We can control the formatting of output by using format specifiers with the `printf()` method.

We can control the number of digits printed after the decimal by adding a . and an integer for the number of places.

```
System.out.printf("%.2f", price);
```

causes `price` to print as

```
1.22
```

rather than

```
1.215456873
```

Formatted Output

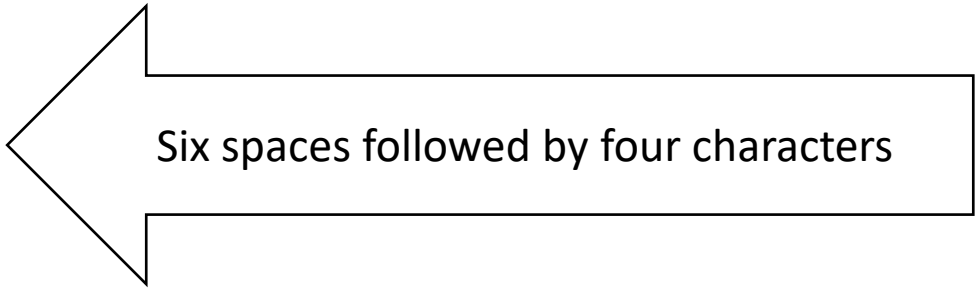
We can control the formatting of output by using format specifiers with the `printf()` method.

We can control the width of the print field

```
System.out.printf("%10.2f", price);
```

causes `price` to print as

```
      1.22
```



rather than

```
1.215456873
```

Formatted Output

Format specifiers with the `printf()` method.

For floating point numbers

```
System.out.printf("%10.2f", price);
```

For integers

```
System.out.printf("%5d", count);
```

For strings

```
System.out.printf("%s", name);
```

Formatted Output

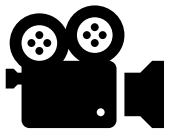
The `printf()` method, like the `print()` method, does not perform a carriage return (start a new line after the output).

If you want to add a carriage return/start the next output on a new line, use the

`%n`

format specifier

```
System.out.printf("%10.2f%n", price);
System.out.printf("%5d%n", count);
System.out.printf("%s%n", name);
```



Source History

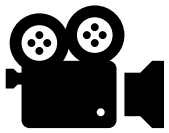
```
8   public class InputOutput
9   {
10      public static void main(String[] args)
11      {
12          Scanner in = new Scanner(System.in);
13          System.out.print("How much is a 6 pack of Coke? ");
14          double cost = in.nextDouble();
15          System.out.println("So each Coke costs " + cost/6);
16      }
17
18  }
19
```

inputoutput.InputOutput > main

Output - InputOutput (run)

```
ant -f C:\\Users\\Donna\\Desktop\\UTA\\CSE1310\\Programs\\InputOutp
```

```
public static void main(String[] args)
{
    Scanner in = new Scanner(System.in);
    System.out.print("How much is a 6 pack of Coke? ");
    double cost = in.nextDouble();
    System.out.printf("So each Coke costs %10.2f%n", cost/6);
}
```



Source History

```
10      public static void main(String[] args)
11      {
12          Scanner in = new Scanner(System.in);
13          System.out.print("Enter first addend ");
14          double addend1 = in.nextDouble();
15          System.out.print("Enter second addend ");
16          double addend2 = in.nextDouble();
17
18          System.out.println(addend1);
19          System.out.println("+" + addend2);
20          System.out.println("=======");
21          System.out.println(addend1+addend2);
22
```

inputoutput.InputOutput › main

Output - InputOutput (run)

```
ant -f C:\\Users\\Donna\\Desktop\\UTA\\CSE1310\\Programs\\InputOut
```

```
System.out.println(addend1);
System.out.println("+" + addend2);
System.out.println("=======");
System.out.println(addend1+addend2);
```

```
12.34
+45.67
=======
58.010000000000005
```

```
System.out.printf("%7.2f%n", addend1);
System.out.printf("+ %5.2f%n", addend2);
System.out.println("=======");
System.out.printf("%7.2f%n", addend1+addend2);
```

```
  12.34
+ 45.67
=======
  58.01
```

Strings

A string is a sequence of characters enclosed in quotes

- "Hello" is a string
- 123 is not a string
- "123" is a string

Java has a data type called `String` that can hold a string

```
String greeting = "Hello";
```

Strings

String literal vs string variable

"Hello"

string literal – character sequence enclosed in quotes

```
String greeting = "Hello";
```

`greeting` is a string variable – a variable that can hold a string

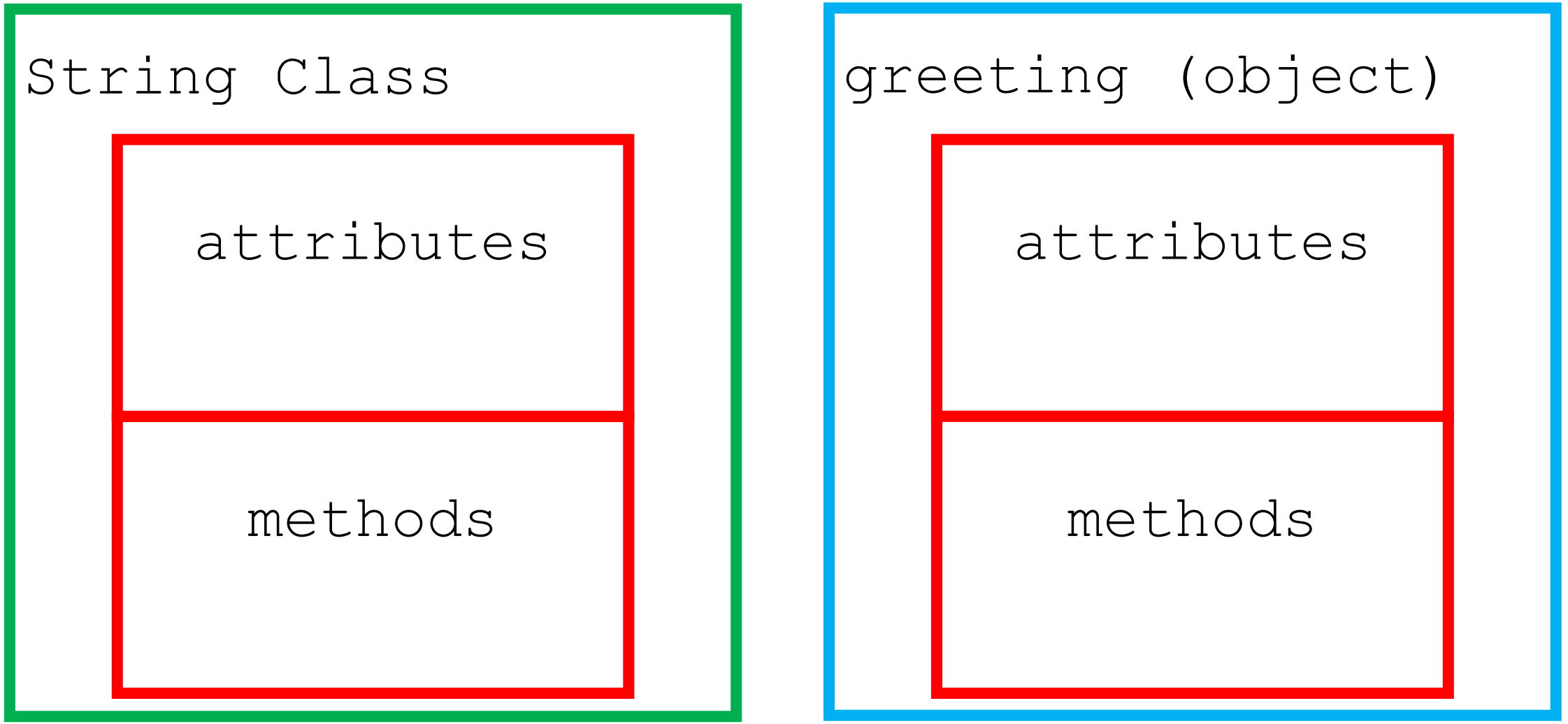
7

is an integer

```
int myNum = 7;
```

`myNum` is an integer variable – a variable that can hold an integer

```
String greeting = "Hello";
```



Primitive Types vs Objects

Primitive

- `byte`
- `short`
- `int`
- `long`
- `float`
- `double`
- `char`
- `boolean`

Classes/Objects

- `String`
  - `length()`
  - `charAt()`
- `Scanner`
  - `nextInt()`
  - `NextDouble()`
- `Math`
  - `abs()`
  - `ceil()`
  - `floor()`

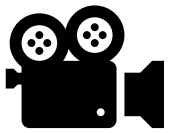
Strings

The number of characters in a string is called the length of the string.

The string literal "Hello" has a string length of 5.

To compute this length in code…

```
String greeting = "Hello";
int length = greeting.length();
```



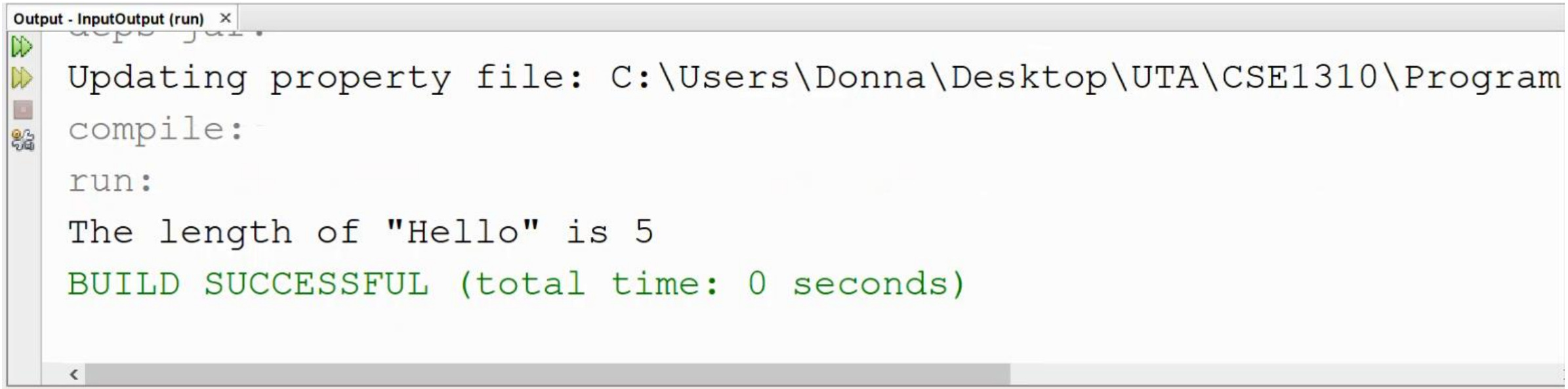
Source History

```
10      public static void main(String[] args)
11      {
12          Scanner in = new Scanner(System.in);
13
14
15
16
17      }
18
19  }
20
```

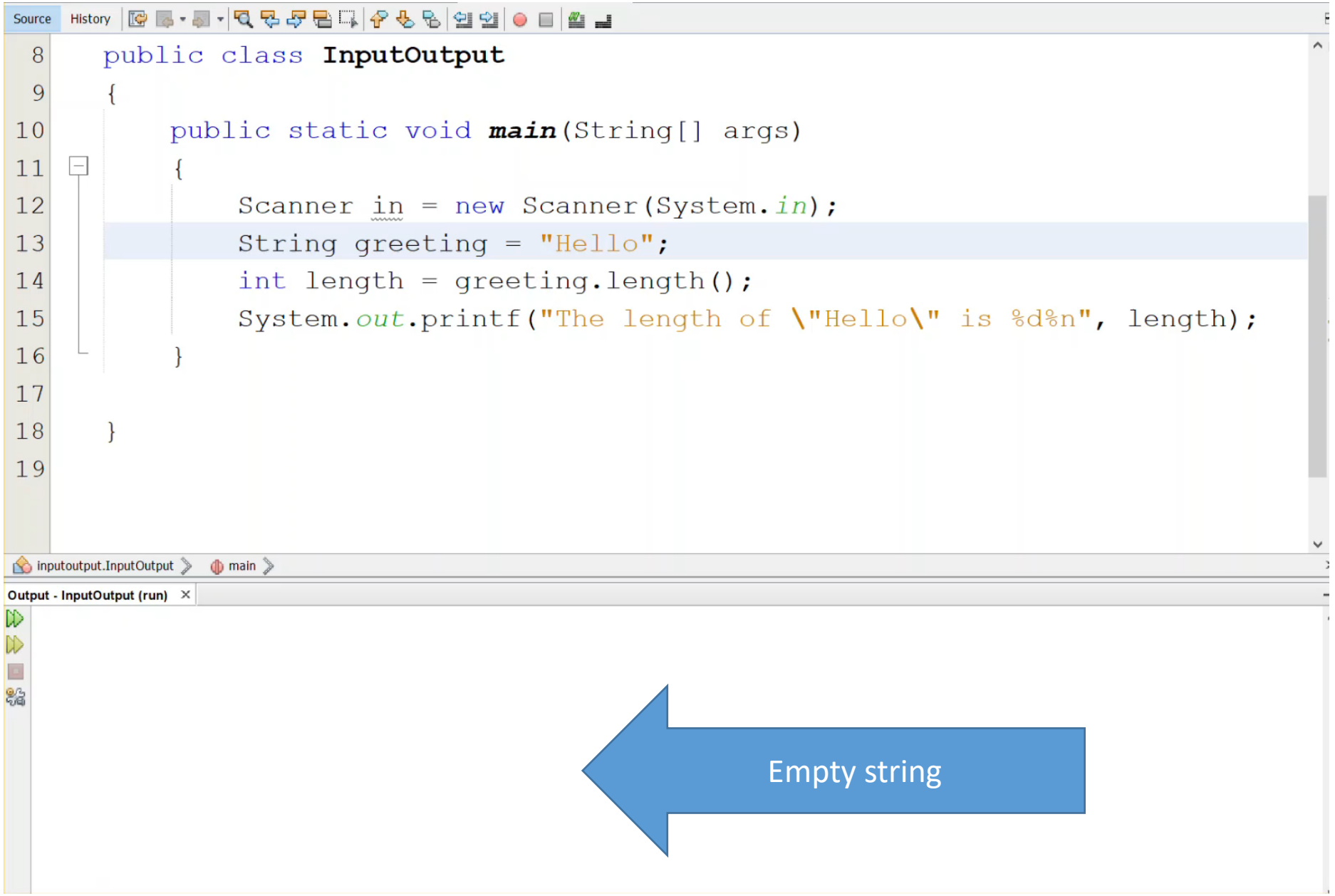
inputoutput.InputOutput > main

Output - InputOutput (run)

```
public static void main(String[] args)
{
    Scanner in = new Scanner(System.in);
    String greeting = "Hello";
    int length = greeting.length();
    System.out.printf("The length of \"Hello\" is %d%n", length);
}
```



What happens if the quotes are empty? ""



Concatenation

Concatenating two strings means joining them together into one string.

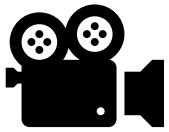
If we have a string

"Butter"

and a string "fly"

then concatenating them will result in

"Butterfly"



Source History

```
8   ublic class InputOutput
9
10      public static void main(String[] args)
11      {
12          Scanner in = new Scanner(System.in);
13
14          String firstName = "Fred"
15      }
16
17
18
```

inputoutput.InputOutput > main > firstName

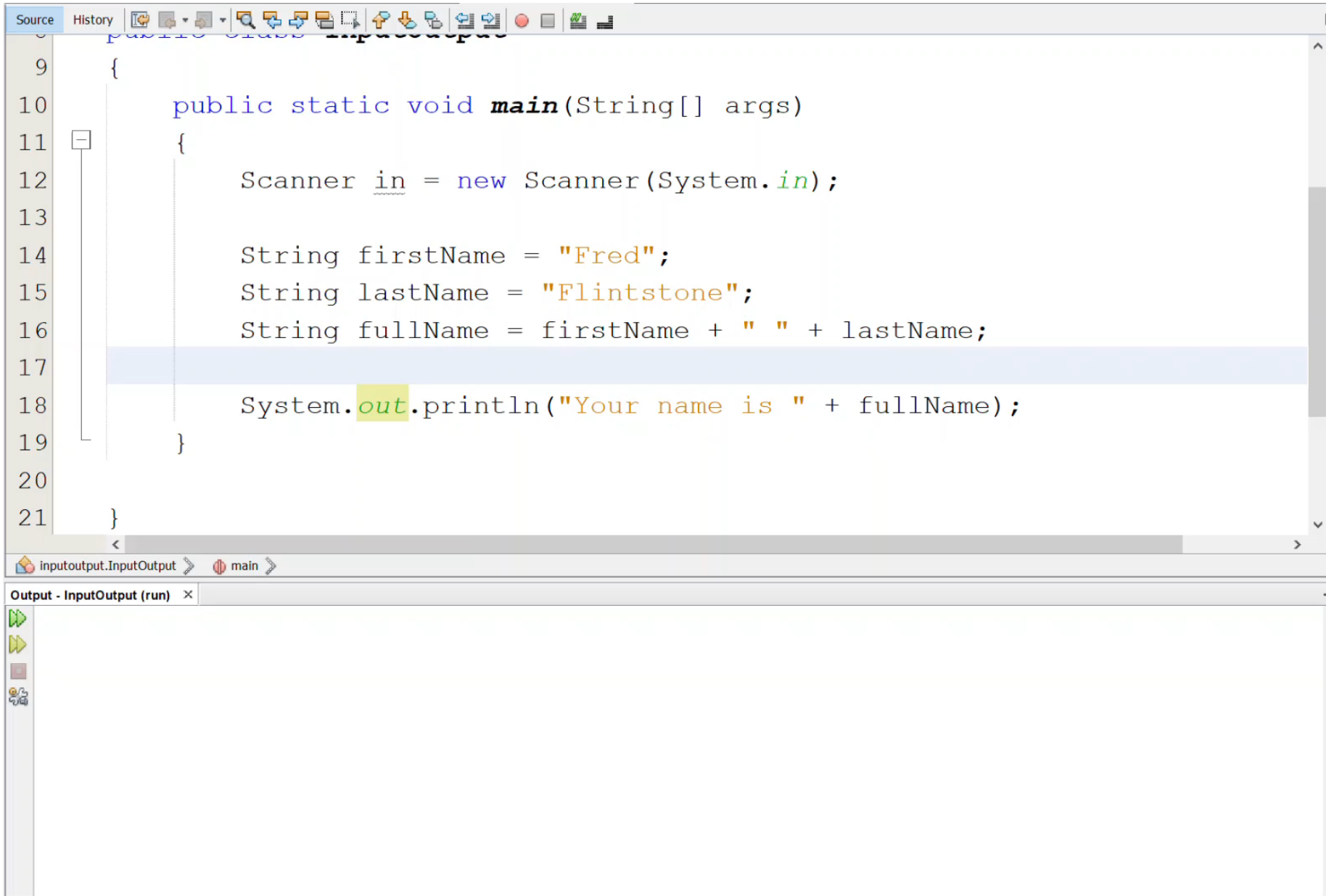
Output - InputOutput (run)

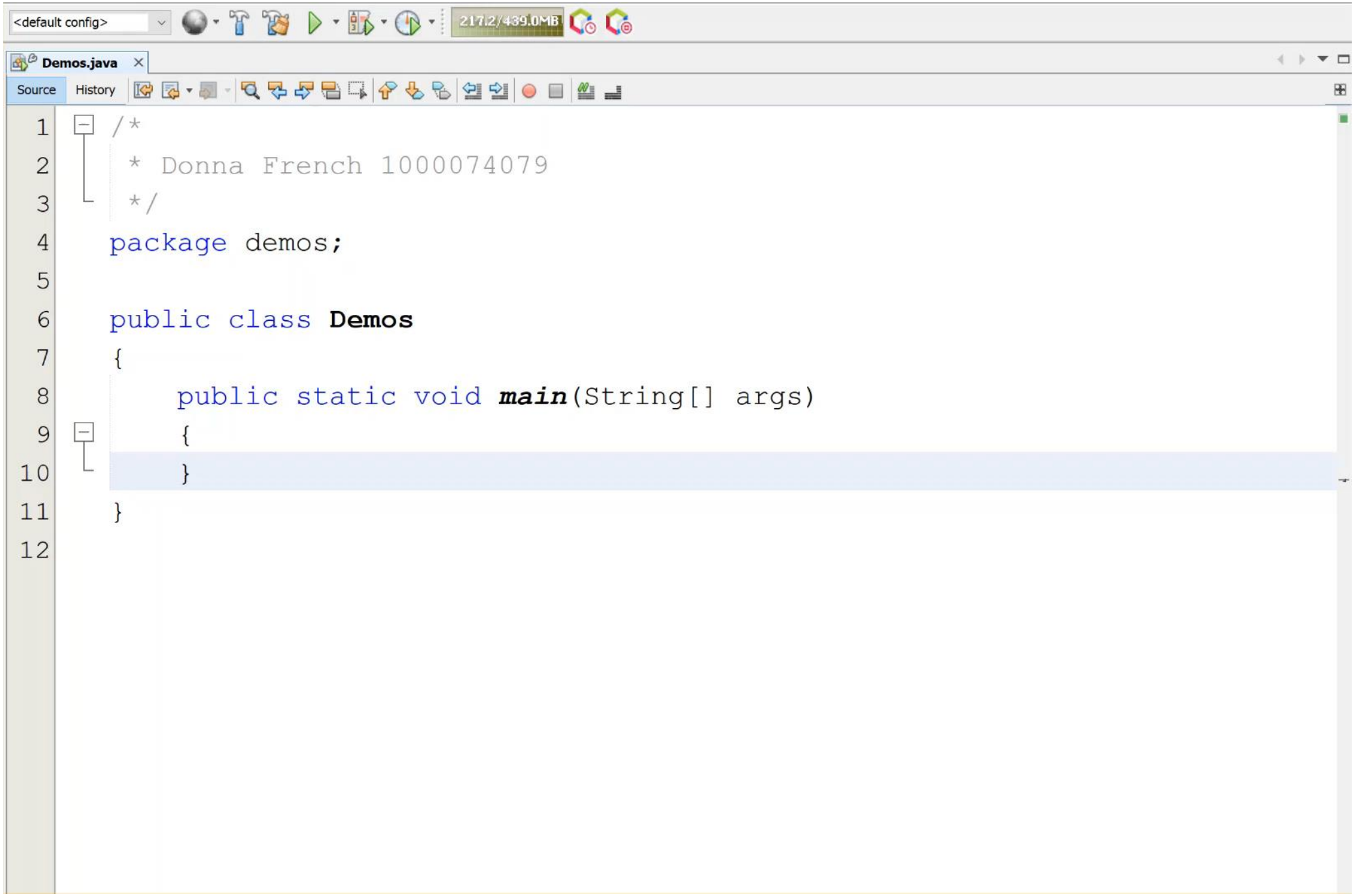
ICQ2 (run) running... 14:34

```
public static void main(String[] args)
{
    Scanner in = new Scanner(System.in);

    String firstName = "Fred";
    String lastName = "Flintstone";
    String fullName = firstName + " " + lastName;

    System.out.println("Your name is " + fullName);
}
```

Concatenation

Whenever one of the arguments of the + operator is a string, then the other argument is converted to a string (if it is not already).




<default config>
217.2/439.0MB
Demos.java
Source
History
1 /*
2 * Donna French 1000074079
3 */
4 package demos;
5
6 public class Demos
7 {
8 public static void main(String[] args)
9 {
10 }
11 }
12

String Input

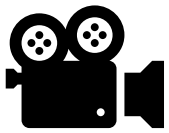
We used `nextInt()` to read integers.

We used `nextDouble()` to read floating point values.

We will use

```
next()
```

to read strings.

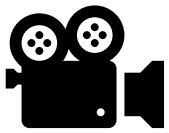


```
Source  History
 9      {
10          public static void main(String[] args)
11          {
12              Scanner in = new Scanner(System.in);
13
14              String firstName = "Fred";
15              String lastName = "Flintstone";
16              String fullName = firstName + " " + lastName;
17              int nameLength = fullName.length();
18
19              System.out.println("Your name is " + fullName);
20              System.out.println("Your name is " + nameLength + " characters l
21          }
```

inputoutput.InputOutput > main > firstName

Output - InputOutput (run)

```
public static void main(String[] args)
{
	Scanner in = new Scanner(System.in);

	System.out.print("Enter your first name ");
	String firstName = in.next();
	System.out.print("Enter your last name ");
	String lastName = in.next();
	String fullName = firstName + " " + lastName;
	int nameLength = fullName.length();

	System.out.println("Your name is " + fullName);
	System.out.println("Your name is " + nameLength + " characters long");
}
```



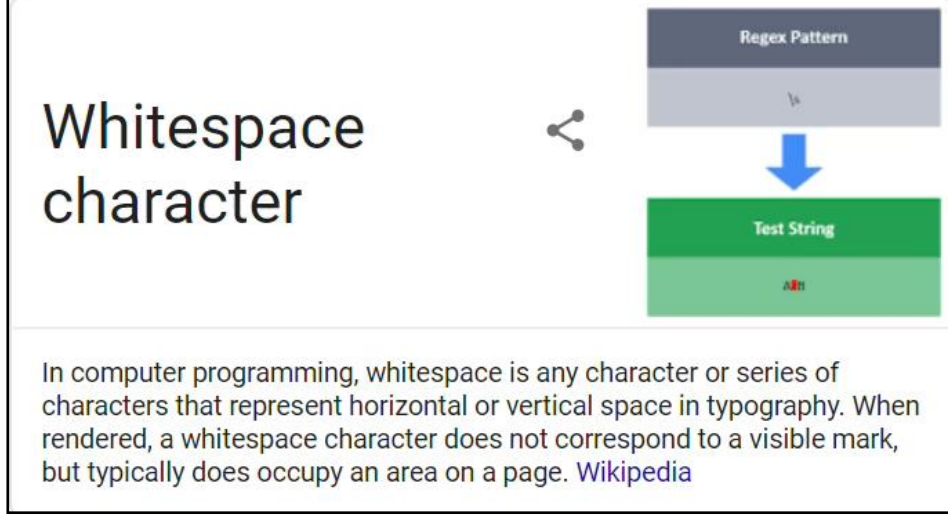
Source History

```
11      {
12          Scanner in = new Scanner(System.in);
13
14          System.out.print("Enter your first name ");
15          String firstName = in.next();
16          System.out.print("Enter your last name ");
17          String lastName = in.next();
18          String fullName = firstName + " " + lastName;
19          int nameLength = fullName.length();
20
21          System.out.println("Your name is " + fullName);
22          System.out.println("Your name is " + nameLength + " characters l
23      }
```

inputoutput.InputOutput > main >

Output - InputOutput (run)

String Input



`next()` will not read through white space

It will stop reading as soon as it encounters a white space.

White space is not just spaces - it is also tabs, line feeds, carriage returns and several other unprintable characters.

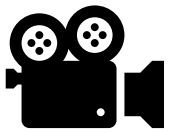
This feature is common among programming languages.

Source History

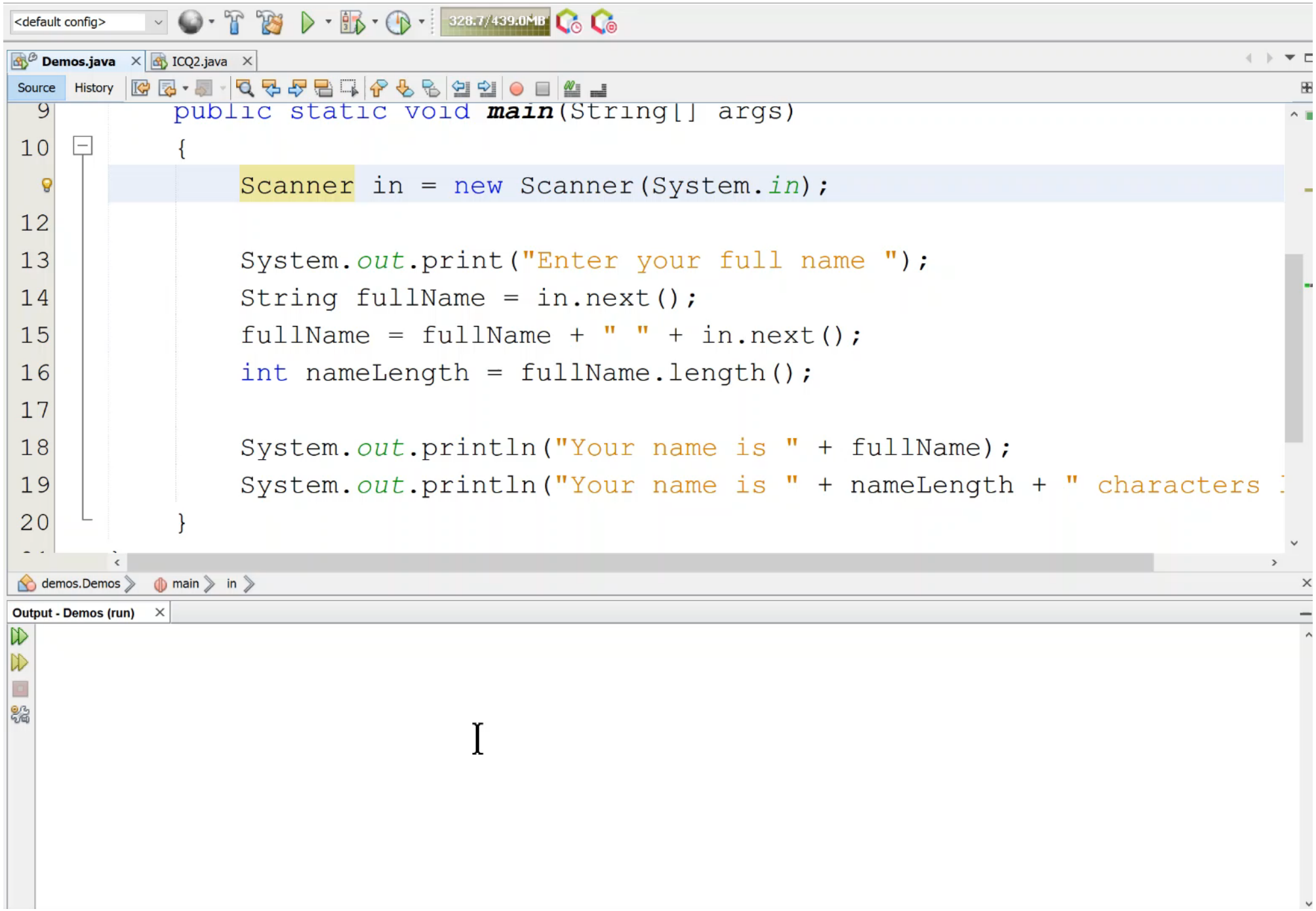
```
12          Scanner in = new Scanner(System.in);
13
14      //      System.out.print("Enter your first name ");
15      //      String firstName = in.next();
16      //      System.out.print("Enter your last name ");
17      //      String lastName = in.next();
18          System.out.print("Enter your full name ");
19          String fullName = in.next();
20          int nameLength = fullName.length();
21
22          System.out.println("Your name is " + fullName);
23          System.out.println("Your name is " + nameLength + " characters l
24      }
```

inputoutput.InputOutput > main > fullName

Output - InputOutput (run)



```
public static void main(String[] args)
{
    Scanner in = new Scanner(System.in);

    System.out.print("Enter your full name ");
    String fullName = in.next();
    fullName = fullName + " " + in.next();
    int nameLength = fullName.length();

    System.out.println("Your name is " + fullName);
    System.out.println("Your name is " + nameLength + " characters long");
}
```


Demos.java
ICQ2.java
Source
History
public static void main(String[] args)
{
Scanner in = new Scanner(System.in);
System.out.print("Enter your full name ");
String fullName = in.next();
fullName = fullName + " " + in.next();
int nameLength = fullName.length();
System.out.println("Your name is " + fullName);
System.out.println("Your name is " + nameLength + " characters
}
demos.Demos
main
in
Output - Demos (run)

String Input

`next()`

reads from the standard input but stops at whitespace

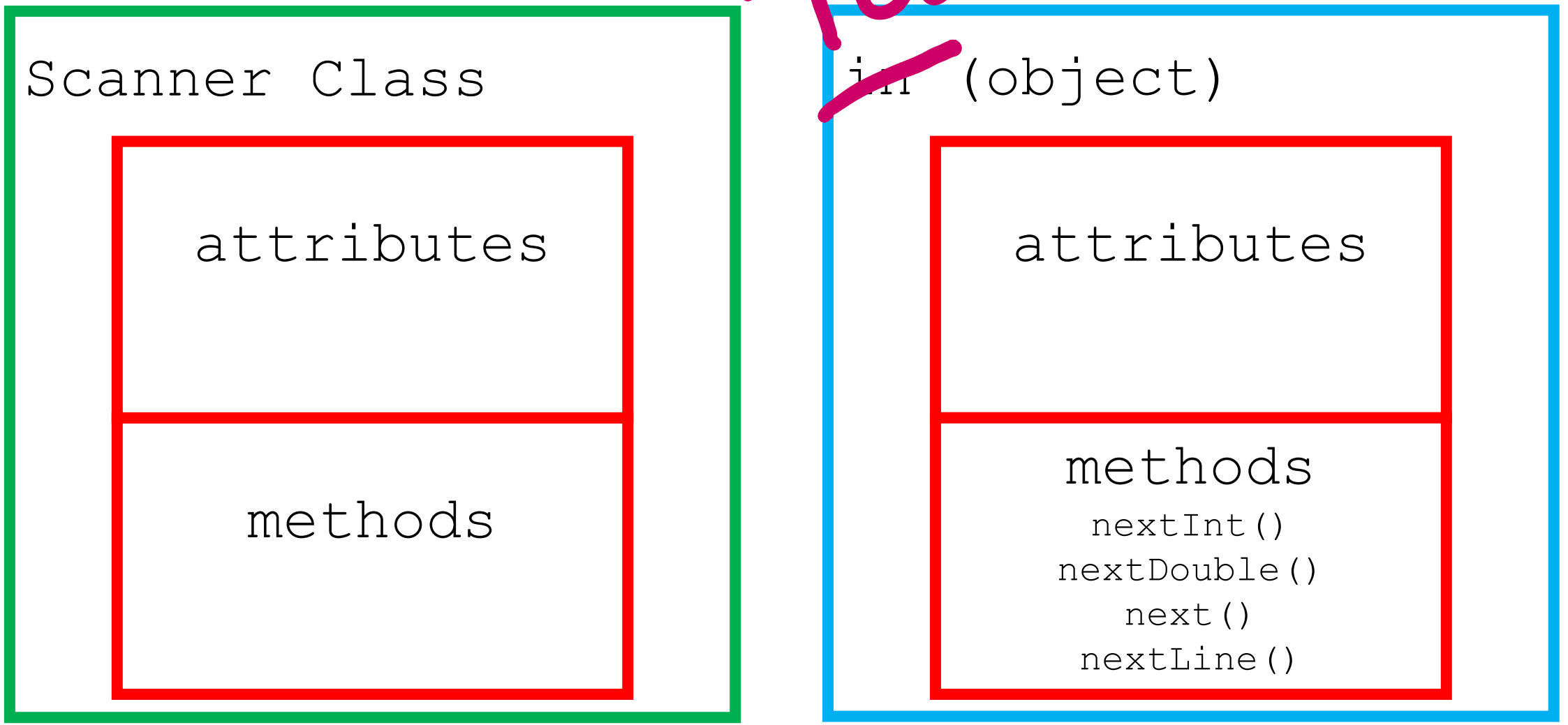
`nextLine()`

reads from the standard input until <ENTER> is pressed

```
Scanner in = new Scanner(System.in);
```

Toad

Toad



Escape Sequences

Escape sequences allow us to print certain characters in our strings.

These characters are normally reserved for use with the language itself.

For example, a string is denoted by surrounding a sequence of characters with quotes. Quotes are part of the syntax.

So how do we print quotes as part of the string itself?

We "escape it"

Source History

```
9    {
10       public static void main(String[] args)
11       {
12           Scanner in = new Scanner(System.in);
13           String myQuote1 = "A famous quote is ";
14           String myQuote2 = "May the Force be with you.";
15
16           System.out.println(myQuote1 + myQuote2);
17       }
18
19   }
20
```

Output - InputOutput (run)

Escape Sequences

- Backspace is replaced with \b.
- Newline is replaced with \n.
- Tab is replaced with \t.
- Carriage return is replaced with \r.
- Form feed is replaced with \f.
- Double quote is replaced with \"
- Single quote is replaced with \'
- Backslash is replaced with \\

Source History

```
9    {
10       public static void main(String[] args)
11       {
12           Scanner in = new Scanner(System.in);
13           String myQuote1 = "A famous quote is ";
14           String myQuote2 = "\"May the Force be with you.\"";
15
16           System.out.println(myQuote1 + myQuote2);
17       }
18
19   }
20
```

inputoutput.InputOutput > main > myQuote2

Output - InputOutput (run)

```
Compiling 1 source file to C:\Users\Donna\Desktop\UTA\CSE1310\Progr
compile:
run:
A famous quote is "May the Force be with you."
BUILD SUCCESSFUL (total time: 1 second)
```

Escape Sequences

Special Note about the newline character in Java

Using `\n` to print a newline character causes the start of a new line on the display.

However, in Windows, you need to add a "return" character `\r` before each `\n`.

Instead, use the `%n` format specifier in a `System.out.printf()` call and it will automatically be translated into `\n` or `\r\n` as needed.

Enumeration

One of the simplest user-defined data type is the enumerated type.

An **enumerated data type** (also called an **enumeration** or **enum**) is a data type where every possible value is defined as a symbolic constant (called an **enumerator**).

Enumerations are defined via the **enum** keyword.

Enumerated Types

```
enum MovieRating
{
    EXCELLENT, AVERAGE, BAD
}

MovieRating rating = MovieRating.AVERAGE;
```

Note that each enumerator is separated by a comma

A semicolon is not needed after the last definition but adding it will not cause a syntax error – it is just ignored.

Enumerated Types

Defining an enumeration does not allocate any memory.

The enum can be printed and the enum variable can be printed

```
MovieRating rating = MovieRating.AVERAGE;
System.out.println(MovieRating.AVERAGE);
System.out.println(rating);
```

```
AVERAGE
```

Enumerated Types

An enumeration provides a way to restrict the values of a variable.

```
rating = MovieRating.EXCELLENT;
```

```
rating = 4;
```

```
error: incompatible types: int cannot be converted to MovieRating
```

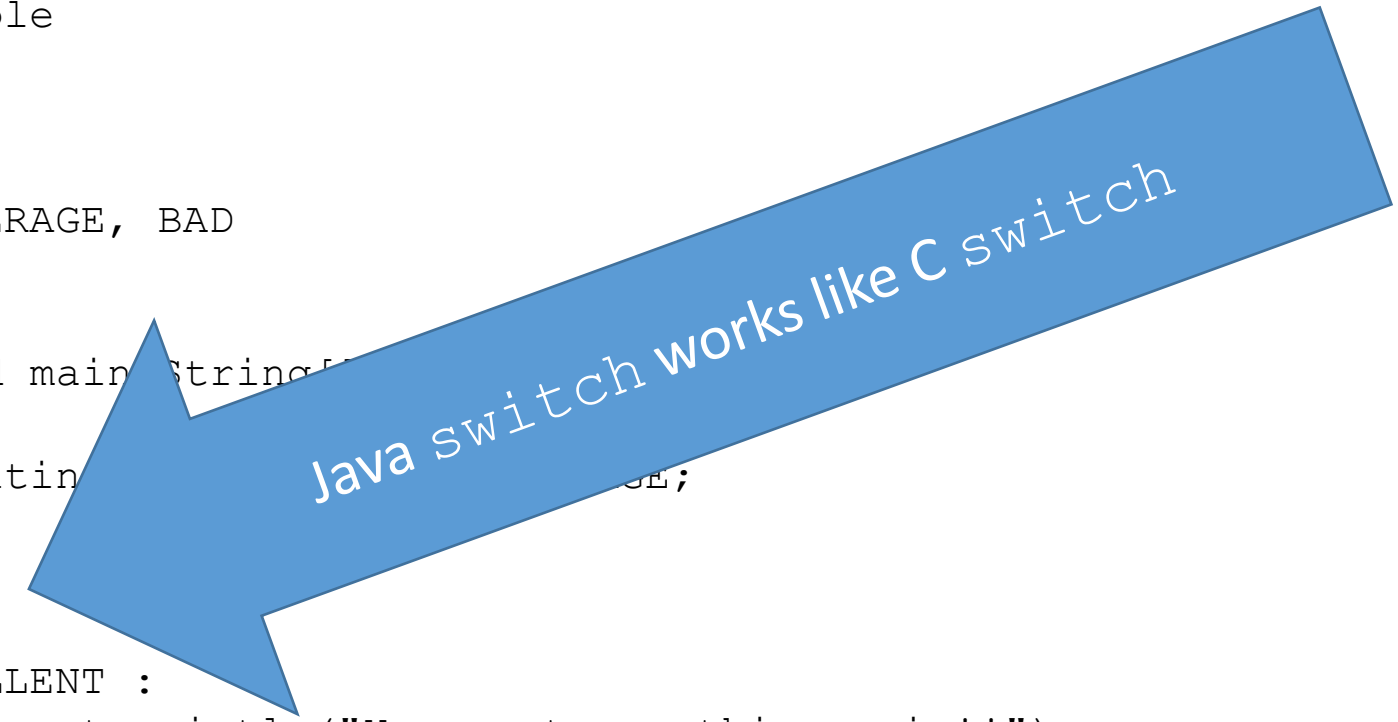
This ability makes enumerations useful in `switch` statements

```
8       enum MovieRating
9       {
10          EXCELLENT, AVERAGE, BAD
11      }
12
13      public static void main(String[] args)
14      {
15          MovieRating rating = MovieRating.AVERAGE;
16
            switch(rating)
18          {
19              case EXCELLENT :
20                  System.out.println("You must see this movie!!");
21                  break;
22              case AVERAGE :
23                  System.out.println("This movie is OK, but not great.");
24                  break;
25              case BAD :
26                  System.out.println("Skip it!");
27                  break;
28              default :
29                  System.out.println("Movie?  I was supposed to watch a movie?");
30                  break;
31          }
```

Output - Enum (run) ×

```
run:
This movie is OK, but not great.
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
public class EnumExample
{
    enum MovieRating
    {
        EXCELLENT, AVERAGE, BAD
    }

    public static void main String[
    {
        MovieRating ratin                    GE;

        switch(rating)
        {
            case EXCELLENT :
                System.out.println("You must see this movie!!");
                break;
            case AVERAGE :
                System.out.println("This movie is OK, but not great.");
                break;
            case BAD :
                System.out.println("Skip it!");
                break;
            default :
                System.out.println("Movie?  I was supposed to watch a movie?");
                break;
        }
    }
}
```



Passing Parameters to Functions

Two basic methods of passing parameters to functions

- *pass by value*
  - parameter is called *value parameter*
  - a copy is made of the current value of the parameter
  - operations in the function are done on the copy – the original does not change

- *pass by reference*
  - parameter is called a *variable parameter*
  - the address of the parameter's storage location is known in the function
  - operations in the function are done directly on the parameter

Passing Parameters to Functions

In C

all parameters are passed by value

the ability to pass by reference does not exist

Pass by reference can be simulated

- pass the address of the variable
- address cannot be modified
- contents of address can be modified

Review from 1320

```
int main(void)
{
  int MyMainNum = 0;

  printf("Before PassByValue call\tMyMainNum = %d\n", MyMainNum);
  PassByValue(MyMainNum);
  printf("After  PassByValue call\tMyMainNum = %d\n", MyMainNum);

  printf("Before PassByRef   call\tMyMainNum = %d\n", MyMainNum);
  PassByRef(&MyMainNum);
  printf("After  PassByRef   call\tMyMainNum = %d\n", MyMainNum);

  return 0;
}
```

Review from 1320

A copy is passed

```
int PassByValue(int MyNum)
{
      MyNum += 100;
      printf("Inside PassByValue\tMyNum      = %d\n", MyNum);
}
```

The address of the actual variable is passed

```
int PassByRef(int *MyNumPtr)
{
      *MyNumPtr += 100;
      printf("Inside PassByRef\tMyRefNum  = %d\n", *MyNumPtr);
}
```

Review from 1320

```
int MyMainNum = 0;

printf("Before PassByValue call\tMyMainNum = %d\n", MyMainNum);
PassByValue(MyMainNum);
printf("After  PassByValue call\tMyMainNum = %d\n", MyMainNum);

int PassByValue(int MyNum)
{
     MyNum += 100;
     printf("Inside PassByValue\tMyNum     = %d\n", MyNum);
}
```

```
Before PassByValue call MyMainNum = 0
Inside PassByValue      MyNum     = 100
After  PassByValue call MyMainNum = 0
```

Review from 1320

```
int MyMainNum = 0;

printf("Before PassByRef   call\tMyMainNum = %d\n", MyMainNum);
PassByRef(&MyMainNum);
printf("After  PassByRef   call\tMyMainNum = %d\n", MyMainNum);

int PassByRef(int *MyNumPtr)
{
     *MyNumPtr += 100;
     printf("Inside PassByRef\tMyNumPtr  = %d\n", *MyNumPtr);
}
```

```
Before PassByRef   call MyMainNum = 0
Inside PassByRef        MyRefNum  = 100
After  PassByRef   call MyMainNum = 100
```

Pass by Reference in Java

Does not really exist*

Why?

Does not really need to…

Pass by Reference in Java

Does not really exist*

When an array is passed to a function, the contents will be updated.

```
8       public static void UpdatePet(int Cat, int Dog)
9       {
10          Cat *= 10;
11          Dog /= 2;
12          System.out.printf("UpdatePet : Cat = %d and Dog = %d\n", Cat, Dog);
13      }
14
15      public static void main(String[] args)
16      {
17          int Cat = 32, Dog =33;
18
19          System.out.printf("main : Cat = %d and Dog = %d\n", Cat, Dog);
20
21          UpdatePet(Cat, Dog);
22
23          System.out.printf("main : Cat = %d and Dog = %d\n", Cat, Dog);
24      }
```

Output - PassByRefKinda (run)

```
run:
main : Cat = 32 and Dog = 33
UpdatePet : Cat = 320 and Dog = 16
main : Cat = 32 and Dog = 33
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
public static void main(String[] args)
{
    int Cat = 32, Dog = 33;
    int Pets[] = {Cat, Dog};

    System.out.printf("main : Cat = %d and Dog = %d\n", Cat, Dog);

    UpdatePet(Pets);
    Cat = Pets[0];
    Dog = Pets[1];

    System.out.printf("main : Cat = %d and Dog = %d\n", Cat, Dog);
}
```

```
public static void UpdatePet(int Pets[])
{
   int Cat = Pets[0];
   int Dog = Pets[1];

   Cat *= 10;
   Dog /= 2;

   System.out.printf("UpdatePet : Cat = %d and Dog = %d\n", Cat, Dog);

   Pets[0] = Cat;
   Pets[1] = Dog;
}
```

```
main : Cat = 32 and Dog = 33
UpdatePet : Cat = 320 and Dog = 16
main : Cat = 320 and Dog = 16
```

String.valueOf()

This method returns the string representation of the argument sent to it.

```
double a = 102939939.939;
int b = -1;

System.out.printf("double a %.3f : %s\n", a, String.valueOf(a));
System.out.printf("int b %d : %s\n", b, String.valueOf(b));
```

```
double a 102939939.939 : 1.02939939939E8
int b -1 : -1
```

String.valueOf()

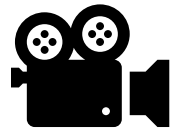
This method returns the string representation of the argument sent to it.

```
long l = 1232874;
char[] arr = {'a', 'b', 'c', 'd', 'e', 'f','g' };

System.out.printf("long l %d : %s\n", l, String.valueOf(l));
System.out.printf("char arr[] {'a', 'b', 'c', 'd', 'e', 'f','g' } : %s\n",
                  String.valueOf(arr));
```

```
long l 1232874 : 1232874
char[] arr {'a', 'b', 'c', 'd', 'e', 'f','g' } : abcdefg
```

String.valueOf()

```
Scanner in = new Scanner(System.in);

int PhraseNumber = 0;
String PhraseString = "";

System.out.println("Enter a number ");
PhraseNumber = in.nextInt();

PhraseString = String.valueOf(PhraseNumber);
System.out.printf("The number you entered has a length of %d",
      PhraseString.length());
```



```
7
8   lic class Demos
9
10  public static void main(String[] args)
11  {
12      Scanner in = new Scanner(System.in);
13
14      int PhraseNumber = 0;
15
16      System.out.println("Enter a number ");
17      PhraseNumber = in.nextInt();
18
19      System.out.printf("The number you entered has a length of %d\n"
20          String.valueOf(PhraseNumber).length());
21
22  }
23
24
```

```
public static void main(String[] args)
{
     Scanner in = new Scanner(System.in);

     System.out.print("Enter the number of seconds ");
     int seconds = in.nextInt();
     System.out.printf("%s\n", displayTime(seconds));
}
```

```
public static String displayTime(int time)
{
   String minutes = String.valueOf(time / 60);

   String seconds = String.valueOf(time % 60);

   return minutes + " minutes and " +
          (seconds.length() == 1 ? "0" : "") + seconds +
          " second(s)";
}
```

```
Enter the number of seconds 100
1 minutes and 40 second(s)

Enter the number of seconds 8
0 minutes and 08 second(s)

Enter the number of seconds 1001
16 minutes and 41 second(s)
```

Abstraction

In order to use a method, you only need to know its name, inputs, outputs, and where it lives.

You don’t need to know how it works, or what other code it’s dependent upon to use it.

This lowers the amount of knowledge required to use other people’s code (including everything in the standard library).

Strings and Characters

- Strings are sequences of Unicode characters.
  - Unicode is an extended version of ASCII
  - ASCII is basically the set of the keys on your keyboard
  - every character in ASCII/Unicode has a numeric value

Java uses

`int`
: to represent numbers (integers)

`double`
: to represent fractional numbers

`char`
: to represent single characters

Ascii	Char	Ascii	Char	Ascii	Char	Ascii	Char
0	Null	32	Space	64	@	96	`
1	Start of heading	33	!	65	A	97	a
2	Start of text	34	"	66	B	98	b
3	End of text	35	#	67	C	99	c
4	End of transmit	36	$	68	D	100	d
5	Enquiry	37	%	69	E	101	e
6	Acknowledge	38	&	70	F	102	f
7	Audible bell	39	'	71	G	103	g
8	Backspace	40	(	72	H	104	h
9	Horizontal tab	41	)	73	I	105	i
10	Line feed	42	*	74	J	106	j
11	Vertical tab	43	+	75	K	107	k
12	Form feed	44	,	76	L	108	l
13	Carriage return	45	-	77	M	109	m
14	Shift in	46	.	78	N	110	n
15	Shift out	47	/	79	O	111	o
16	Data link escape	48	0	80	P	112	p
17	Device control 1	49	1	81	Q	113	q
18	Device control 2	50	2	82	R	114	r
19	Device control 3	51	3	83	S	115	s
20	Device control 4	52	4	84	T	116	t
21	Neg. acknowledge	53	5	85	U	117	u
22	Synchronous idle	54	6	86	V	118	v
23	End trans. block	55	7	87	W	119	w
24	Cancel	56	8	88	X	120	x
25	End of medium	57	9	89	Y	121	y
26	Substitution	58	:	90	Z	122	z
27	Escape	59	;	91	[	123	{
28	File separator	60	<	92	\	124	|
29	Group separator	61	=	93	]	125	}
30	Record separator	62	>	94	^	126	~
31	Unit separator	63	?	95	_	127	Forward del.

A	65
a	97
0	48
Space	32

Strings and Characters

- A string literal is a sequence of characters enclosed in double quotes.

  "Hello"

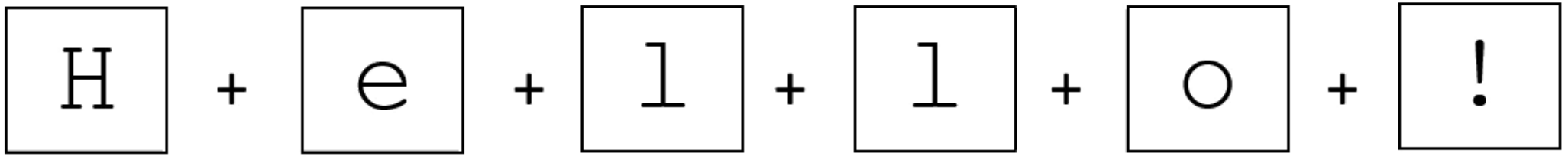
- A character literal is a single character enclosed in single quotes.

  'H'

We can store a character literal in a variable of type `char`

Strings and Characters

Linking together multiple characters creates a string.



The letter in the first position of the string is labeled 0. 'H' is the 0th character in the string.

The letter in the second position of the string is labeled 1. 'e' is the 1st character in the string.

H	e	l	l	o	!
0	1	2	3	4	5

Strings and Characters

H	e	l	l	o	!
0	1	2	3	4	5

The position number of the last character is one less than the length of the string.

The length of the "Hello!" string is 6.

The position number of the "!" is 5.