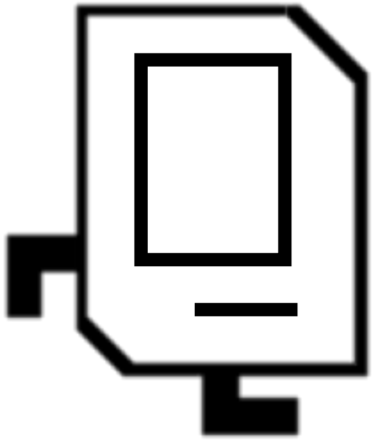
CS106A Review Session

Nick Troccoli

Topic List

- Karel
- Java constructs
- Graphics + Animation
- Memory (Pass-by-reference vs. pass by value)
- Event-driven programming
- Characters and Strings

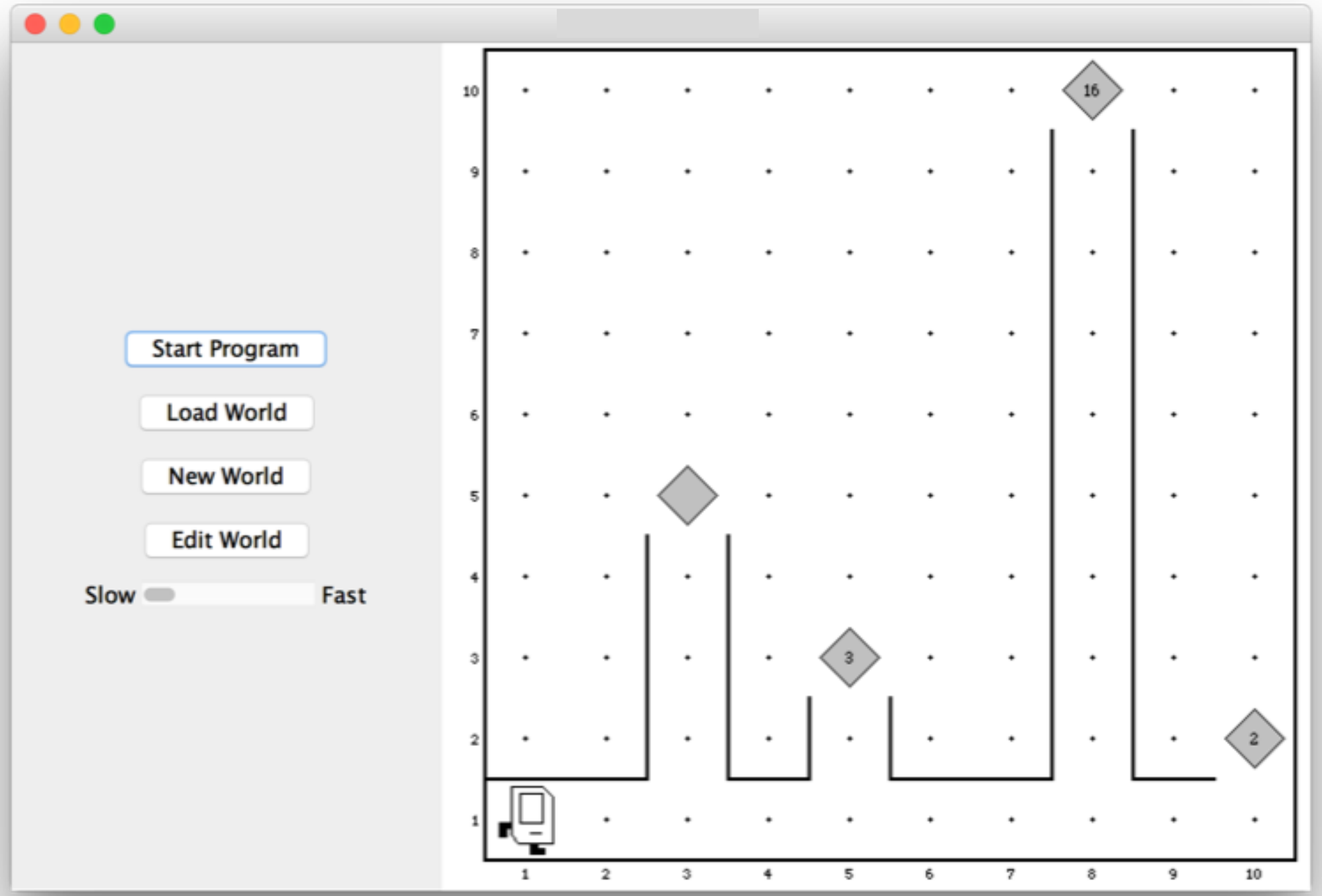
Karel



Karel the Robot

- Tips:
  - Pseudocode first
  - Decompose the problem
  - Might be limitations on constructs
    - E.g. no Java features (variables, break, etc.)

Mail Karel



Start Program
Load World
New World
Edit World
Slow
Fast
16
3
2

Mail Karel

Karel is in a world with walkways to houses that have mail to pick up. Karel should go to every house in order, go up the walkway and take all the mail (beepers). House walkways can be any distance apart, and have guide walls on the left and right up to the mailbox.

Challenge: solve this before proceeding to solution!

Mail Karel

Loop:

```
- if there's a house:
        pick up mail
- if front is clear:
        move
```

Pick up mail:

```
- traverse walkway
- take mail
- traverse walkway
```

Mail Karel

```
public void run() {
      while (frontIsClear()) {
            if (leftIsClear()) {
                  pickUpMail();
            }
            if (frontIsClear()) {
                  move();
            }
      }
      if (leftIsClear()) {    // maybe house on the last square!
            pickUpMail();
      }
}
```

Mail Karel

```
private void pickUpMail() {
     turnLeft();
     traverseWalkway();
     takeMail();
     turnAround();
     traverseWalkway();
     turnLeft();
}
```

Mail Karel

```
private void traverseWalkway() {
      move();
      while (leftIsBlocked() && rightIsBlocked()) {
            move();
      }
}
```

Mail Karel

```
private void takeMail() {
     while (beepersPresent()) {
          pickBeeper();
     }
}
```

Java Constructs

Java Constructs

- **Variable types**: primitives (int, double,...) + objects (GRect, Goval,...)
- **Control statements**: if, while, for, switch
  - What is each useful for?
- **Methods**
  - **Parameters**
  - **Return**

Java Constructs

- **Variable types**: primitives (int, double,…) + objects (GRect, Goval,…)
- **Control statements**: if, while, for, switch
  - What is each useful for?
- **Methods**
  - **Parameters**
  - **Return**

For or While?

WHILE	• Read in user input until you hit the SENTINEL
FOR	• Iterate through a string
WHILE	• Move Karel to a wall
FOR	• Put down 8 beepers

Java Constructs

- **Variable types**: primitives (int, double,…) + objects (GRect, Goval,…)
- **Control statements**: if, while, for, switch
  - What is each useful for?
- **Methods**
  - **Parameters**
  - **Return**

Java Constructs - Methods

Methods let you define custom Java commands.

Java Constructs - Methods

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

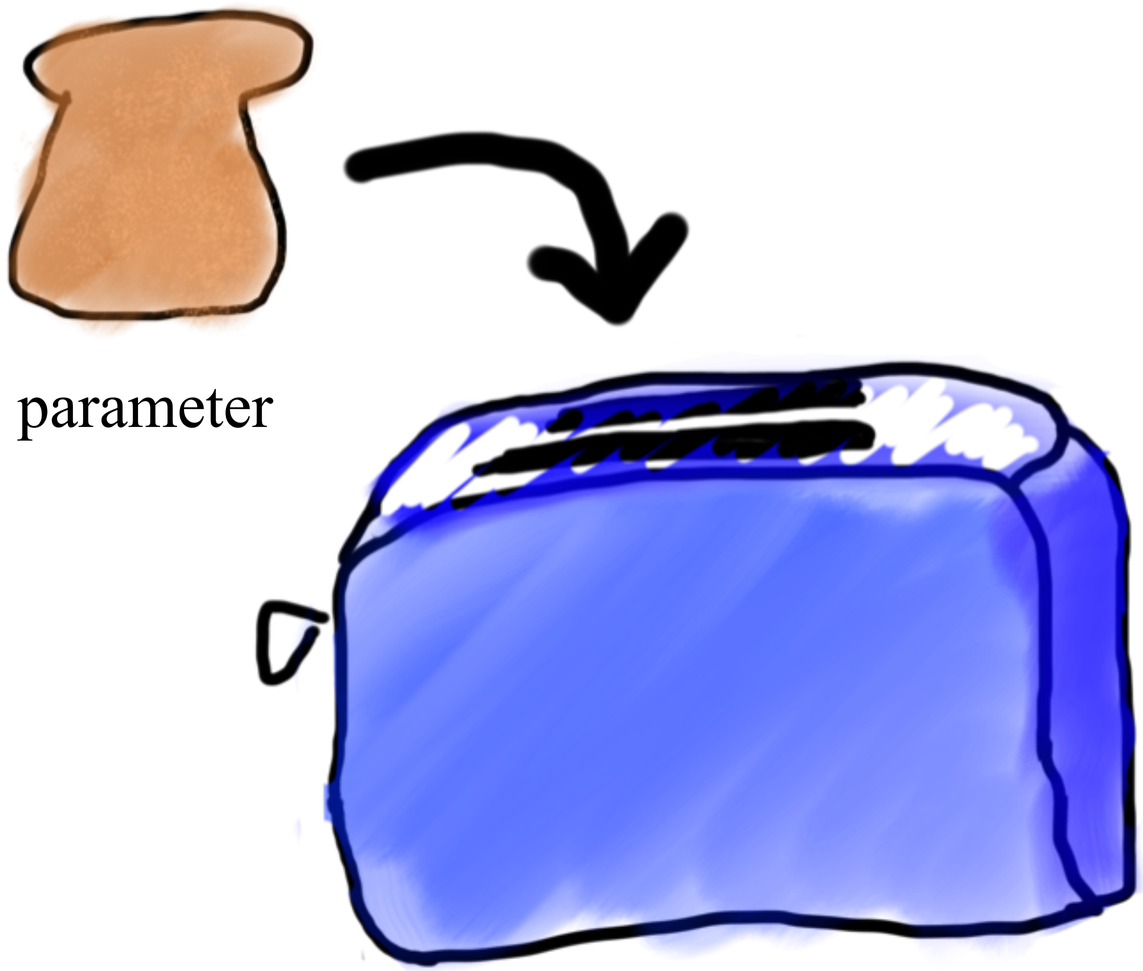
Java Constructs: Methods

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

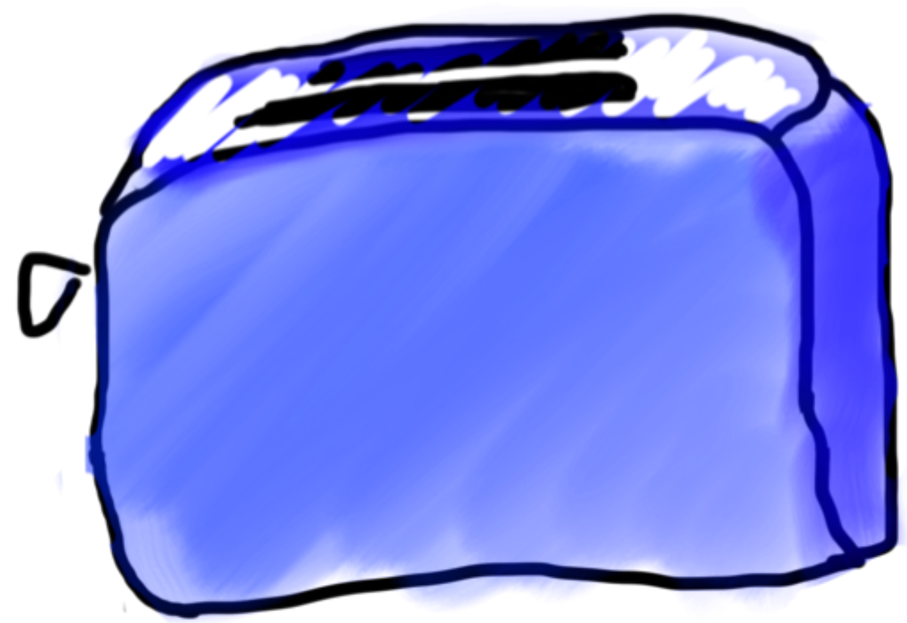
Java Constructs - Methods

parameter

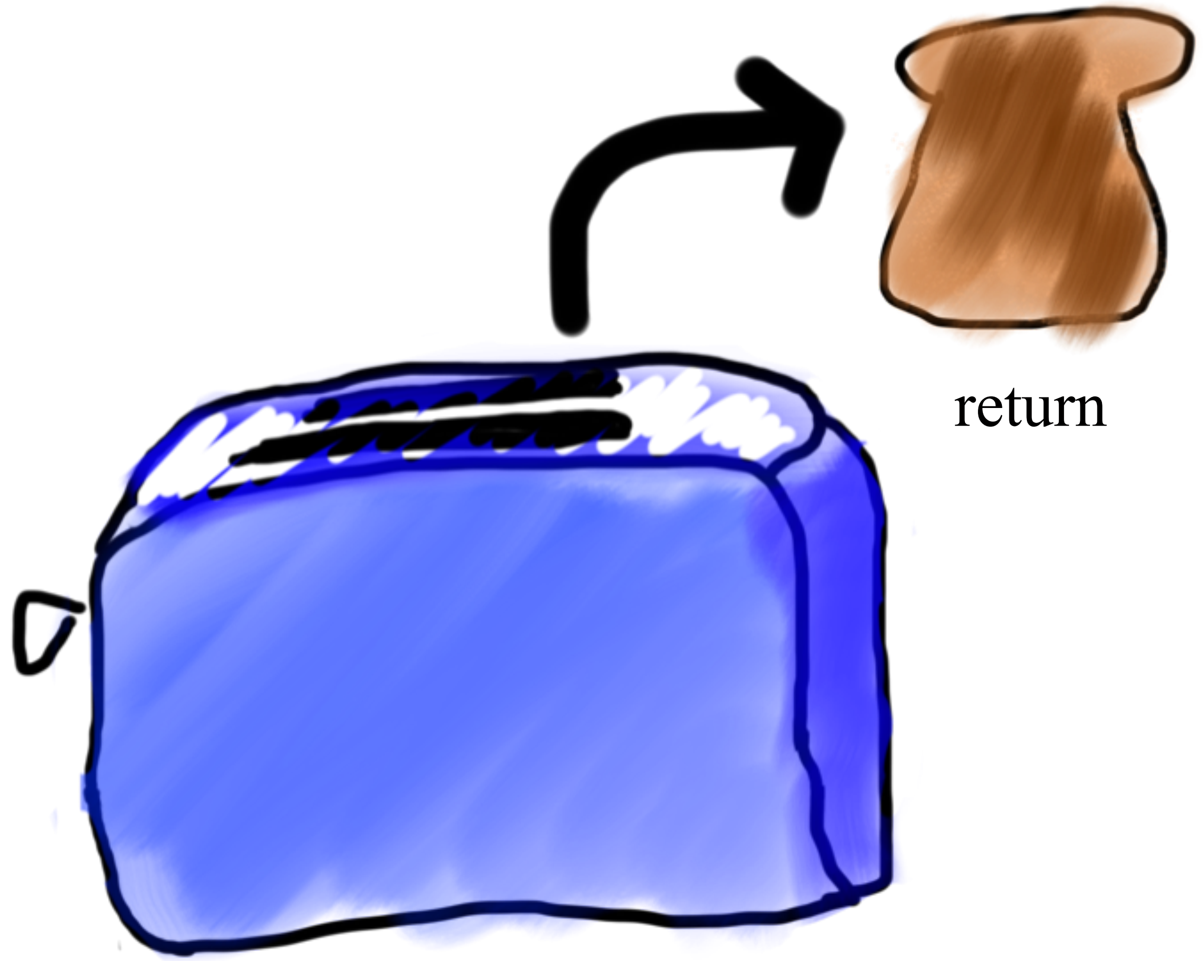
Java Constructs - Methods



Java Constructs - Methods



Java Constructs - Methods



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? ");
```

Example: readInt

- **Variable types**: primitives (int, double,…) + objects (GRect, Goval,…)
- **Control statements**: if, while, for, switch
  - What is each useful for?
- **Methods**
  - **Parameters**
  - **Return**

Parameters: drawBlueRect

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

```
private void drawBlueRect(int width, int height) {
     // use width and height variables
     // to draw a rect at 0, 0
}
```

Parameters: drawBlueRect

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

```
private void drawBlueRect(int width, int height) {
     // use width and height variables
     // to draw a rect at 0, 0
}
```

Parameters: drawBlueRect

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

```
private void drawBlueRect(int width, int height) {
      // use width and height variables
      // to draw a rect at 0, 0
}
```

Parameters: drawBlueRect

We call drawBlueRect

We give drawBlueRect some information (the size of the rect we want)

```
drawBlueRect(50, 20);
```

Parameters: drawBlueRect

```
int width = ...   70
int height = ... 40
...
```

```
               70      40
drawBlueRect(~~width~~, ~~height~~);
```

Parameters: drawBlueRect

```
int width = ...  70
int height = ... 40
...




drawBlueRect(70, 40);
```

Parameters: drawBlueRect

First parameter to drawBlueRect

Second parameter to drawBlueRect

```
drawBlueRect(70, 40);
```

Parameters: drawBlueRect

```
                                  70         40
private void drawBlueRect(int width, int height) {
     // use width and height variables
     // to draw a rect at 0, 0
}
```

Parameters: drawBlueRect

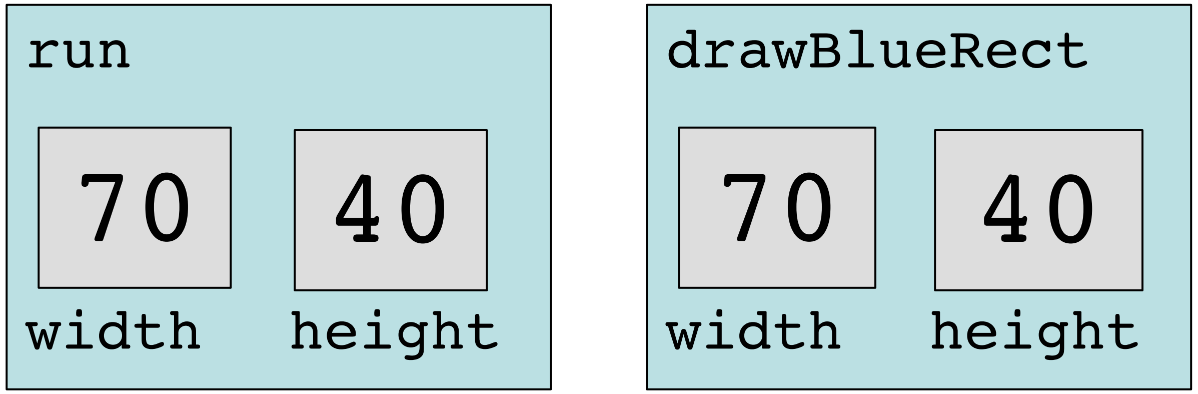
```
                                          70           40
private void drawBlueRect(int width, int height) {
    GRect rect = new GRect(width, height); // 70x40
    ...
}
```

Parameters: drawBlueRect

Parameter names do not affect program behavior.

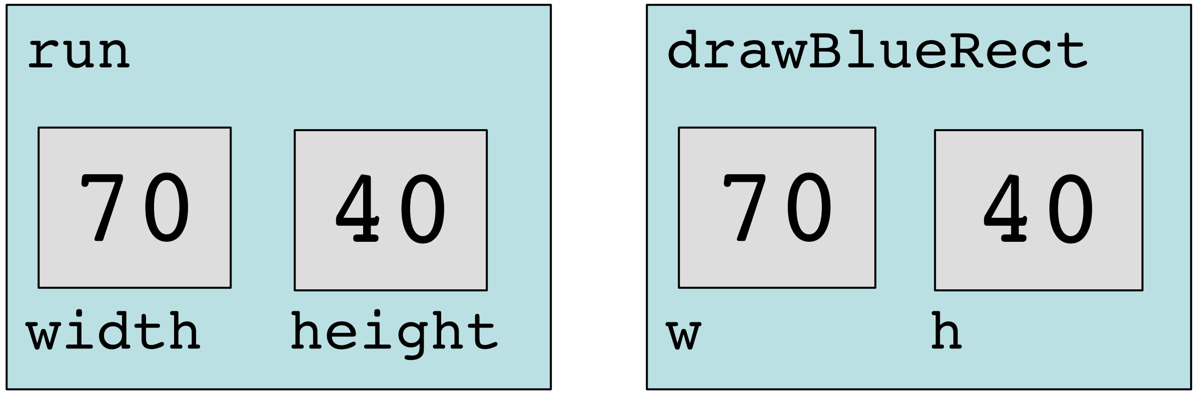
Parameters: drawBlueRect

```
public void run() {
     int width = ...      70
     int height = ...     40
     drawBlueRect(width, height);
}

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



Parameters: drawBlueRect

```
public void run() {
     int width = ...        70
     int height = ...       40
     drawBlueRect(width, height);
}

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



Java Constructs

- **Variable types**: primitives (int, double,…) + objects (GRect, Goval,…)
- **Control statements**: if, while, for, switch
  - What is each useful for?
- **Methods**
  - **Parameters**
  - **Return**

Return

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

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

Return

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

$\uparrow$

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

Return

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

Return

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

```
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

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

     double cm = metersToCm(meters);
     println(cm + " centimeters.");
}

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

Return

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

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

Return

```
                            7
public void run() {
    double meters = readDouble("# meters? ");
    ...
                                    7
    double cm = metersToCm(meters);
    println(cm + " centimeters.");
}
                                         7
private double metersToCm(double meters) {
    double centimeters = meters * 100;
    return ~~centimeters~~;
}                 700
```

Return

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

Return

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

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

Return

```
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.*

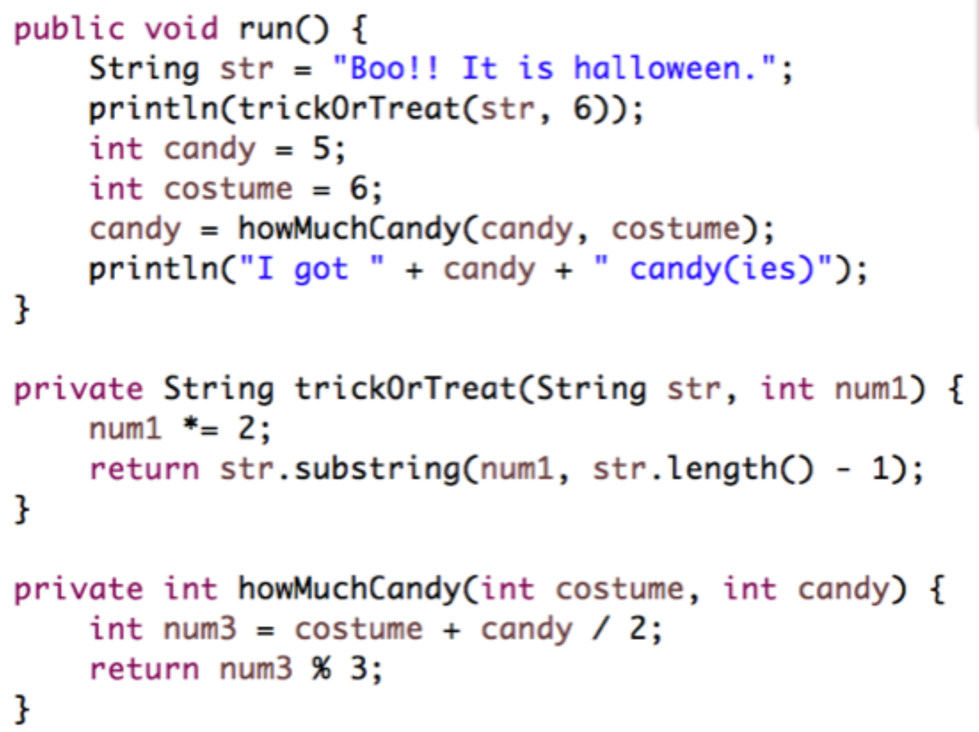
Return

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

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

Return

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

Program Trace



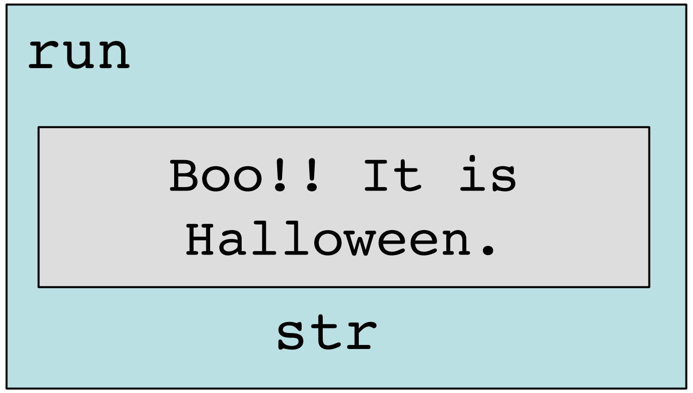
```
public void run() {
    String str = "Boo!! It is halloween.";
    println(trickOrTreat(str, 6));
    int candy = 5;
    int costume = 6;
    candy = howMuchCandy(candy, costume);
    println("I got " + candy + " candy(ies)");
}

private String trickOrTreat(String str, int num1) {
    num1 *= 2;
    return str.substring(num1, str.length() - 1);
}

private int howMuchCandy(int costume, int candy) {
    int num3 = costume + candy / 2;
    return num3 % 3;
}
```

Challenge: find output of this before proceeding!

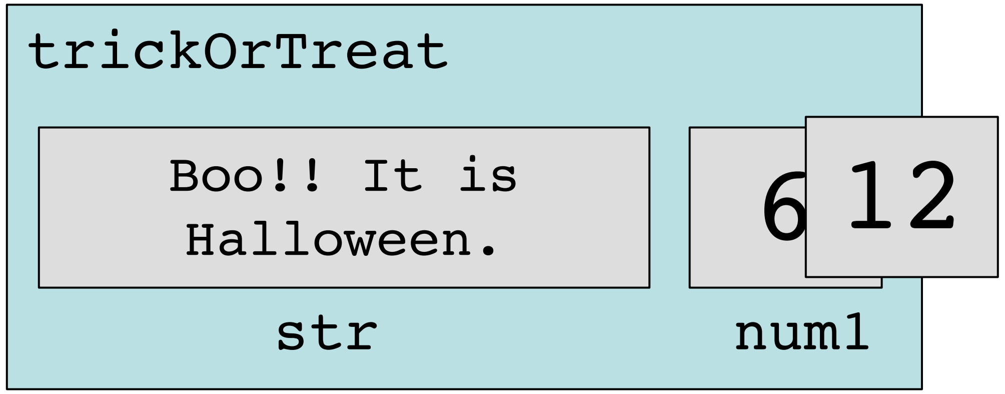
(Dug up from an old program – do not write code like this at home! :))

Program Trace

```
public void run() {
     String str = “Boo!! It is halloween.”;
     println(trickOrTreat(str, 6));
     ...
}
```



Program Trace



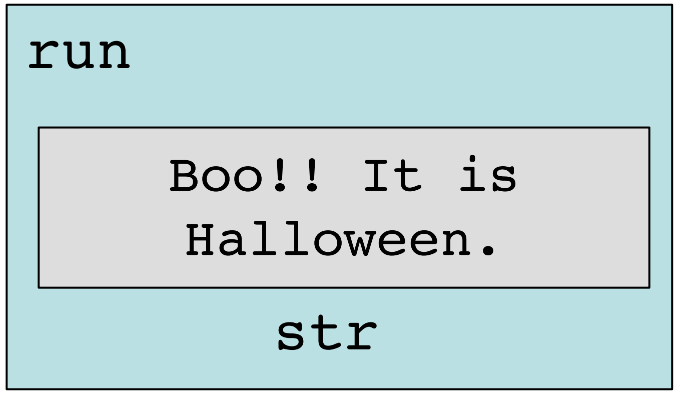
```
private String trickOrTreat(String str, int num1) {
      num1 *= 2; // 12
      return str.substring(num1, str.length() – 1);
}
```

12

21

Program Trace

```
public void run() {
     String str = "Boo!! It is halloween.";
     println(trickOrTreat(str, 6));
     ...
}
```

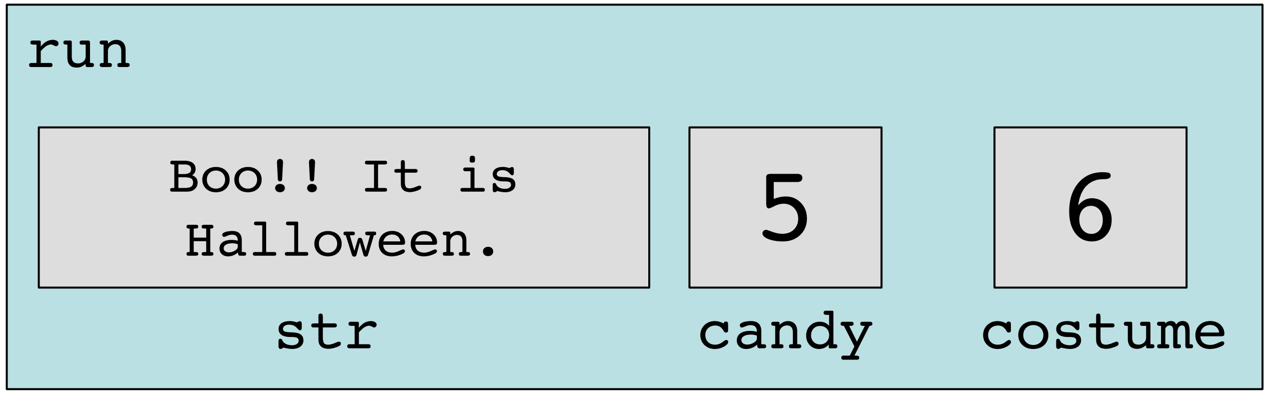


```
halloween
```

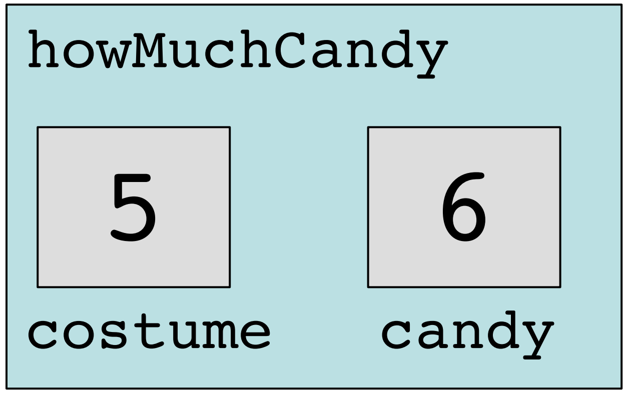
(Console)

Program Trace

```
public void run() {
    ...
    int candy = 5;
    int costume = 6;           5       6
    candy = howMuchCandy(candy, costume);
    println("I got " + candy + " candy(ies)");
}
```

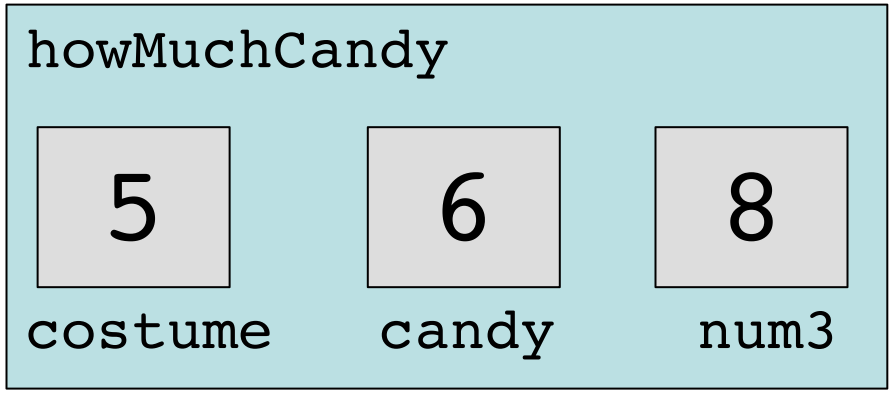


Program Trace



```
private int howMuchCandy(int costume, int candy) {
      int num3 = costume + candy / 2;
      return num3 % 3;
}
```

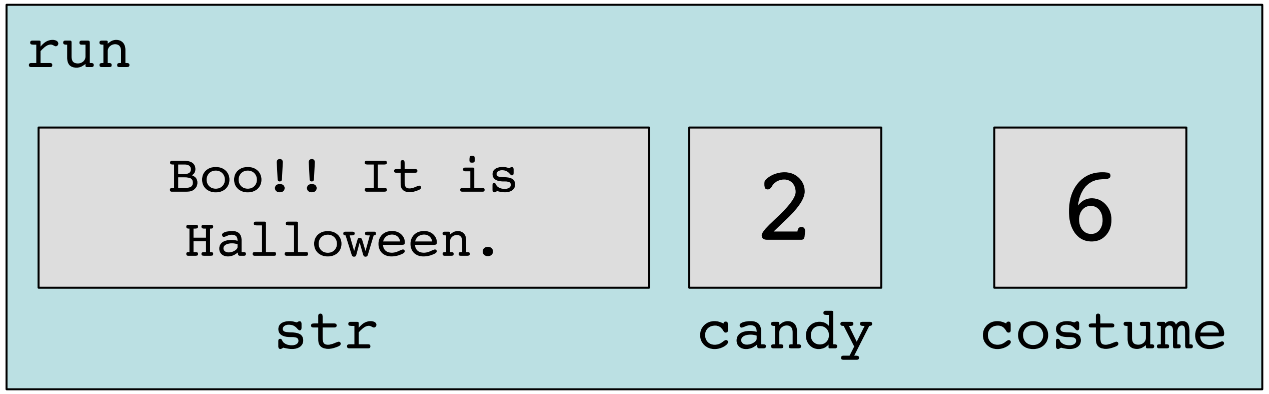
Program Trace



```
private int howMuchCandy(int costume, int candy) {
      int num3 = costume + candy / 2;  // 8
      return num3 % 3;  // 2
}
```

Program Trace

```
public void run() {
     ...
     int candy = 5;
     int costume = 6;
     candy = howMuchCandy(candy, costume);
     println(“I got ” + candy + “ candy(ies)”);
}
```



(Console)

```
halloween
I got 2 candy(ies)
```

Program Trace

- **Tricky spots:** precedence, parameter/variable names…
- Draw pictures! / Label variable values

Graphics and Animation

Graphics

- Look at lecture slides for lists of different GObject types and their methods
- Remember: the x and y of GRect, GOval, etc. is their **upper left corner**, but the x and y of GLabel is its **leftmost baseline coordinate**.
- Remember: a label’s height is gotten from **getAscent**.

Animation

Standard format for animation code:

(see Event-Driven Programming for example program)

```
while (CONDITION) {
     updateGraphics();
     performChecks();
     pause(PAUSE_TIME);
}
```

Memory

Memory

- **Stack and heap**
  - **Stack** is where local variables live
  - **Heap** is where objects live
- When you make an object, the local variable (what you named it) is a box that stores an **address** where the object actually lives.
- When you make a primitive, the local variable is a box that stores the **actual value**.

Memory

- **==** is dangerous because it compares what's in the **variable boxes!**
  - For primitives, ok
  - For objects, compares their addresses! So only true if they're the exact same object living in the exact same place.

Memory

- **Parameters:** when you pass a parameter, Java passes a copy of whatever is in the variable's box.
  - For primitives – a copy of their **value**
  - For objects – a copy of their **address!** So there's still only 1 object version

Memory

```
public void run() {
     GRect rect = new GRect(0,0,50,50);
     fillBlue(rect);
     add(rect);      // rect is blue!
}

private void fillBlue(GRect myRect) {
     myRect.setFilled(true);
     myRect.setColor(Color.BLUE);
}
```

Memory

```
public void run() {
     int x = 2;
     addTwo(x);
     println(x);     // x is still 2!
}

private void addTwo(int x) {
     x += 2;         // this modifies addTwo's COPY!
}
```

Memory

```
public void run() {
     int x = 2;
     x = addTwo(x);
     println(x);     // x is still 2!
}
private int addTwo(int x) {
     x += 2;         // this modifies addTwo's COPY!
     return x;
}
```

Event-Driven Programming

Event-Driven Programming

- **Example:** mouse events
- **Two** ways for Java to execute your code: from run() and from event handler (mouseClicked, mouseMoved, etc.).
- These programs are **asynchronous** – code is not run in order any more, since you don’t know when the user will interact with your program!

Event-Driven Programming

1. Sign up for notifications for mouse events
2. Implement the method corresponding to what event you care about (e.g. **mousePressed**, **mouseMoved**).
3. Java will call that method whenever the corresponding event occurs.

Color-Changing Square

```
public class ColorChangingSquare extends GraphicsProgram {

   /* Size of the square in pixels */
   private static final int SQUARE_SIZE = 100;

   /* Pause time in milliseconds */
   private static final int PAUSE_TIME = 1000;


   public void run() {
      GRect square = new GRect(SQUARE_SIZE, SQUARE_SIZE);
      square.setFilled(true);
      add(square, (getWidth() - SQUARE_SIZE) / 2,
                  (getHeight() - SQUARE_SIZE) / 2);

      /* Note: we meant to have this infinite loop */
      while (true) {
         square.setColor(rgen.nextColor());
         pause(PAUSE_TIME);
      }
   }

/* Private instance variables */
   private RandomGenerator rgen = RandomGenerator.getInstance();
}
```

Color-Changing Shape

Let’s write a program like “random color-changing square”, but when you click the shape changes between a square and a circle.

The shapes should always be the same random color, even if you click in between colors changing.

Challenge: solve this before proceeding to solution!

Color-Changing Shape

Modifications:

- Have a **square and circle** at the same time
- Change the colors of **both** every PAUSE_TIME
- On click, add one and remove the other

Constants and Instance Variables

```
private static final int SQUARE_SIZE = 100;
private static final int PAUSE_TIME = 100;
private GRect square;
private GOval circle;
private boolean isShowingSquare = true;

private RandomGenerator rgen =
RandomGenerator.getInstance();
```

Color-Changing Shape

```java
public void run() {
	setup();
	while (true) {
		Color c = rgen.nextColor();
		square.setColor(c);
		circle.setColor(c);
		pause(PAUSE_TIME);
	}
}
```

Color-Changing Shape

```
private void setup() {
	double x = (getWidth() – SQUARE_SIZE) / 2.0;
	double y = (getHeight() – SQUARE_SIZE) / 2.0;
	square = new GRect(x, y, SQUARE_SIZE, SQUARE_SIZE);
	circle = new GOval, y, SQUARE_SIZE, SQUARE_SIZE);
	square.setFilled(true);
	circle.setFilled(true);
	add(square);    // only add square!
	addMouseListeners();
}
```

Color-Changing Shape

```
public void mouseClicked(MouseEvent e) {
     if (isShowingSquare) {
          remove(square);
          add(circle);
     } else {
          remove(circle);
          add(square);
     }
     isShowingSquare = !isShowingSquare; // flip boolean
}
```

Characters and Strings

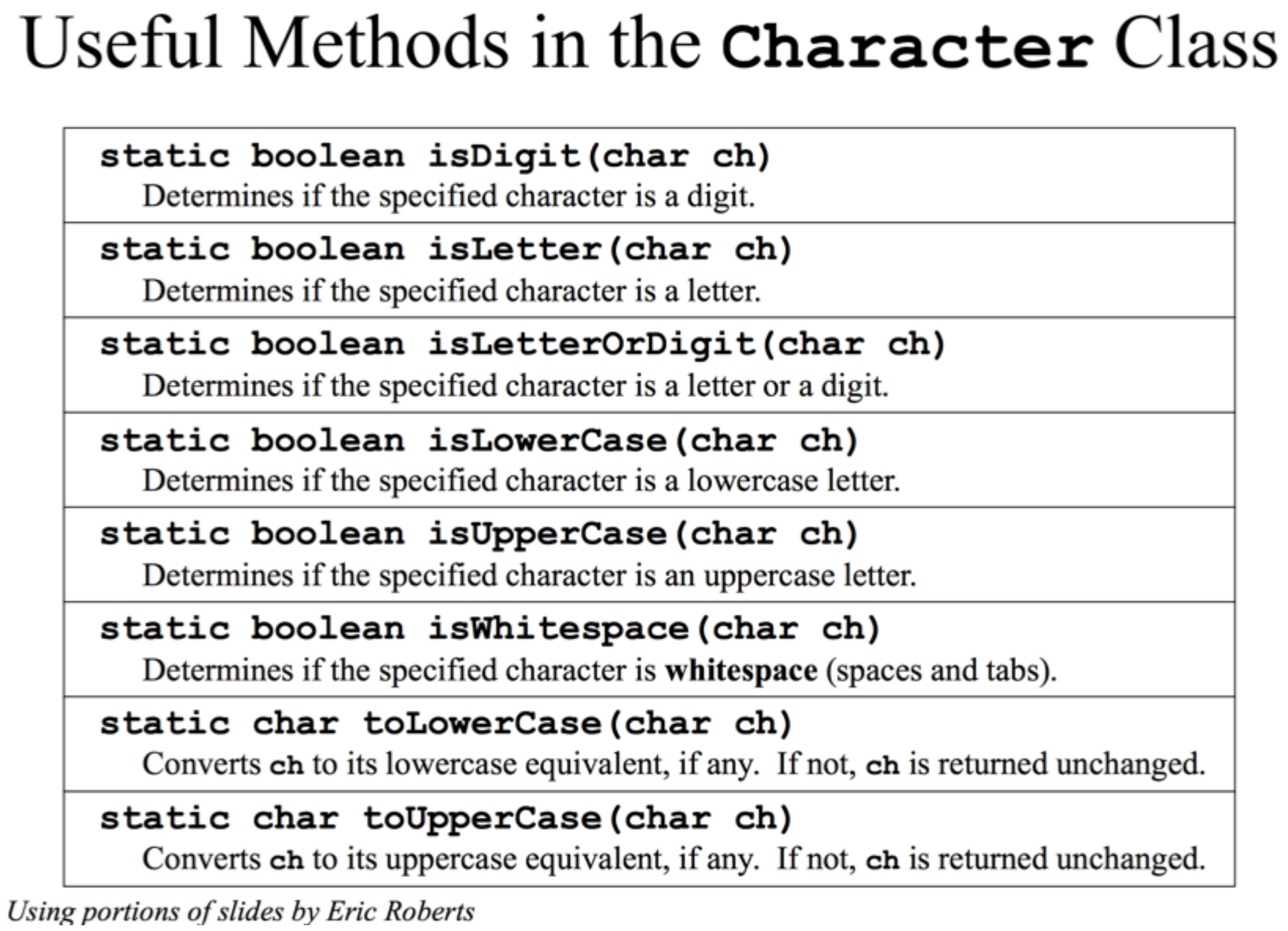
Characters and Strings

- A **char** is a primitive type that represents a single letter, digit, or symbol. Uses single quotes (‘’).
- Computers represent **chars** as numbers under the hood (ASCII encoding scheme).
- A string is an immutable object that represents a sequence of characters. Uses double quotes (“”).

Characters

```
char uppercaseA = ‘A’;

// We need to cast to a char so the type on the right matches
// the type on the left (char arithmetic defaults to int)
char uppercaseB = (char)(uppercaseA + 1);

int lettersInAlphabet = ‘Z’ – ‘A’ + 1;
// equivalent: ‘z’ – ‘a’ + 1
// A to Z and a to z are sequential numbers.
```

Characters



Useful Methods in the **`Character`** Class

`static boolean isDigit(char ch)`	Determines if the specified character is a digit.
`static boolean isLetter(char ch)`	Determines if the specified character is a letter.
`static boolean isLetterOrDigit(char ch)`	Determines if the specified character is a letter or a digit.
`static boolean isLowerCase(char ch)`	Determines if the specified character is a lowercase letter.
`static boolean isUpperCase(char ch)`	Determines if the specified character is an uppercase letter.
`static boolean isWhitespace(char ch)`	Determines if the specified character is **whitespace** (spaces and tabs).
`static char toLowerCase(char ch)`	Converts **`ch`** to its lowercase equivalent, if any. If not, **`ch`** is returned unchanged.
`static char toUpperCase(char ch)`	Converts **`ch`** to its uppercase equivalent, if any. If not, **`ch`** is returned unchanged.

Using portions of slides by Eric Roberts

Characters

- **Note:** chars are <u>primitives</u>. This means we can't call methods on them!
- Instead we use the **Character** class and call methods on it. We pass in the character of interest as a <u>parameter</u>.
- These methods *<u>do not change the char</u>*! They return a modified char.

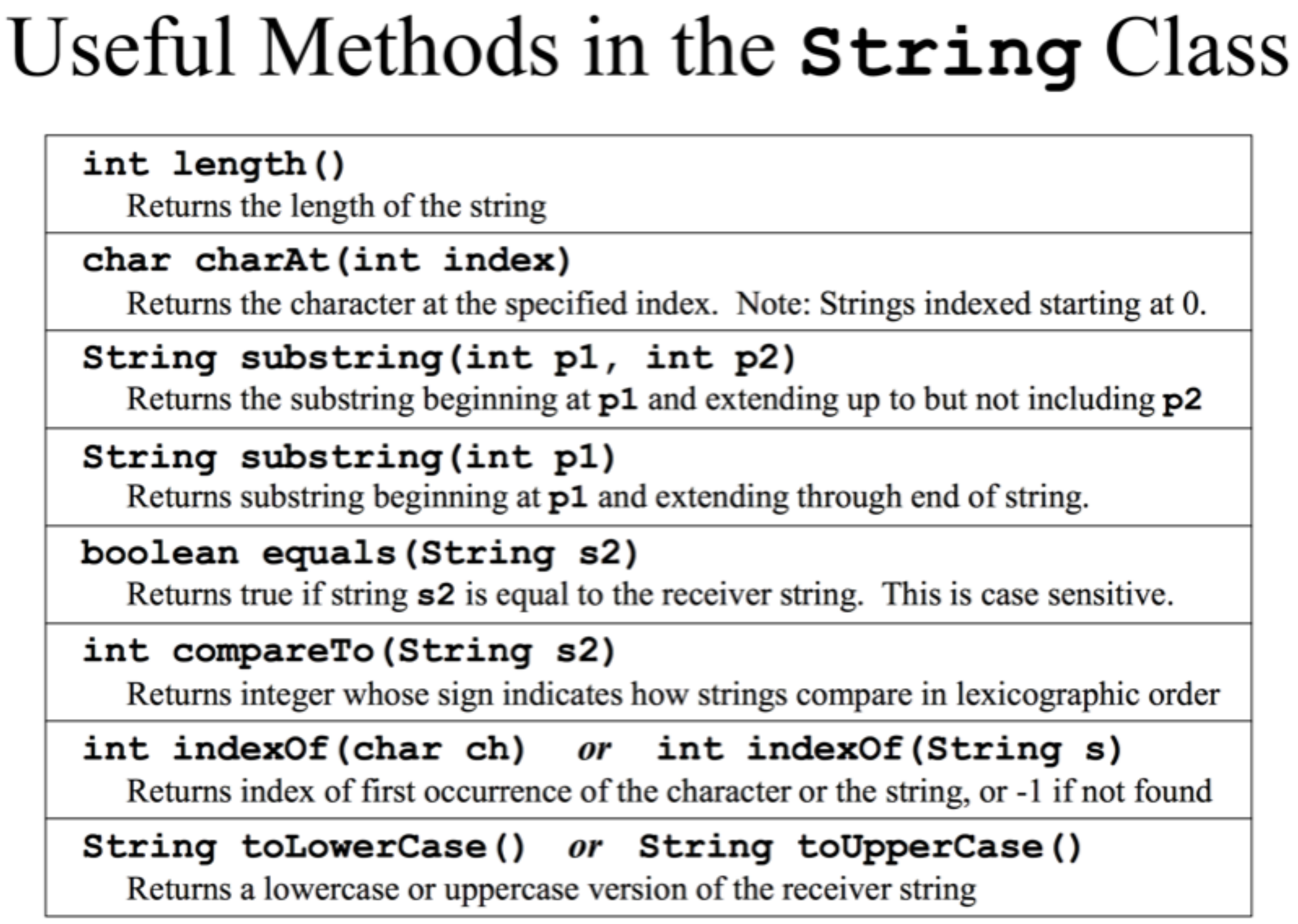
Characters

```
char ch = ‘a’;
Character.toUpperCase(ch);     // does nothing!
ch.toUpperCase();              // won’t compile!
ch = Character.toUpperCase(ch);    // ✓

if (Character.isUpperCase(ch)) {
     println(ch + “ is upper case!”);
}
```

Strings

- **Note:** strings are (immutable) objects. This means we <u>can</u> call methods on them!
- We *<u>cannot change a string after creating it</u>*. We can *overwrite* the entire variable with a new string, but we cannot go in and modify an existing string.
- Strings can be combined with ints, doubles, chars, etc.

Strings



Useful Methods in the `String` Class

`int length()`	Returns the length of the string
`char charAt(int index)`	Returns the character at the specified index. Note: Strings indexed starting at 0.
`String substring(int p1, int p2)`	Returns the substring beginning at **`p1`** and extending up to but not including **`p2`**
`String substring(int p1)`	Returns substring beginning at **`p1`** and extending through end of string.
`boolean equals(String s2)`	Returns true if string **`s2`** is equal to the receiver string. This is case sensitive.
`int compareTo(String s2)`	Returns integer whose sign indicates how strings compare in lexicographic order
`int indexOf(char ch)` *or* `int indexOf(String s)`	Returns index of first occurrence of the character or the string, or -1 if not found
`String toLowerCase()` *or* `String toUpperCase()`	Returns a lowercase or uppercase version of the receiver string

Using portions of slides by Eric Roberts

Strings

```
String str = “Hello world!”;      // no new needed
str.toUpperCase();                // does nothing!
str = str.toUpperCase();          // ✓

for (int i = 0; i < str.length(); i++) {
     println(str.charAt(i));
}
// prints each char on its own line
```

Putting It All Together

```
String str = "'ello mate!";
str = str.substring(1);
str = 'H' + str;            // str = "Hello mate!"
String newStr = "";
for (int i = 0; i < str.length(); i++) {
     newStr = str.charAt(i) + newStr;
}
// newStr = "!etam olleH"
```

Type Conversion

```
println(“B” + 8 + 4);
// prints “B84”
println(“B” + (8 + 4));
// prints “B12”
println(’A’ + 5 + “ella”);
// prints “70ella (note: ‘A’ corresponds to 65)”
// just an example; you don’t need to know int values of chars!
println((char)(‘A’ + 5) + “ella”);
// prints “Fella”
```

Type Conversion

- This seems nonsensical - but it’s not!
- **Just use precedence rules** and keep track of the type along the way. Evaluate 2 at a time.

```
println(’A’ + 5 + “ella”);
// ‘A’ + 5 is int (70), int + “ella” is string
println((char)(‘A’ + 5) + “ella”);
// ‘A’ + 5 is char (‘F’), char + “ella” is string
```

Strings Practice

Lets write a method that, given a string, removes all **strings within asterisks** and returns the result.

```
String str = “int s = 2; * This is 2 *”;
println(removeComments(str)); // “int s = 2; ”
str = “Hi * Hello * Hello”;
println(removeComments(str)); // “Hi  Hello“
str = “No comments!”;
println(removeComments(str)); // “No comments!“
```

Challenge: solve this before proceeding to solution!

Strings Practice

- Super helpful Strings pattern: given a string, iterate through and build up **a new string**. (Since strings are immutable!)

```
String oldStr = ...
String newStr = “”;
for (int i = 0; i < oldStr.length(); i++) {
    // build up newStr
}
```

Strings Practice

```
private String removeComments(String str) {
	String newStr = “”;
	boolean inComment = false;
	for (int i = 0; i < str.length(); i++) {
		if (str.charAt(i) == ‘*’) inComment = !inComment;

		// Note: if at end of comment, inComment already false!
		if (!inComment && str.charAt(i) != ‘*’) {
			newStr += str.charAt(i);
		}
	}
	return newStr;   // DON’T FORGET!!
}
```

Parting Words

Parting Words

- Try to get to every problem
- Don’t rush to coding too quickly
- Pseudocode!
- Look over the practice midterm

Good Luck!