

# CS 106A, Lecture 8

## Characters and Strings

suggested reading:

*Java Ch. 8.1-8.4*

# Plan For Today

- Announcements
- Recap
  - Parameters
  - Return
- Random Numbers
- Text Processing
  - Characters
  - Strings

# Announcements

- HW 2 reminder: some features not allowed
  - Prohibited: parameters, returns, Strings, global variables, anything after 7/3 lecture
  - Allowed: constants, Math.sqrt
- Output comparison tool updated (see site)
- [Exam conflict form](#) due **today at 5PM**
  - See Midterm link on website's sidebar
- Colin's office hours updated

# Plan For Today

- Announcements
- Recap
  - Parameters
  - Return
- Random Numbers
- Text Processing
  - Characters
  - Strings

# Parameters

Parameters let you provide a method some information when you are calling it.

# Return

Return values let you give back some information when a method is finished.

# Methods = Toasters



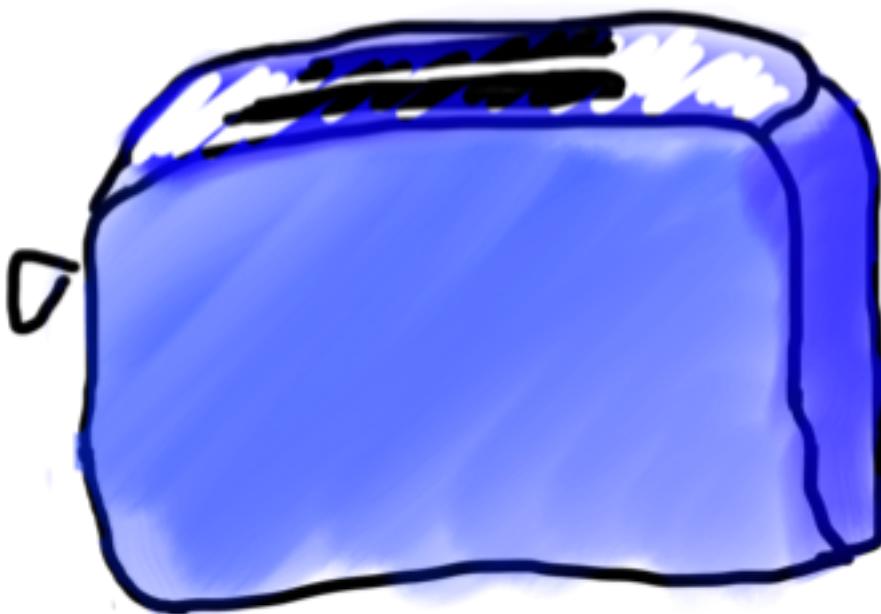
parameter



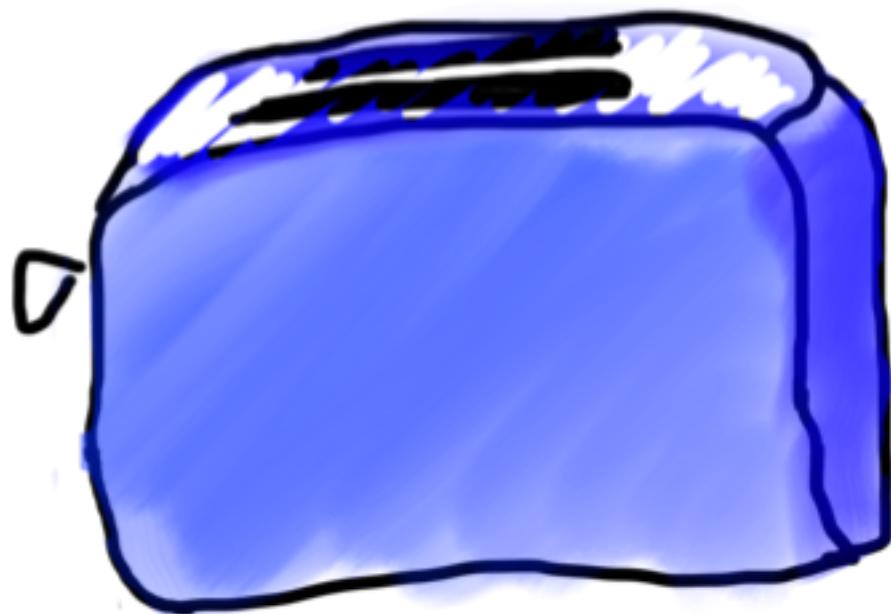
# Methods = Toasters



parameter



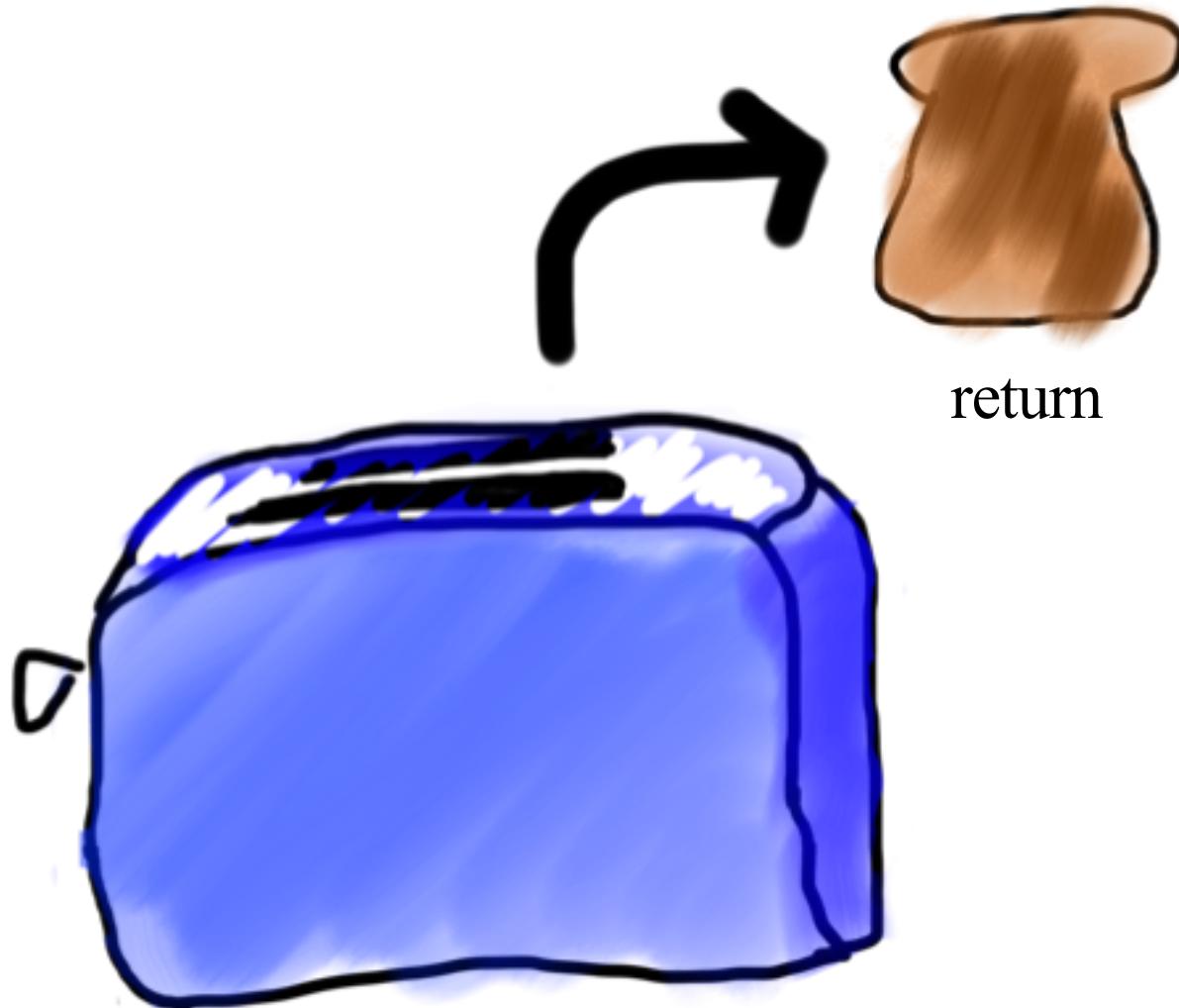
# Methods = Toasters



# Methods = Toasters



# Methods = Toasters



# Example: readInt

```
int x = readInt("Your guess? ");
```

# Example: readInt

We call  
readInt



```
int x = readInt("Your guess? ");
```

We give readInt some  
information (the text to  
print to the user)



# Example: `readInt`

When we include values in the parentheses of a method call, this means we are passing them as *parameters* to this method.

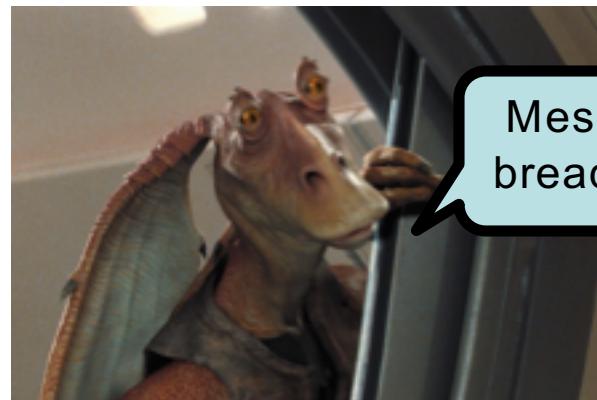
```
int x = readInt("Your guess? ");
```

# Example: readInt

When we include values in the parentheses of a method call, this means we are passing them as *parameters* to this method.

```
int x = readInt("Your guess? ");
```

Par Par Binks  
Parameters in Parentheses



Mesa love puttin'  
bread in toasters!

# Example: readInt

When finished, readInt gives us information back (the user's number) and we put it in x.



```
int x = readInt("Your guess? ");
```

# Example: readInt

When we set a variable equal to a method, this tells Java to save the return value of the method in that variable.

```
int x = readInt("Your guess? ");
```

# Example: readInt

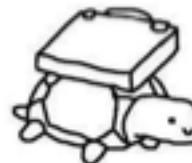
When we set a variable equal to a method, this tells Java to save the return value of the method in that variable.

```
int x = readInt("Your guess? ");
```

tARTL

to Assign, Right to Left

that  
turtle  
has a suitcase.  
it must be going to work.  
i hope that the turtle has a good day.



# Plan For Today

- Announcements

- Recap

- Parameters

- Return

- Random Numbers

- Text Processing

- Characters

- Strings

*The following drawBox example has animations  
that do not show up in the lecture slides—  
please watch the lecture video*

# Parameters Example: drawBox

Tells Java this method  
needs two *ints* in order to  
execute.



```
private void drawBox(int width, int height) {  
    // use width and height variables  
    // to draw a box  
}
```

# Parameters Example: drawBox

*Inside drawBox, refer to  
the first parameter value  
as width...*



```
private void drawBox(int width, int height) {  
    // use width and height variables  
    // to draw a box  
}
```

# Parameters Example: drawBox

...and the second  
parameter value as *height*.



```
private void drawBox(int width, int height) {  
    // use width and height variables  
    // to draw a box  
}
```

# drawBox

We call  
drawBox



```
drawBox( 7, 4 );
```

We give drawBox some  
information (the size of  
the box we want)

# drawBox

```
int width = readInt("Width? ");  
int height = readInt("Height? ");  
...
```

We call  
drawBox

`drawBox(width, height);`

We give drawBox some  
information (the size of  
the box we want)

# drawBox

```
int width = readInt("Width? ");    7
int height = readInt("Height? ");  4
...
drawBox(width, height);
```

# drawBox

```
int width = readInt("Width? ");    7  
int height = readInt("Height? ");  4  
...  
7   4
```

```
drawBox(width, height);
```

# drawBox

```
int width = readInt("Width? ");    7
int height = readInt("Height? ");  4
...
drawBox(7, 4);
```

# drawBox

First  
parameter  
to drawBox

7

Second  
parameter to  
drawBox

4

# drawBox

7 4

# drawBox

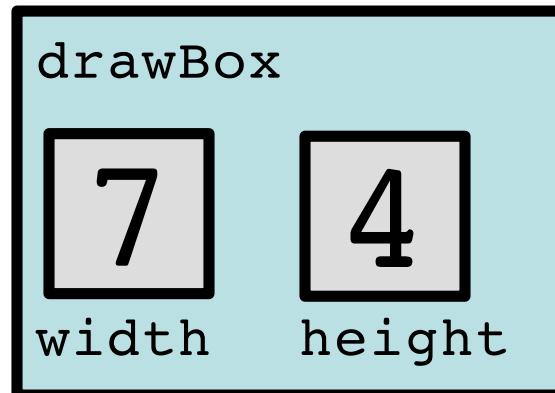
```
private void drawBox(int width, int height) {  
    // use width and height variables  
    // to draw a box  
}
```

7

4

# drawBox

```
private void drawBox(int width, int height) {  
    ...  
}
```



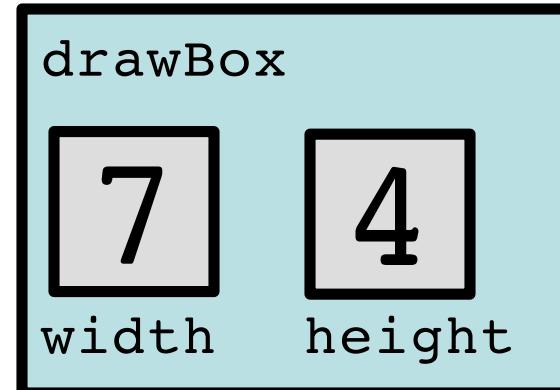
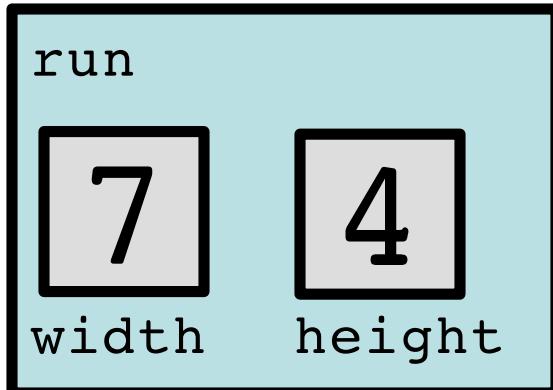
# Parameter Names

Parameter names do not affect program behavior.

# Parameter Names

```
public void run() {  
    int width = readInt("Width? ");      7  
    int height = readInt("Height? ");     4  
    drawBox(width, height);  
}  
}
```

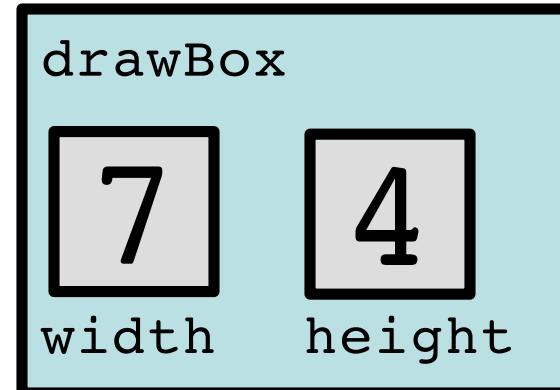
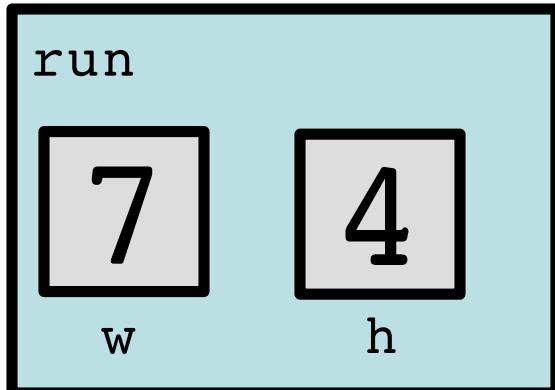
```
private void drawBox(int width, int height) {  
    ...  
}
```



# Parameter Names

```
public void run() {  
    int w = readInt("Width? "); 7  
    int h = readInt("Height? "); 4  
    drawBox(w, h);  
}
```

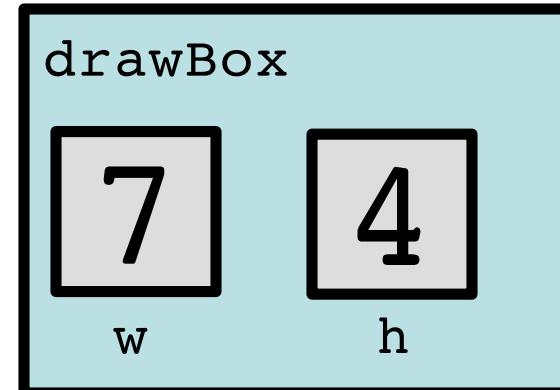
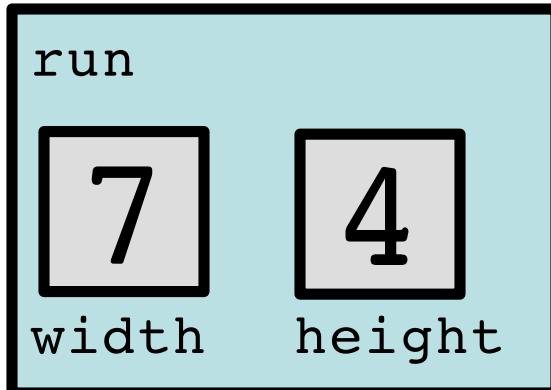
```
private void drawBox(int width, int height) {  
    ...  
}
```



# Parameter Names

```
public void run() {  
    int width = readInt("Width? ");      7  
    int height = readInt("Height? ");     4  
    drawBox(width, height);  
}  
}
```

```
private void drawBox(int w, int h) {  
    ...  
}
```



# Plan For Today

- Announcements
- Recap
  - Parameters
  - Return
- Random Numbers
- Text Processing
  - Characters
  - Strings

# Return Example: metersToCm

When this method finishes,  
it will return a *double*.



```
private double metersToCm(double meters) {  
    ...  
}
```

# Return Example: metersToCm

```
private double metersToCm(double meters) {  
    double centimeters = meters * 100;  
    return centimeters;  
}
```



Returns the *value of this expression* (centimeters).

# Return Example: metersToCm

Setting a variable *equal* to a method means we save the method's return value in that variable.

```
public void run() {  
    double cm = metersToCm(5.0);  
    ...  
}
```

# Return Example: metersToCm

```
public void run() {  
    double meters = readDouble("# meters? ");  
    ...  
  
    double cm = metersToCm(meters);  
    println(cm + " centimeters.");  
}  
  
private double metersToCm(double meters) {  
    double centimeters = meters * 100;  
    return centimeters;  
}
```

# Return Example: metersToCm

```
public void run() {    7.0
    double meters = readDouble("# meters? ");
    ...
    double cm = metersToCm(meters);
    println(cm + " centimeters.");
}

private double metersToCm(double meters) {
    double centimeters = meters * 100;
    return centimeters;
}
```

# Return Example: metersToCm

```
public void run() {    7.0
    double meters = readDouble("# meters? ");
    ...
    double cm = metersToCm(meters);
    println(cm + " centimeters.");
}

private double metersToCm(double meters) {
    double centimeters = meters * 100;
    return centimeters;
}
```

# Return Example: metersToCm

```
public void run() {    7.0
    double meters = readDouble("# meters? ");
    ...
    double cm = metersToCm(meters);
    println(cm + " centimeters.");
}

private double metersToCm(double meters) {
    double centimeters = meters * 100;
    return centimeters;
}                                700.0
```

# Return Example: metersToCm

```
public void run() {    7.0
    double meters = readDouble("# meters? ");
    ...
    700.0
    double cm = metersToCm(meters);
    println(cm + " centimeters.");
}
```

# Return Values and Expressions

```
public void run() {    7.0
    double meters = readDouble("# meters? ");
    println(metersToCm(meters) + " cm.");
}

private double metersToCm(double meters) {
    ...
}
```

# Return Values and Expressions

```
public void run() {    7.0
    double meters = readDouble("# meters? ");
    println(metersToCm(meters) + " cm.");
}

private double metersToCm(double meters) {
    ...
}
```

You can use a method's return  
value *directly in an expression.*

# Buggy Example!

```
public void run() {    7.0
    double meters = readDouble("# meters? ");
    ...
    metersToCm(meters); // Does nothing!
    ...
}
```

# Buggy Example!

```
public void run() {    7.0
    double meters = readDouble("# meters? ");
    ...
    700.0
metersToCm(meters); // Does nothing!
    ...
}
```

# Factorial Code Walkthrough

See extra slides

# Plan For Today

- Announcements
- Recap
  - Parameters
  - Return
- Random Numbers
- Text Processing
  - Characters
  - Strings

# RandomGenerator

```
import acm.util.*;
```

Method	Description
RandomGenerator.getInstance().nextInt( <i>min</i> , <i>max</i> )	a random integer in the given range, inclusive

```
// random number from 0-9 inclusive
int digit = RandomGenerator.getInstance().nextInt(0, 9);
println(digit);
```

```
// prints "hello!" between 3-6 times
int times = RandomGenerator.getInstance().nextInt(3, 6);
for (int i = 0; i < times; i++) {
    println("hello!");
}
```

# RandomGenerator

The `RandomGenerator` class defines the following methods:

`int nextInt(int low, int high)`

Returns a random `int` between `low` and `high`, inclusive.

`int nextInt(int n)`

Returns a random `int` between 0 and `n - 1`.

`double nextDouble(double low, double high)`

Returns a random `double`  $d$  in the range  $\text{low} \leq d < \text{high}$ .

`double nextDouble()`

Returns a random `double`  $d$  in the range  $0 \leq d < 1$ .

`boolean nextBoolean()`

Returns a random `boolean` value, which is `true` 50 percent of the time.

`boolean nextBoolean(double p)`

Returns a random `boolean`, which is `true` with probability `p`, where  $0 \leq p \leq 1$ .

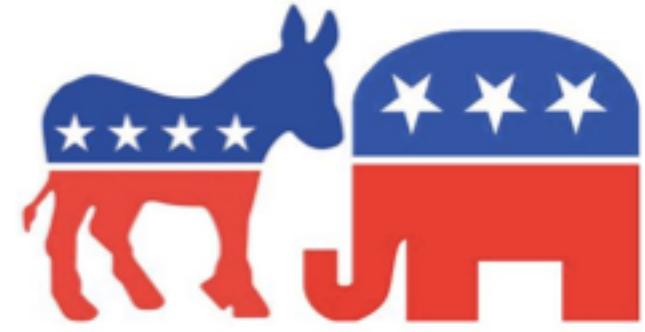
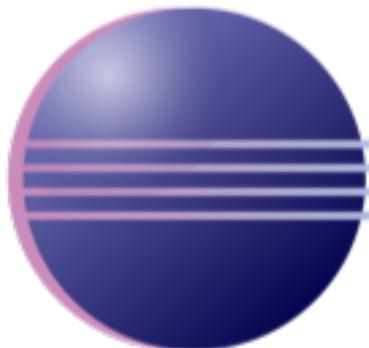
`Color nextColor()`

Returns a random color.

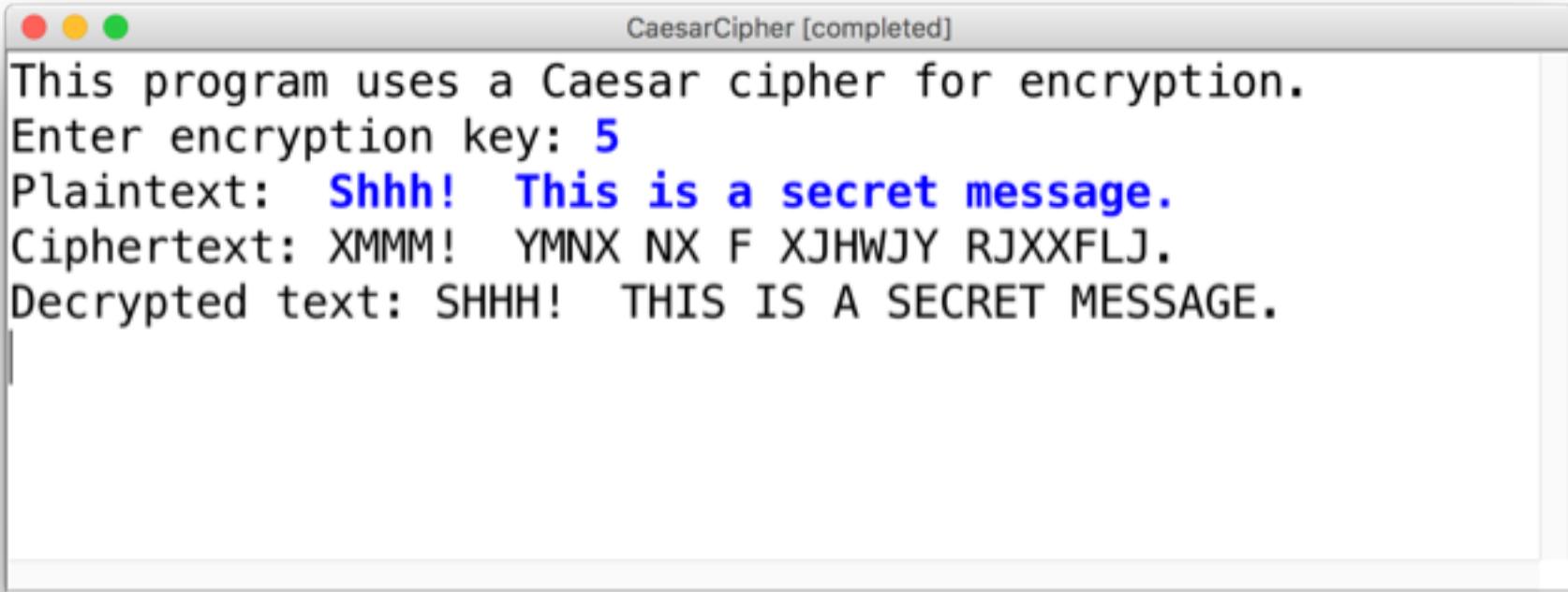
# Plan For Today

- Announcements
- Recap
  - Parameters
  - Return
- Random Numbers
- Text Processing
  - Characters
  - Strings

# Text Processing



# Goal



```
CaesarCipher [completed]
This program uses a Caesar cipher for encryption.
Enter encryption key: 5
Plaintext: Shhh! This is a secret message.
Ciphertext: XMNM! YMNX NX F XJHWJY RJXXFLJ.
Decrypted text: SHHH! THIS IS A SECRET MESSAGE.
```

After learning text processing, we will be able to write our own encryption program!

# Plan For Today

- Announcements
- Recap
  - Parameters
  - Return
- Random Numbers
- Text Processing
  - Characters
  - Strings

# Char

A **char** is a variable type that represents a single character or “glyph”.

```
char letterA = 'A';
char plus = '+';
char zero = '0';
char space = ' ';
char newLine = '\n';
char tab = '\t';
char singleQuote = '\'';
char backSlash = '\\';
```

# Char

Under the hood, Java represents each **char** as an *integer* (its “ASCII value”).

- Uppercase letters are sequentially numbered
- Lowercase letters are sequentially numbered
- Digits are sequentially numbered

```
char uppercaseA = 'A';           // Actually 65
char lowercaseA = 'a';           // Actually 97
char zeroDigit = '0';           // Actually 48
```

# Char Math!

We can take advantage of Java representing each **char** as an *integer* (its “ASCII value”):

```
boolean areEqual = 'A' == 'A';           // true
boolean earlierLetter = 'f' < 'c'; // false
char uppercaseB = 'A' + 1;
int diff = 'c' - 'a';                  // 2
int numLettersInAlphabet = 'z' - 'a' + 1;
// or
int numLettersInAlphabet = 'z' - 'A' + 1;
```

# Char Math!

We can take advantage of Java representing each **char** as an *integer* (its “ASCII value”):

```
// prints out every character
for (char ch = 'a'; ch <= 'z'; ch++) {
    print(ch);
}
```

# Char Math!

Not every integer maps to a character. So when you have an expression with **ints** and **chars**, Java picks **int** as the *most expressive type*.

```
'A' + 1           // evaluates to 66 (int)  
'c' + (2*5) - 1 // evaluates to 108
```

We can make it a char by putting it in a char variable.

```
char uppercaseB = 'A' + 1;  
// or  
char uppercaseB = 66;
```

# Side Note: Type-casting

If we want to force Java to treat an expression as a particular type, we can also *cast it* to that type.

```
'A' + 1          // evaluates to 66 (int)
(char)('A' + 1)  // evaluates to 'B' (char)
(char)'A' + 1    // evaluates to 66 (int)

1 / 2            // evaluates to 0 (int)
(double)1 / 2    // evaluates to 0.5 (double)
1 / (double)2    // evaluates to 0.5 (double)
```

# Character Methods

There are some helpful built-in Java methods to manipulate **chars**.

```
char lowercaseA = 'a';
char uppercaseA = Character.toUpperCase(lowercaseA);

char plus = '+';
if (Character.isLetter(plus)) {
    ... // Does not execute: + is not a letter
}
```

# Character Methods

Method	Description
Character.isDigit( <b>ch</b> )	true if <i>ch</i> is '0' through '9'
Character.isLetter( <b>ch</b> )	true if <i>ch</i> is 'a' through 'z' or 'A' through 'Z'
Character.isLetterOrDigit( <b>ch</b> )	true if <i>ch</i> is 'a' through 'z', 'A' through 'Z' or '0' through '9'
Character.isLowerCase( <b>ch</b> )	true if <i>ch</i> is 'a' through 'z'
Character.isUpperCase( <b>ch</b> )	true if <i>ch</i> is 'A' through 'Z'
Character.toLowerCase( <b>ch</b> )	returns lowercase equivalent of a letter
Character.toUpperCase( <b>ch</b> )	returns uppercase equivalent of a letter
Character.isWhitespace( <b>ch</b> )	true if <i>ch</i> is a space, tab, new line, etc.

Remember: `toLowerCase` and  
`toUpperCase` return the new char;  
they cannot modify an existing char!

# Character Methods

Remember to always save the return value of Character methods!

```
char lowercaseA = 'a';
Character.toUpperCase(lowercaseA); // Does nothing!
println(lowercaseA);           // prints 'a'!
```

```
char uppercaseA =
    Character.toUpperCase(lowercaseA); // OK
println(uppercaseA);           // prints 'A' !
```

# Plan For Today

- Announcements
- Recap
  - Parameters
  - Return
- Random Numbers
- Text Processing
  - Characters
  - Strings

# Strings

A **String** is a variable type representing a sequence of characters.

```
String text = "Hi parents!";
```

<i>index</i>	0	1	2	3	4	5	6	7	8	9	10
<i>character</i>	'H'	'i'	' '	'p'	'a'	'r'	'e'	'n'	't'	's'	'!'

- Each character is assigned an *index*, going from 0 to length-1
- There is a **char** at each index

# Creating Strings

```
String str = "Hello, world!";  
String empty = "";  
println(str);
```

// Read in text from the user

```
String name = readLine("What is your name? ");
```

// String concatenation (using "+")

```
String message = 2 + " cool " + 2 + " handle";  
int x = 2;  
println("x has the value " + x);
```

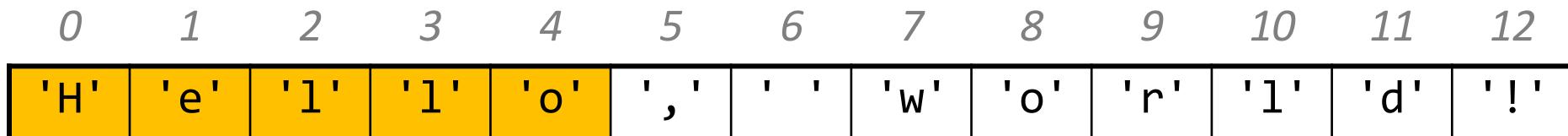
# Common String Operations

```
String str = "Hello, world!";  
  
// Length  
int strLength = str.length();           // 13  
  
// Access individual characters  
char firstLetter = str.charAt(0);  
char lastLetter = str.charAt(strLength - 1);  
char badTimes = str.charAt(strLength); // ERROR
```

# Substrings

A *substring* is a subset of a string.

```
String str = "Hello, world!";
String hello = str.substring(0, 5);
```



# Substrings

A *substring* is a subset of a string.

```
String str = "Hello, world!";
```

```
String worldExclm = str.substring(7, 13);
```

0	1	2	3	4	5	6	7	8	9	10	11	12
'H'	'e'	'l'	'l'	'o'	','	' '	'w'	'o'	'r'	'l'	'd'	'!'

# Substrings

A *substring* is a subset of a string.

```
String str = "Hello, world!";
String worldExclm = str.substring(7); // to end
```

0	1	2	3	4	5	6	7	8	9	10	11	12
'H'	'e'	'l'	'l'	'o'	','	' '	'w'	'o'	'r'	'l'	'd'	'!'

# String Methods

Method name	Description
<code>s.length()</code>	number of characters in this string
<code>s.charAt(index)</code>	char at the given index
<code>s.indexOf(str)</code>	index where the start of the given string appears in this string (-1 if not found)
<code>s.substring(index1, index2)</code> or <code>s.substring(index1)</code>	the characters in this string from <i>index1</i> (inclusive) to <i>index2</i> (exclusive); if <i>index2</i> is omitted, goes until end
<code>s.toLowerCase()</code>	a new string with all lowercase letters
<code>s.toUpperCase()</code>	a new string with all uppercase letters

- These methods are called using **dot notation**:

```
String className = "CS 106yay!";  
println(className.length()); // 10
```

# Strings are Immutable

Once you create a String, its contents **cannot be changed**.

```
// Cannot change individual chars in the string
String typo = "Hello, warld!";
typo.charAt(8) = 'o'; // Error! Will not run.
```

To change a String, you must create a *new* String containing the value you want (e.g. using String methods).

```
String corrected = "Hello, world!";
```

# Strings are Immutable

```
String className = "cs 106a";
className.toUpperCase();           // does nothing!

className = className.toUpperCase(); // 
println(className);               // CS 106A
```

# Comparing Strings

```
String greeting = "Hello!";
if (greeting == "Hello!") {    // Doesn't work!
```

...

}

// Instead:

```
if (greeting.equals("Hello!")) {
```

...

}

*Always use .equals instead of == and !=*

# Comparing Strings

Method	Description
<code>s1.equals(s2)</code>	whether two strings contain the same characters
<code>s1.equalsIgnoreCase(s2)</code>	whether two strings contain the same characters, ignoring upper vs. lower case
<code>s1.startsWith(s2)</code>	whether <b>s1</b> contains <b>s2</b> 's characters at start
<code>s1.endsWith(s2)</code>	whether <b>s1</b> contains <b>s2</b> 's characters at end
<code>s1.contains(s2)</code>	whether <b>s2</b> is found within <b>s1</b>

# Looping Over Strings

A common String programming pattern is looping over a string and operating on each character.

```
String str = "Hello!";
for (int i = 0; i < str.length(); i++) {
    char ch = str.charAt(i);
    // Do something with ch here
}
```

# Looping Over Strings

A common String programming pattern is looping over a string and operating on each character.

```
// Prints out each letter on a separate line
String str = "Hello!";
for (int i = 0; i < str.length(); i++) {
    char ch = str.charAt(i);
    println(ch);
}
```

# Looping Over Strings

A common String programming pattern is looping over a string and operating on each character.

```
// Creates a new String in all caps
String str = "Hello!";
String newStr = "";
for (int i = 0; i < str.length(); i++) {
    char ch = str.charAt(i);
    newStr += Character.toUpperCase(ch);
}
println(newStr);          // HELLO!
```

# Recap

- Recap
  - Parameters
  - Return
- Random Numbers
- Text Processing
  - Characters
  - Strings

**Next time:** problem-solving with Strings

```
public void run() {        4
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "!" + factorial(i));
    }
}
```

```
public void run() {  
    for int i = 0; i < MAX_NUM; i++) {  
        println(i + " ! = " + factorial(i));  
    }  
}
```

```
public void run() {  
    for int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" = " + factorial(i));  
    }  
}
```

i 0

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```

i 0

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + " ! = " + factorial(i));  
    }  
}
```

i 0

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```

i 0

```
private int factorial(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; i++) {  
        result *= i;  
    }  
    return result;  
}
```

n 0 result  i

```
private int factorial(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; i++) {  
        result *= i;  
    }  
    return result;  
}
```

n  result  i

```
private int factorial(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; i++) {  
        result *= i;  
    }  
    return result;  
}
```

n  result  i

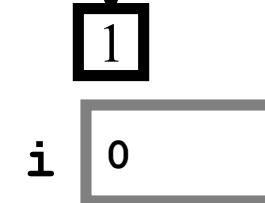
```
private int factorial(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; i++) {  
        result *= i;  
    }  
    return result;  
}
```

n  result  i

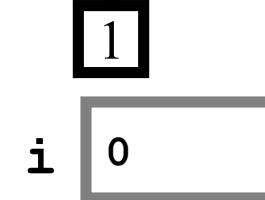
```
private int factorial(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; i++) {  
        result *= i;  
    }  
    return result;  
}
```

n 0      result 1      i 1

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```



```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + " ! = " + factorial(i));  
    }  
}
```



0 ! = 1

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```

i 0

0! = 1

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```

i 1

0! = 1

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + " ! = " + factorial(i));  
    }  
}
```

i 1

0 ! = 1

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + " ! = " + factorial(i));  
    }  
}
```

i 1

0 ! = 1

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + " ! = " + factorial(i));  
    }  
}
```

i 1

0 ! = 1

```
private int factorial(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; i++) {  
        result *= i;  
    }  
    return result;  
}
```

n 1      result       i

0! = 1

```
private int factorial(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; i++) {  
        result *= i;  
    }  
    return result;  
}
```

n  result  i

$0! = 1$

```
private int factorial(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; i++) {  
        result *= i;  
    }  
    return result;  
}
```

n  result  i

0! = 1

```
private int factorial(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; i++) {  
        result *= i;  
    }  
    return result;  
}
```

n  result  i

0! = 1

```
private int factorial(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; i++) {  
        result *= i;  
    }  
    return result;  
}
```

n  result  i

0! = 1

```
private int factorial(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; i++) {  
        result *= i;  
    }  
    return result;  
}
```

n  result  i

0! = 1

```
private int factorial(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; i++) {  
        result *= i;  
    }  
    return result;  
}
```

n  result  i

$0! = 1$

```
private int factorial(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; i++) {  
        result *= i;  
    }  
    return result;  
}
```

n  result  i

0! = 1

```
private int factorial(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; i++) {  
        result *= i;  
    }  
    return result;  
}
```

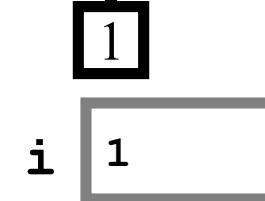
n

result

i

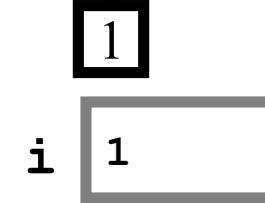
$0! = 1$

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + " ! = " + factorial(i));  
    }  
}
```



0 ! = 1

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + " ! = " + factorial(i));  
    }  
}
```



```
0 ! = 1  
1 ! = 1
```

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```

i 1

0! = 1  
1! = 1

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```

i 2

0! = 1  
1! = 1

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```

i 2

0! = 1  
1! = 1

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + " ! = " + factorial(i));  
    }  
}
```

i 2

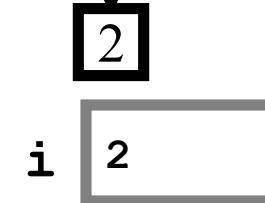
0 ! = 1  
1 ! = 1

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```

i 2

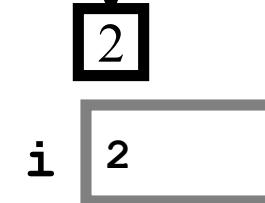
0! = 1  
1! = 1

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```



```
0! = 1  
1! = 1
```

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```



```
0! = 1  
1! = 1  
2! = 2
```

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```

i 2

```
0! = 1  
1! = 1  
2! = 2
```

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```

i 3

```
0! = 1  
1! = 1  
2! = 2
```

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```

i 3

```
0! = 1  
1! = 1  
2! = 2
```

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + " ! = " + factorial(i));  
    }  
}
```

i 3

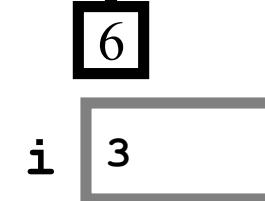
```
0 ! = 1  
1 ! = 1  
2 ! = 2
```

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```

i 3

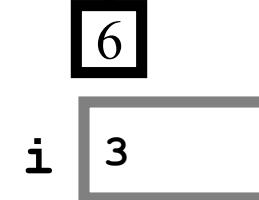
```
0! = 1  
1! = 1  
2! = 2
```

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```



```
0! = 1  
1! = 1  
2! = 2
```

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```



0! = 1  
1! = 1  
2! = 2  
3! = 6

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```

i 3

```
0! = 1  
1! = 1  
2! = 2  
3! = 6
```

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```

i 4

```
0! = 1  
1! = 1  
2! = 2  
3! = 6
```

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```

i 4

```
0! = 1  
1! = 1  
2! = 2  
3! = 6
```