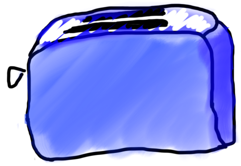
CS 106A, Lecture 8
Characters and Strings

suggested reading:

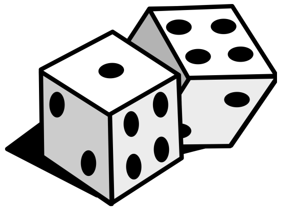
Java Ch. 8.1-8.4

Learning Goals

- Be able to confidently write and call methods that use parameters and return values.



- Be able to generate random values in your programs.



- Be able to use and manipulate **`char`**s.
- Be able to write string algorithms that operate on each character.

Plan For Today

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

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.

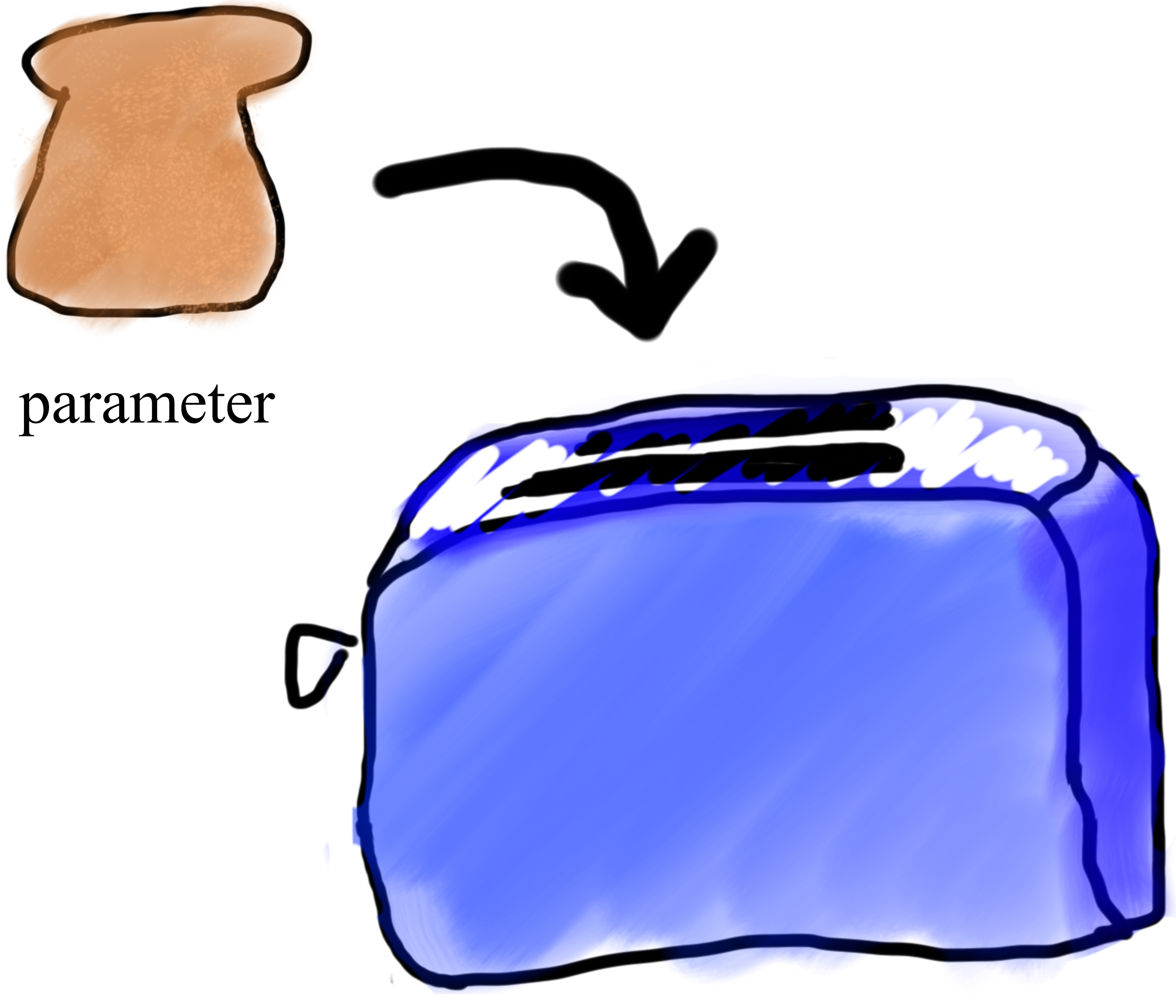
Return

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

Methods = Toasters

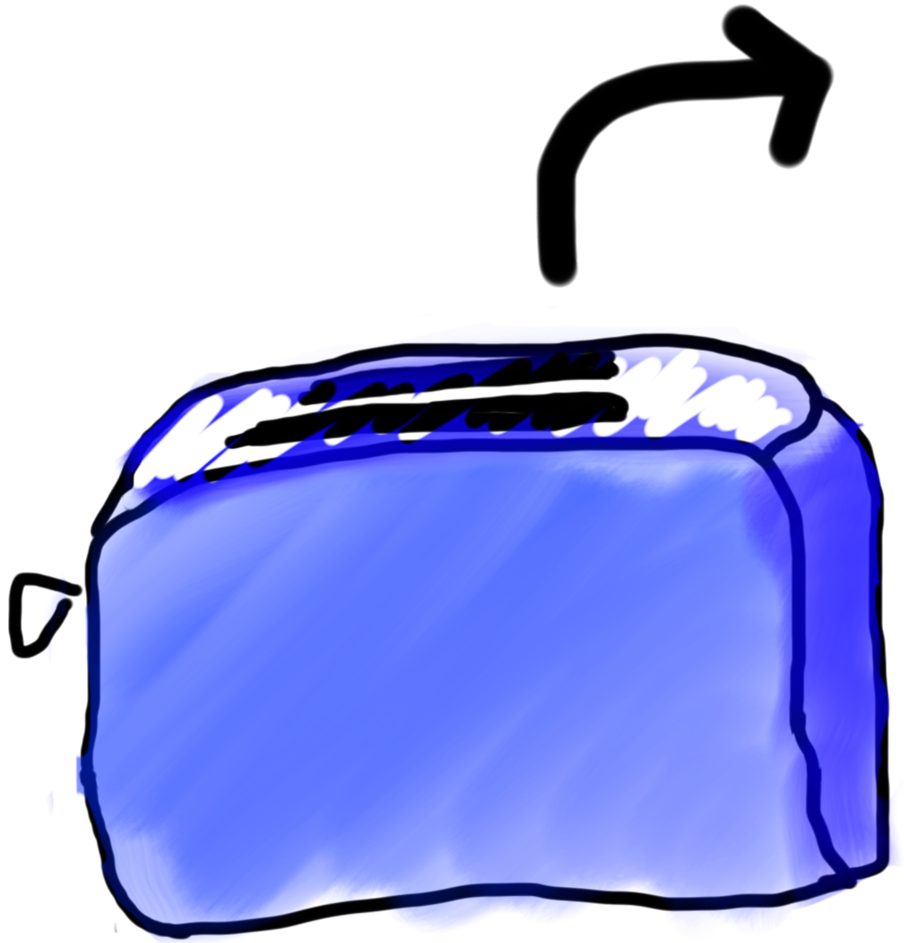
parameter

Methods = Toasters

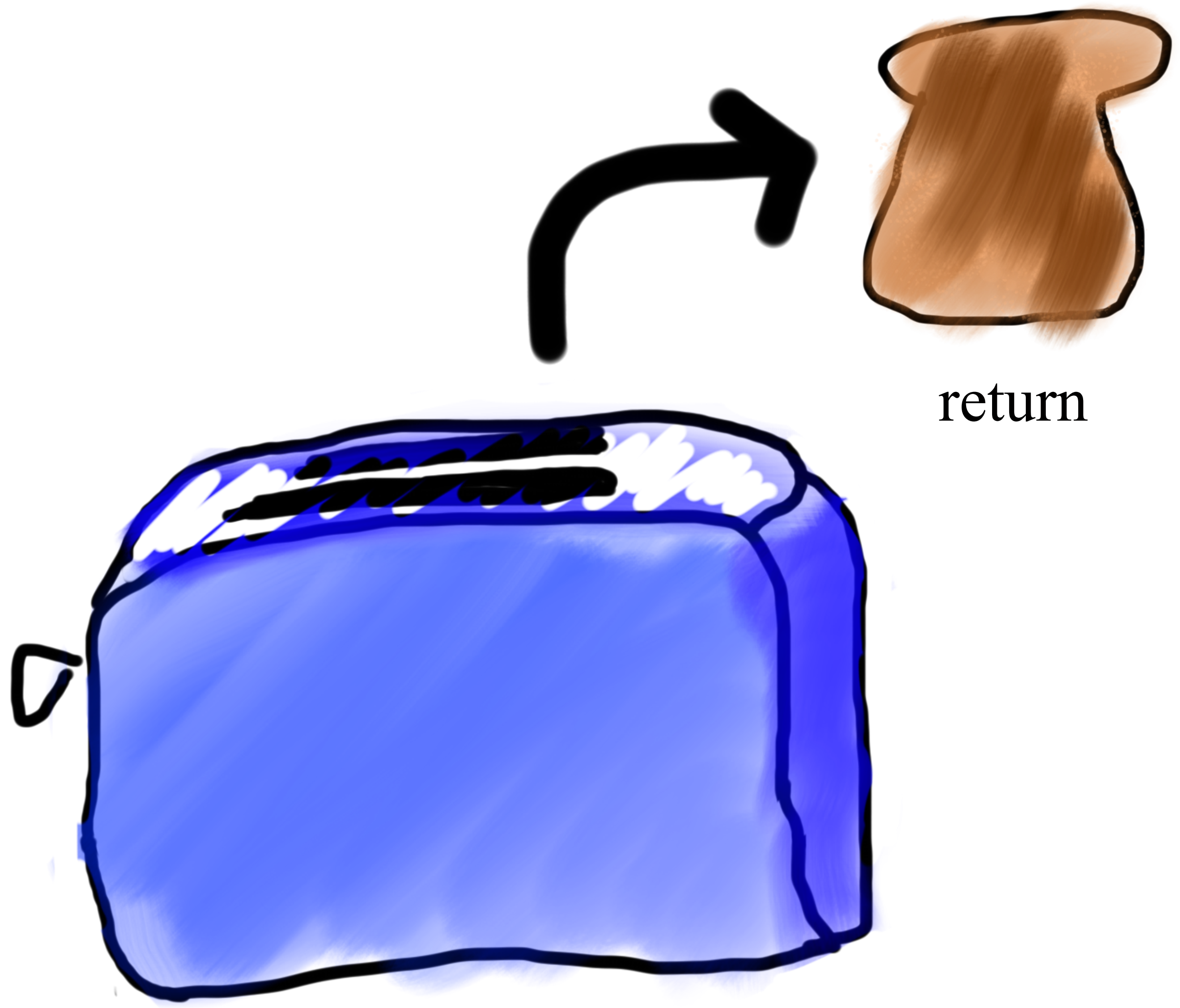


Methods = Toasters

Methods = Toasters



Methods = Toasters



Example: readInt

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

Example: readInt

We call readInt

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

```
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? ");
```

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? ");
```

Plan For Today

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

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

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

```
drawBox(10, 4);
```

drawBox

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

We call drawBox

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

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

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

Second parameter to drawBox

7 **4**

drawBox

7 4

drawBox

```
                                7                 4
private void drawBox(int width, int height) {
        // use width and height variables
        // to draw a box
}
```

drawBox

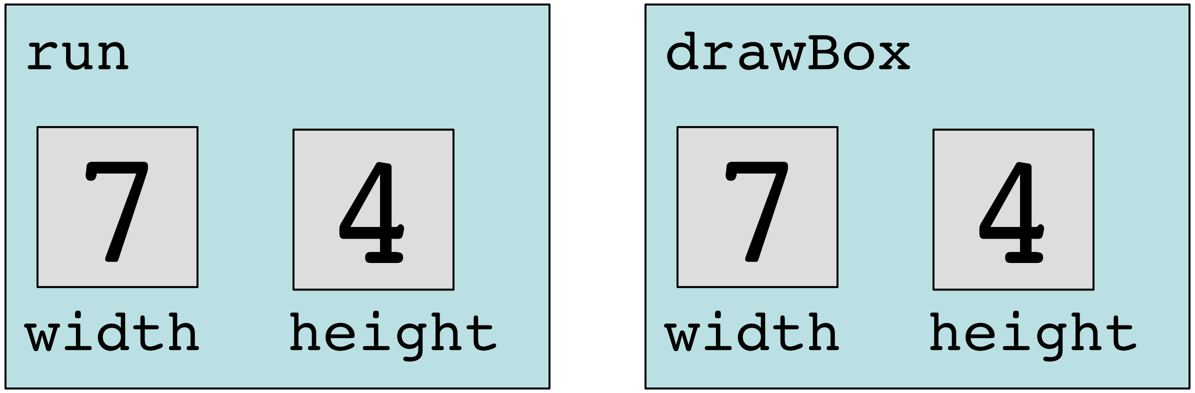
```
                                7             4
private void drawBox(int width, int height) {
     ...
     println(width);      // prints 7
     println(height);     // prints 4
     ...
}
```

Parameter Names

Parameter names do not affect program behavior.

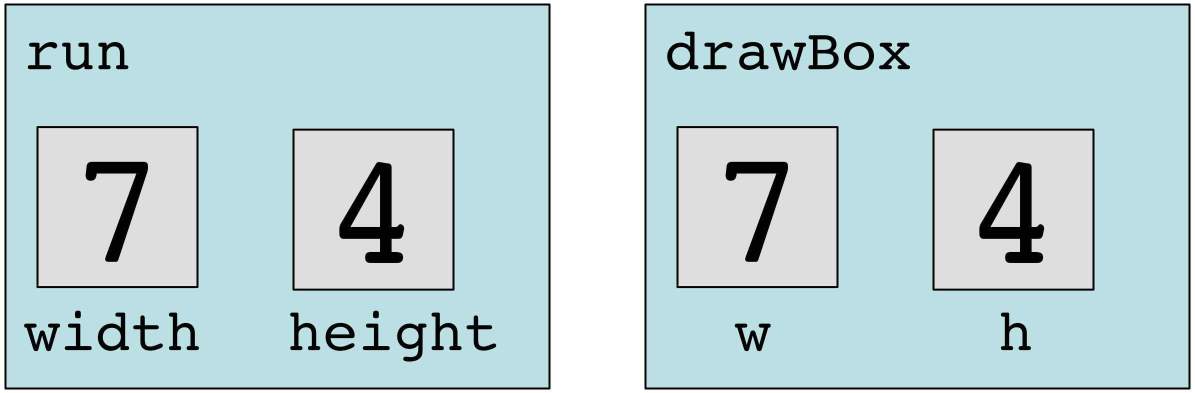
Parameter Names

```
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 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

```
public void run() {
    double cm = metersToCm(10);
    ...
}
```

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(10);
    ...
}
```

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

7

```
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

```
                          7
public void run() {
     double meters = readDouble("# meters? ”);
     ...

     double cm = metersToCm(meters);
     println(cm + " centimeters.");
}
                                    7
private double metersToCm(double meters) {
     double centimeters = meters * 100;
     return centimeters;
}
```

Return Example: metersToCm

```
public void run() {                 7
     double meters = readDouble("# meters? ");
     ...

     double cm = metersToCm(meters);
     println(cm + " centimeters.");
}
                                                7
private double metersToCm(double meters) {
     double centimeters = meters * 100;
     return ~~centimeters~~;
}              700
```

Return Example: metersToCm

```
public void run() {            7
      double meters = readDouble("# meters? ”);
      ...
                         700
      double cm = ~~metersToCm(meters)~~;
      println(cm + " centimeters.");
}
```

Return Values and Expressions

```
public void run() {          7
    double meters = readDouble("# meters? ”);
    println(metersToCm(meters) + " cm.");
}

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

Return Values and Expressions

```
public void run() {                 7
     double meters = readDouble("# meters? ");
     println(~~metersToCm(meters)~~ + " cm.");
}                     700

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

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

Buggy Example!

```
public void run() {          7
      double meters = readDouble("# meters? ");
      ...

      metersToCm(meters); // Does nothing!
      ...
}
```

Buggy Example!

```
public void run() {            7
      double meters = readDouble("# meters? ");
      ...
              700
      ~~metersToCm(meters);~~ // Does nothing!
      ...
}
```

Return Stops Method Execution

```
private int max(int num1, int num2) {
    if(num1 >= num2) {
        return num1;
    }
    return num2; // here only if num1 < num2
}
```

Returning Booleans

```
private boolean isEven(int number) {
      return number % 2 == 0;
}
```

Returning Booleans

56

```
private boolean isEven(int number) {
    return number % 2 == 0;
}
```

Returning Booleans

```
                                                56
private boolean isEven(int number) {
     return ~~number % 2 == 0~~;
}                true
```

Returning Booleans

```
private boolean isEven(int number) {
	return number % 2 == 0;
}




// Example
public void run() {
	if (isEven(2)) {
		...
	}
}
```

Returning Booleans

```java
private boolean isDivisibleBy(int a, int b) {
    return a % b == 0;
}
```

Returning Booleans

```
                                     24      9
private boolean isDivisibleBy(int a, int b) {
   return a % b == 0;
}
```

Returning Booleans

```
                                     24      9
private boolean isDivisibleBy(int a, int b) {
    return ~~a % b == 0~~;
}           false
```

Returning Booleans

```
private boolean isDivisibleBy(int a, int b) {
   return a % b == 0;
}



// Example
public void run() {
     if (isDivisibleBy(4, 2)) {
          ...
     }
}
```

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

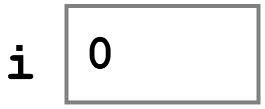
i

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

$\boxed{\texttt{int i = 0}}$

i $\boxed{0}$

```
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));
    }
}
```

$$\begin{array}{c|c|}\cline{2-2} i & 0 \\ \cline{2-2}\end{array}$$

```
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
0	1	

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

n $\boxed{0}$ result $\boxed{1}$ i $\boxed{1}$

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

n $\boxed{0}$ result $\boxed{1}$ i $\boxed{1}$

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

n $\boxed{0}$ result $\boxed{1}$ i $\boxed{1}$

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

$$\underbrace{\texttt{factorial(i)}}_{\boxed{1}}$$

i $\boxed{0}$

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

$\underbrace{\texttt{factorial(i)}}_{\boxed{1}}$

i $\boxed{0}$

```
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 1 result 1 i

```
0! = 1
```

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

n `1`   result `1`   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 1   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 1        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 `1` i `2`

```
0! = 1
```

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

n 1 result 1 i 2

```
0! = 1
```

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

n 1   result 1   i 2

```
0! = 1
```

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

$\underbrace{\texttt{factorial(i)}}_{\boxed{1}}$

i $\boxed{1}$

```
0! = 1
```

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

$\underbrace{\texttt{factorial(i)}}_{\boxed{1}}$

i $\boxed{1}$

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

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

i $\boxed{2}$

```
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 + "! = " + $\boxed{factorial(i)}$);
    }
}
```

```
i | 2
```

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

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

$\underbrace{\texttt{factorial(i)}}_{\boxed{2}}$

i $\boxed{2}$

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

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

$\underbrace{\texttt{factorial(i)}}_{\boxed{2}}$

i $\boxed{2}$

```
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 $\boxed{3}$

```
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 $\boxed{3}$

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

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

$$\underbrace{\texttt{factorial(i)}}_{\boxed{6}}$$

i $\boxed{3}$

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

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

$\underbrace{\texttt{factorial(i)}}_{\boxed{6}}$

i $\boxed{3}$

```
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
```

Plan For Today

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

RandomGenerator

- `import acm.util.*;`

Method	Description
`RandomGenerator.getInstance().nextInt(`***`min, max`***`)`	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 **`low`** ≤ *d* < **`high`**.
`double nextDouble()` Returns a random **`double`** *d* in the range 0 ≤ *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 ≤ **`p`** ≤ 1.
`Color nextColor()` Returns a random color.

Extra: Dice exercise

RollTwoDice

- Write a console program **`RollTwoDice`** that repeatedly rolls two 6-sided dice until they arrive at a given desired sum.

```
Desired sum? 9
3 and 4 = 7
2 and 1 = 3
5 and 5 = 10
6 and 2 = 8
6 and 5 = 11
4 and 5 = 9
```

Plan For Today

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

Text Processing

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

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

Char Math!

Under the hood, Java represents each **`char`** as an *integer* (its “ASCII value”), which we can take advantage of.

```
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!

Under the hood, Java represents each **`char`** as an *integer* (its “ASCII value”), which we can take advantage of.

```
// 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 **`int`**s and **`char`**s, 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 *cast it* to that type.

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

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 **`char`**s.

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


char plus = '+';
if (Character.isLetter(plus)) {
      ...
}
```

Character Methods

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

Remember: these **return** the new char, they cannot modify the parameter!

Character Methods

Remember to always save the return value of Character methods!

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

Plan For Today

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

Strings

A **`String`** is a variable type representing sequences of characters.

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

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
character	`'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);
```

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 worldExlm = 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.

```java
String str = "Hello, world!";
String worldExlm = 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
`s.length()`	number of characters in this string
`s.charAt(`***`index`***`)`	char at the given index
`s.indexOf(`***`str`***`)`	index where the start of the given string appears in this string (-1 if not found)
`s.substring(`***`index1, index2`***`)` or `s.substring(`***`index1`***`)`	the characters in this string from *index1* (inclusive) to *index2* (exclusive); if *index2* is omitted, goes until end
`s.toLowerCase()`	a new string with all lowercase letters
`s.toUpperCase()`	a new string with all uppercase letters

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

```
String className = "CS 106A yay!";
println(className.length());   // 12
```

Strings are Immutable

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

```
// Cannot change individual chars in the string
String typo = "Hello, warld!”;
```

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

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:
String greeting = "Hello!”;
if (greeting.equals("Hello!")) {
      ...
}
```

Always use .equals instead of == and !=

Comparing Strings

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

Looping Over Strings

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

```
// Capitalizes each letter
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