CS106A Midterm Review Session

Annie Hu

Summer 2018

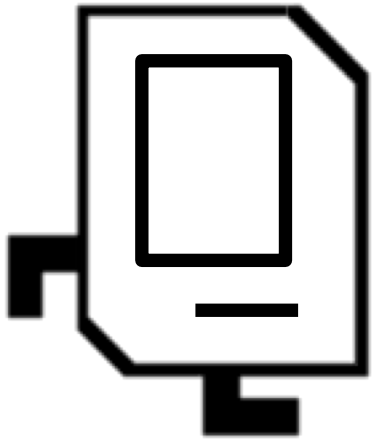
Logistics

- Closed-book, closed-notes
  - Two double-sided sheets of notes allowed
  - You will be provided a reference sheet
- Download BlueBook ahead of time
  - Bring your laptop + charger!
- Functionality should be your main goal, but good style often goes hand in hand with good functionality

Major Topics

- Karel
- Expressions and Variables
- Java Control Statements
- Methods, parameters, returns
- Randomness
- Characters and Strings
- Scanners and File processing
- Graphics Programs
- Memory and Tracing

Karel



Karel the Robot

- Tips:
  - Pseudocode first
  - Decompose the problem
  - Only Karel features!
  - Not allowed:
    - Variables (other than `int i` in for loop
    - Parameters / return
    - break

Expressions and Variables

Variables

- `int count = 0;`
- `double height = 5.2;`
- `boolean readyForMidterm = true;`
- `char letter = 'a';`
- `String str = "I love CS106A!";`

- (which of these are primitives?)

Expressions

- Evaluate:

```
3.0 * (23 % 5) / 2 + 2 * 7 / 3                    =
13 / 2 / 2.0 + 5 / 2.0 / 2                        =
6 == 3 * 2 && !(7 < 6) && 1 + 1 != 3              =
2 + 2 + "[" + 4 * 2 + "]" + 3 + 5                 =
```

Expressions

- Evaluate:

```
3.0 * (23 % 5) / 2 + 2 * 7 / 3                  = 8.5
13 / 2 / 2.0 + 5 / 2.0 / 2                      = 4.25
6 == 3 * 2 && !(7 < 6) && 1 + 1 != 3            = true
2 + 2 + "[" + 4 * 2 + "]" + 3 + 5               = "4[8]35"
```

Java Control Statements

Java Control Statements

- `if`
  - do something **once** if a condition is true
- `while`
  - do something **while** a condition is true
- `for`
  - do something a given **number of times**

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

The "Fencepost" Structure

- Loop over a set of statements, but do some part of those statements *one additional time*
- Frequently comes up in Karel and user input
- Can use *loop-and-a-half*

```
putBeeper();                    // post
while (frontIsClear()) {
  move();                       // fence
  putBeeper();                  // post
}
```

```
int sum = 0;
while (true) {
      int num = readInt("Number? ");
      // half-loop
      if (num == -1) break;
      sum += num;
}
println("Sum is " + sum);
```

Methods, Parameters, and Returns (oh my!)

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.

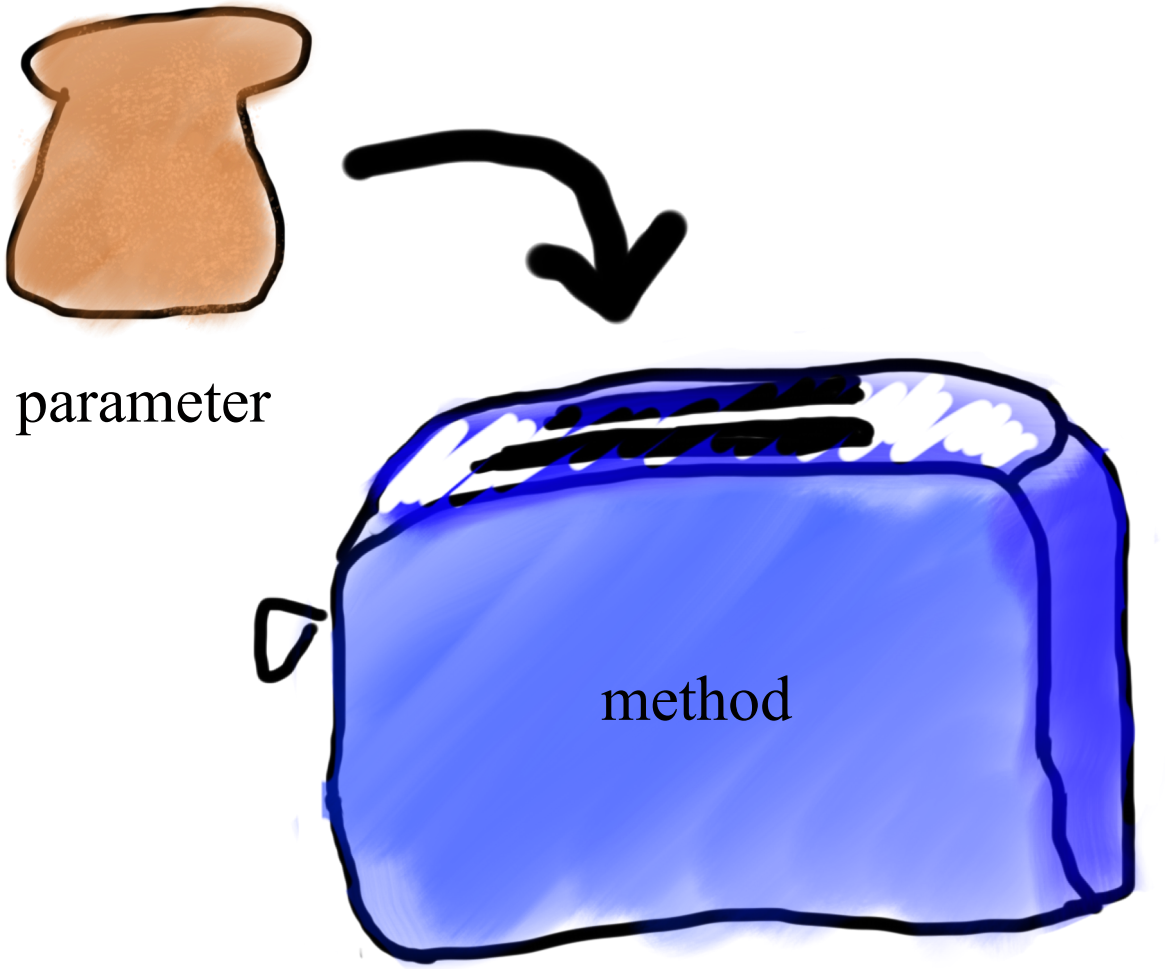
Java Constructs: Methods

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

Java Constructs - Methods

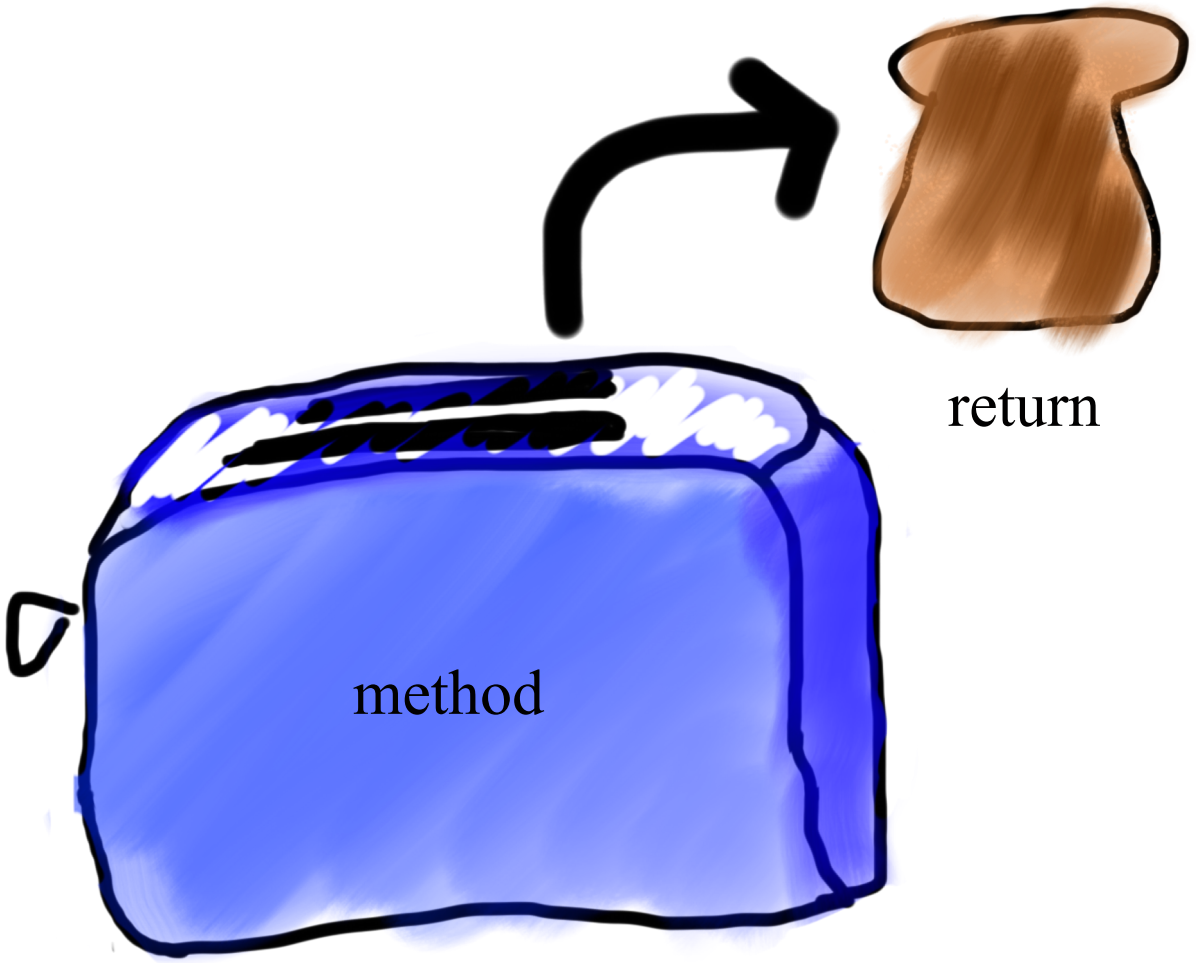
parameter

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? ");
```

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

```
                                        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.

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;
}
```

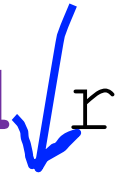
Returns the *value of* this expression (centimeters).

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

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

```
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;
}
```

7

7

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;
}
```

7

7

7

700

Return

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

7

700

Return

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

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

Return

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

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

7

700

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

7

700

Approaching Traces

- local variables in caller **distinct** from callee
- parameters just assigned names by the order in which they're passed

- tricky spots
  - precedence / variable names
  - what's in scope??

- **draw pictures and label variable values!**

Randomness

RandomGenerator

- `int num = RandomGenerator.getInstance().nextInt(1, 5);`

- Can be used to generate:
  - Integers: nextInt(min, max)
  - Doubles: nextDouble(min, max)
  - Colors: nextColor()
  - Booleans: nextBoolean()

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 primitives. 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 parameter.
- These methods *do not change the char*! 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 can call methods on them!
- We *cannot change a string after creating it*. 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: Indexing

Substring: remember that first index is **inclusive** while second is **exclusive**


```
s.substring(4, 10) // "o, wor"
```

Strings Useful Methods

		`String s = "Hello, world!";`
`s.charAt(index)`	Returns character at given index	`s.charAt(2); // '1'` `s.charAt(7); // 'w'`
`s.substring(start, end)` `s.substring(start)`	Returns part of string between given indices	`s.substring(1, 4); // "ell"` `s.substring(7); // "world!"`
`s1 += s2` `s1 = s1 + s2`	Concatenates string s2 to the end of string s1	`s += "!!" // "Hello, world!!"`
`Integer.parseInt(s)`	Converts string into integer representation, if valid	`s = "42";` `Integer.parseInt(s); // 42`

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
```

Type Conversion

- **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

- 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: Don’t forget

- Compare strings using `str.equals(str2)` NOT `str1 == str2`
- chars = single quote, strings = double quote
- to convert char -> string, concatenate with empty string (‘a’ + “” => “a”)
- if a string has N characters, indices go from 0 to N-1
- strings are **immutable**

Scanners and File Processing

File Reading and Scanners

- Use your syntax reference sheet if unsure!

`scanner.next()`	Returns next token (as separated by a space)

`scanner.nextLine()`
`scanner.nextInt()`
`scanner.nextDouble()`
	Returns next line, int, or double

`scanner.hasNext()`
`scanner.hasNextLine()`
`scanner.hasNextInt()`
	Returns true or false value indicating whether the scanner has any more of the given token lined up

`scanner.useDelimiter(String delimiter)`	Uses a different pattern than a space to separate tokens

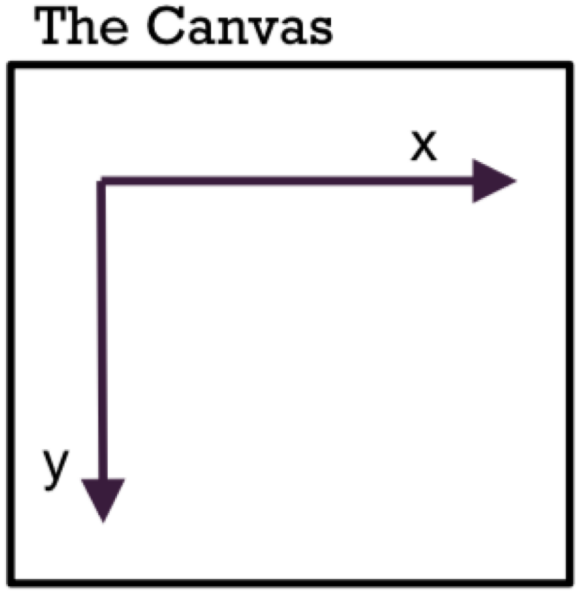
String and Scanners Practice

```
private void parse(String str) {
     Scanner scanner = new Scanner(str);
     while (scanner.hasNext()) {
          String token = scanner.next();
          println(token);
     }
     scanner.close();
}
```

File Reading Practice

```
try {
    Scanner input = new Scanner(new
File("res/data.txt"));
    while (input.hasNextLine()) {
        String line = input.nextLine();
        println(line);
    input.close();
} catch (IOException ex) {
    println("Error reading the file: " + ex);
}
```

Graphics Programs



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();
      pause(PAUSE_TIME);
}
```

Event Handlers

- **Example:** mouse events

```
public void run() {
      // Java runs this when program launches
}

public void mouseClicked(MouseEvent e) {
      // Java runs this when mouse clicked
}
```

Event-Driven Programming

There are many different types of mouse events. Each takes the form:

```
public void eventMethodName(MouseEvent e) {
```

…and contain, at least, the following information:

`e.getX()`	the x-coordinate of mouse cursor in window
`e.getY()`	the y-coordinate of mouse cursor in window

Memory

Instance Variables

```
private type name; // declared outside of any method
```

- scope is throughout an entire file
- useful for data you need throughout the program, or cannot be stored as parameters (e.g. event handling)

Primitives vs. Objects

	Primitives	Objects
What do they store in their variable box, directly?	Actual value	Location of the object
How do you compare them?	==	.equals()
How are they passed as parameters?	By **copy** (value)	By **reference** (passes location of original)
Does the original change when it's passed as a parameter?	No	Yes
How are they created?	Normal declaration	With **new**

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

"null"

- Only objects can be **null**
- Check if a variable is **null**:

```
if (mole == null) { …
```

- Why?

“null”

```
// may be a GObject, or null if nothing at (x, y)
GObject mole = getElementAt(x, y);
if (mole != null) {
    int x = mole.getX(); // OK
} else {
    int x = mole.getX(); // CRASH!
}
```

Parting Words + Tips

Tips

- Try to get to every problem
- Don't rush to coding too quickly. Read all instructions.
- Look over the practice midterms
- More practice:
  - Section problems
  - CodeStepByStep
  - Review concepts from assignments
  - Textbook

Tips

- Two kinds of questions: read and write
- Reading questions (e.g. code trace)
  - Write out everything clearly
  - Pay attention to details
- Writing questions
  - Pseudocode!
  - Can you decompose to make it easier?
  - Pay attention to edge cases

Questions?

Good Luck! :-)

Extra Slides

Program Trace

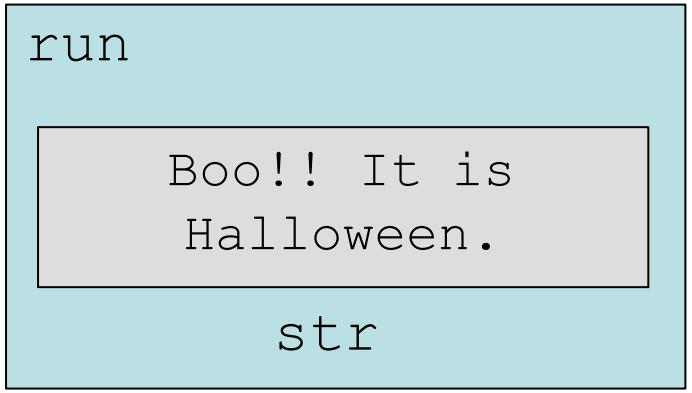
```
public void run() {
    String str = "Boo!! It is halloween.";
    println(trickOrTreat(str, 6));
    int candy = 5;
    int costume = 6;
    candy = howMuchCandy(candy, costume);
    println("I got " + candy + " candy(ies)");
}

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!

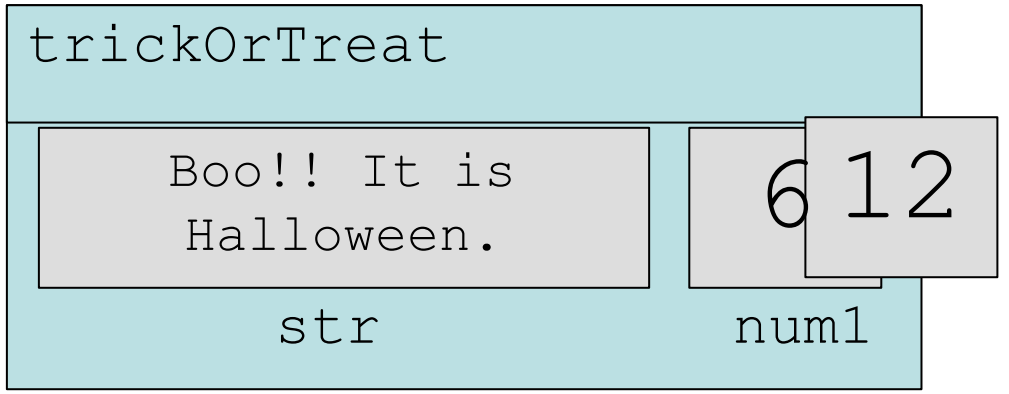
(Dug up from an old program – do not write code like this at home! :))

Program Trace

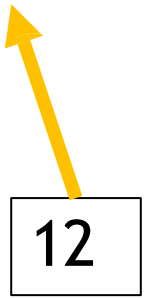
```
public void run()  {
      String str = "Boo!! It is halloween.";
      println(trickOrTreat(str, 6));
      ...
}
```

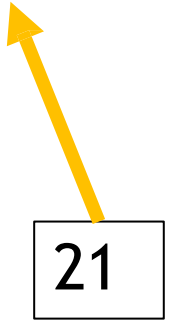


Program Trace



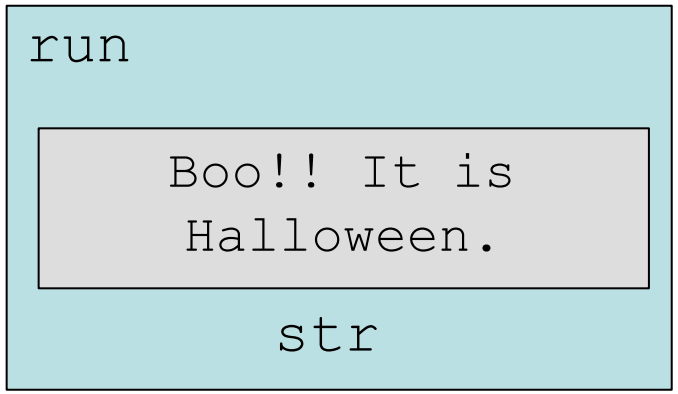

```
private String trickOrTreat(String str, int num1) {
      num1 *= 2; // 12
      return str.substring(num1, str.length() - 1);
}
```





Program Trace

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



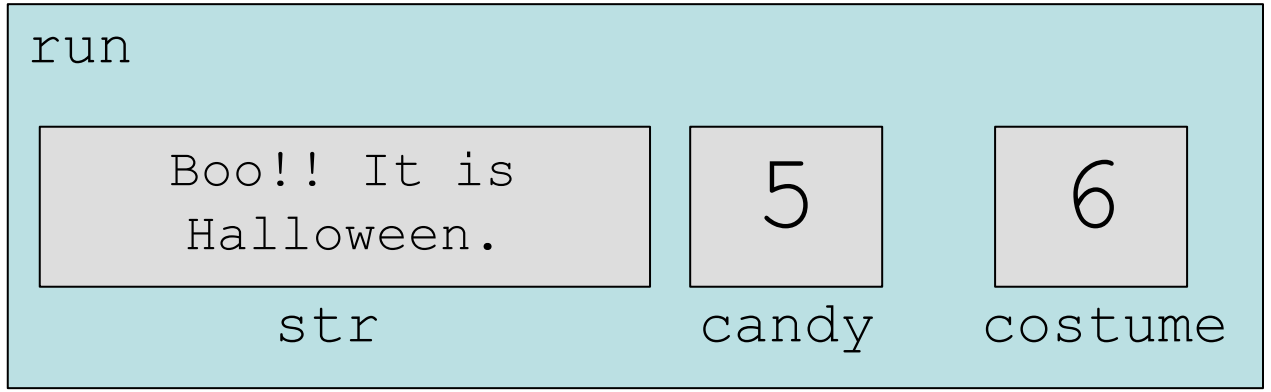
halloween

(Console)

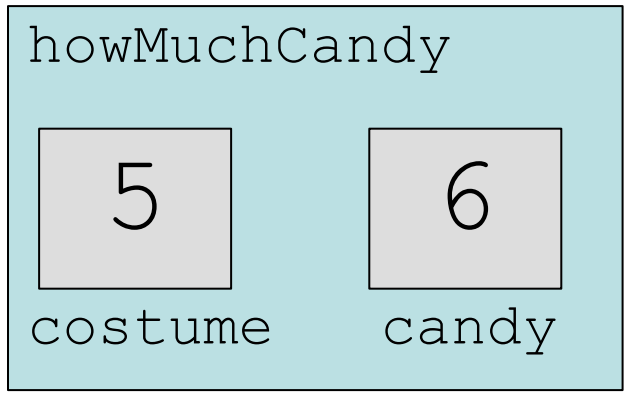
Program Trace

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

5 6

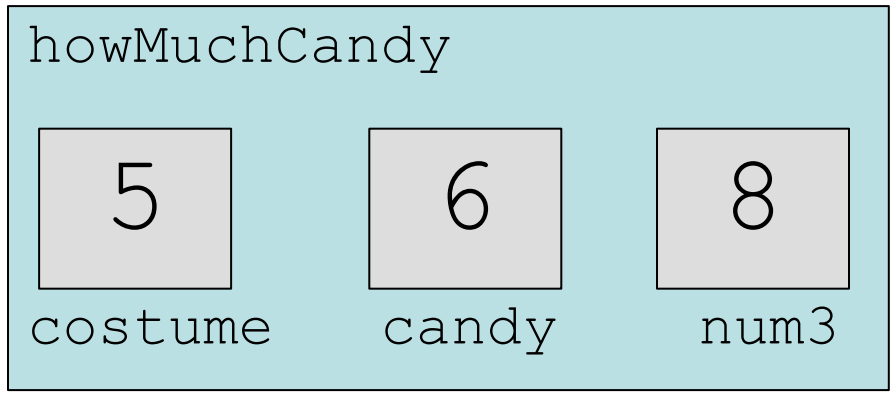


Program Trace




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

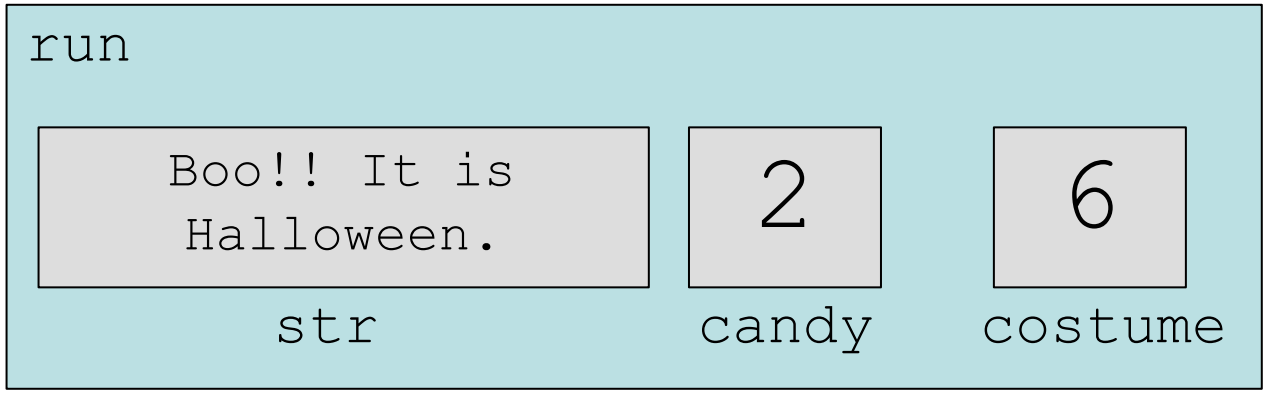
Program Trace




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

Program Trace

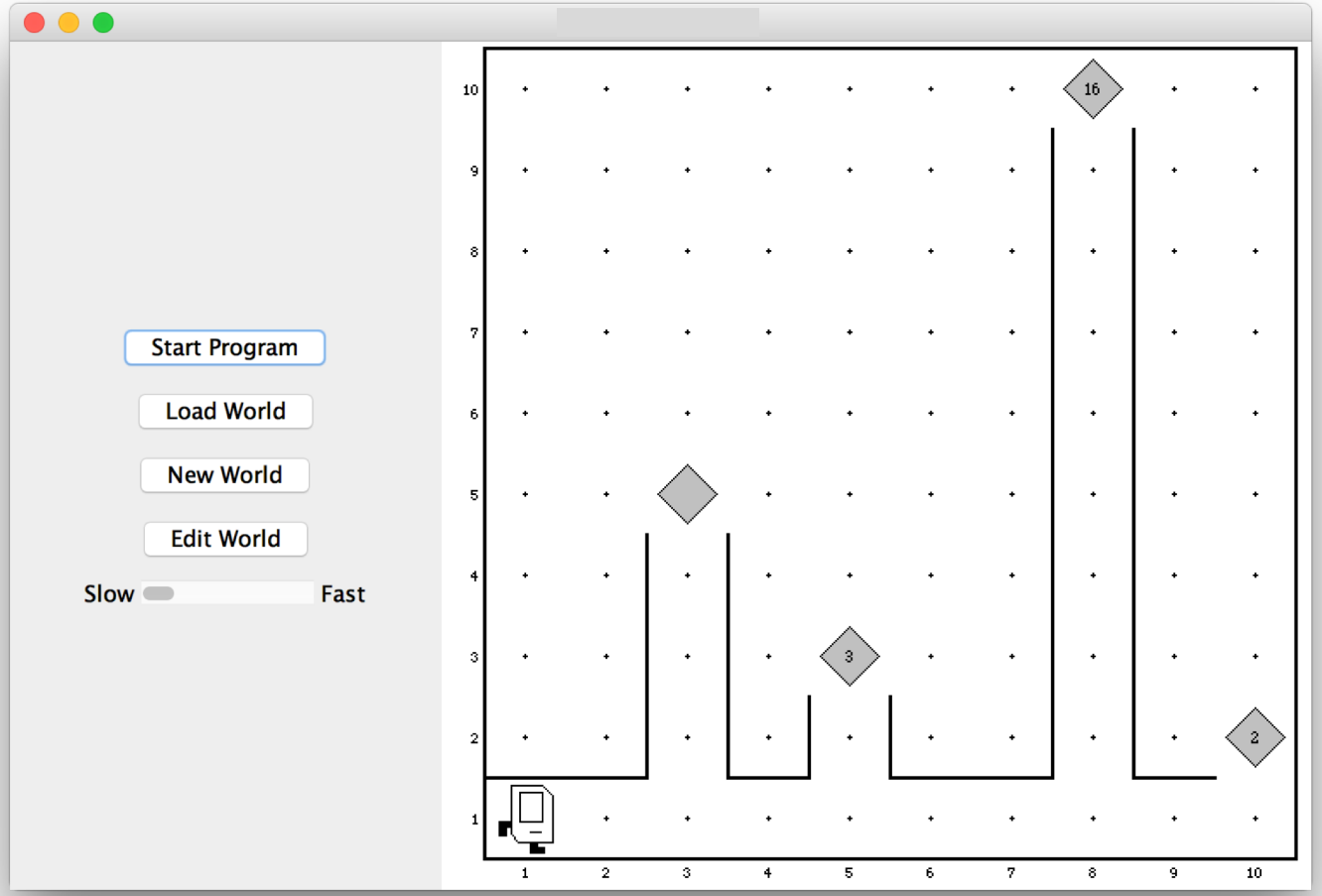
```
public void run() {
        ...
        int candy = 5;
        int costume = 6;
        candy = howMuchCandy(candy, costume);
        println("I got " + candy + " candy(ies)");
}
```



(Console)

halloween
I got 2 candy(ies)

Mail Karel


Start Program
Load World
New World
Edit World
Slow
Fast
16
3
2
1
2
3
4
5
6
7
8
9
10

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();
      }
}
```