CS106A Review Session

Sunday Oct. 29, 2017
Nick Troccoli
Topic List

• Karel
• Java constructs
• Graphics + Animation
• Classes and Interfaces
• Memory (Pass-by-reference vs. pass by value)
• Event-driven programming
• Characters and Strings
Karel
• Tips:
  • Pseudocode first
  • Decompose the problem
  • Might be limitations on constructs
    • E.g. no Java features (variables, break, etc.)
Trick-or-Treat Karel
Karel is in a world with walkways to houses that have candy. Karel should go to every house in order, go up the walkway and take all the candy (beepers). House walkways can be any distance apart, and have guide walls on the left and right up to the candy pot.

Challenge: solve this before proceeding to solution!
Trick-or-Treat Karel

Loop:
- if there’s a house: trick or treat
- if front is clear: move

Trick or treat:
- traverse walkway
- take candy 🍬
- traverse walkway
public void run() {
    while (frontIsClear()) {
        if (leftIsClear()) {
            trickOrTreat();
        }
        if (frontIsClear()) {
            move();
        }
    }
    if (leftIsClear()) {
        // maybe house on the last square!
        trickOrTreat();
    }
}
private void trickOrTreat() {
    turnLeft();
    traverseWalkway();
    takeCandy();
    turnAround();
    traverseWalkway();
    turnLeft();
}
private void traverseWalkway() {
    move();
    while (leftIs Blocked() && rightIsBlocked()) {
        move();
    }
}
private void takeCandy() {
    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
Methods let you define custom Java commands.
Parameters let you provide a method some information when you are calling it.
Return values let you give back some information when a method is finished.
Java Constructs - Methods
Java Constructs - Methods

parameter
Java Constructs - Methods
Java Constructs - Methods

return
Example: readInt

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

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

- We call `readInt`
- We give `readInt` some information (the text to print to the user)
When we include values in the parentheses of a method call, this means we are passing them as parameters to this method.

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

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

```java
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.

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

- **Variable types**: primitives (int, double, \ldots) + objects (GRect, Goval, \ldots)
- **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.

```java
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...

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

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

...and the second parameter value as `height`.
```
Parameters: `drawBlueRect`

drawBlueRect(50, 20);

We call `drawBlueRect` some information (the size of the rect we want).
Parameters: `drawBlueRect`

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

drawBlueRect(width, height);
```
Parameters: `drawBlueRect`

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

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

define `drawBlueRect()`

drawBlueRect(70, 40);

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

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

Parameter names do not affect program behavior.
public void run() {
    int width = ... 70
    int height = ... 40
    drawBlueRect(width, height);
}

private void drawBlueRect(int width, int height) {
    ...
}
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
private double metersToCm(double meters) {
    ...
}

When this method finishes, it will return a double.
private double metersToCm(double meters) {
    double centimeters = meters * 100;
    return centimeters;
}

Returns the value of this expression (centimeters).
public void run() {
    double cm = metersToCm(10);
    ...
}

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

```java
public void run() {
    double cm = metersToCm(10);
    ...
}
```
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;
}
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;
}
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;
}
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;
}
public void run() {
    double meters = readDouble("# meters? ");
    ...
    double cm = metersToCm(meters);
    println(cm + " centimeters.");
}
public void run() {
    double meters = readDouble("# meters? ");
    println(metersToCm(meters) + " cm.");
}

private double metersToCm(double meters) {
    ...
}
public void run() {
    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.
public void run() {
    double meters = readDouble("# meters? ");
    ...

    metersToCm(meters); // Does nothing!
    ...
}
public void run() {
    double meters = readDouble("# meters? ");
    ...
    700
    metersToCm(meters); // Does nothing!
    ...
}
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;
}
public void run() {
    String str = "Boo!! It is halloween."
    println(trickOrTreat(str, 6));
    ...
}
private String trickOrTreat(String str, int num1) {
    num1 *= 2; // 12
    return str.substring(num1, str.length() - 1);
}
public void run() {
    String str = "Boo!! It is halloween."
    println(trickOrTreat(str, 6));
    ...
}

halloween

(Console)
public void run() {
    ...
    int candy = 5;
    int costume = 6;
    candy = howMuchCandy(candy, costume);
    println("I got \" + candy + \" candy(ies)\);
}
private int howMuchCandy(int costume, int candy) {
    int num3 = costume + candy / 2; // 8
    return num3 % 3; // 2
}
public void run() {

    ... 

    int candy = 5;
    int costume = 6;
    candy = howMuchCandy(candy, costume);
    println("I got " + candy + " candy(ies)");
}

halloween
I got 2 candy(ies)
Program Trace

- **Tricky spots:** precedence, parameter/variable names...
- Draw pictures! / Label variable values
Graphics + Animation
• 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 $\text{getAscent}$. 
Standard format for animation code:

```java
while (CONDITION) {
    updateGraphics();
    performChecks();
    pause(PAUSE_TIME);
}
```
Classes and Interfaces
• A class is a way to tell Java about a new variable type. (AKA a new “class” of object).

• To do that, you must define what information that object contains (instance variables), what it can do (methods), and how you create one (constructor).
Let’s design a class called TrickOrTreater. They will start with some initial candy in their bag, and be able to trick or treat to get more candy. They can also eat candy.
public void run() {
    TrickOrTreater nick = new TrickOrTreater(4);
    nick.trickOrTreat(100);
    println(nick.getNumCandies()); // 104
    for (int i = 0; i < 50; i++) {
        nick.eatCandy();
    }
    println(nick.getNumCandies()); // 54
}
• Step 1: What information does this variable type store?

1 instance variable for storing the # of candies
• Step 2: What can this variable type do?

Methods:
- `trickOrTreat(candy)`: adds candy to bag
- `eatCandy()`: removes 1 candy
- `getNumCandies()` returns # candies
• Step 3: How do we create a new instance of this variable type?

Create one and provide initial candy in bag.
public class TrickOrTreater {
    private int numCandies = 0;
    public TrickOrTreater(int candies) {
        numCandies = candies;
    }
    public void eatCandy() {
        numCandies--;
    }
    public void trickOrTreat(int numCandy) {
        numCandies += numCandy;
    }
    public int getNumCandies() {
        return numCandies;
    }
}
What’s up with this? (Note: no return type, same name as the class, public).

```java
public TrickOrTreater(int candies) {
    numCandies = candies;
}
```
• This is the class’s **constructor**. It’s the method called when a new **TrickOrTreater** is created to initialize it.

```java
TrickOrTreater nick = new TrickOrTreater(5);
```
Constructors

• You can have constructors that take different arguments, to initialize a type different ways.

```java
public TrickOrTreater(int candies) {
    numCandies = candies;
}

public TrickOrTreater() {
    numCandies = 4;
}
```
An interface is a list of method names (they’re unimplemented!). Any class can implement an interface, which means they provide an implementation in their code of every method listed in the interface.

Why is this useful?
• Interfaces let different classes tell Java they implement the same behavior.
  • E.g. you can call setFilled on both GRect and GOval
• But each class can uniquely implement this behavior.
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 on the heap** where the object actually lives.

• When you make a primitive, the local variable is a box that stores the **actual value**.
• `==` 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
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);
}
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!
}
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**

- **Two** types of events: keyboard and mouse
- **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 key or 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.
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();
}
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
private static final int SQUARE_SIZE = 100;
private static final int PAUSE_TIME = 100;
GRect square;
GOval circle;
boolean isShowingSquare = true;

private RandomGenerator rgen = RandomGenerator.getInstance();
public void run() {
    setup();
    while (true) {
        Color c = rgen.nextColor();
        square.setColor(c);
        circle.setColor(c);
        pause(PAUSE_TIME);
    }
}
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(x, y, SQUARE_SIZE, SQUARE_SIZE);
    square.setFilled(true);
    circle.setFilled(true);
    add(square); // only add square!
    addMouseListener();
}
public void mouseClicked(MouseEvent e) {
    if (isShowingSquare) {
        remove(square);
        add(circle);
    } else {
        remove(circle);
        add(square);
    }
    isShowingSquare = !isShowingSquare;
}
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 (“”).
char uppercaseA = ‘A’;
char uppercaseB = (char)(uppercaseA + 1);
int lettersInAlphabet = ‘Z’ – ‘A’ + 1;
// equivalent: ‘z’ – ‘a’ + 1
// A to Z and a to z are sequential numbers.
# Useful Methods in the `Character` Class

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>static boolean isDigit(char ch)</code></td>
<td>Determines if the specified character is a digit.</td>
</tr>
<tr>
<td><code>static boolean isLetter(char ch)</code></td>
<td>Determines if the specified character is a letter.</td>
</tr>
<tr>
<td><code>static boolean isLetterOrDigit(char ch)</code></td>
<td>Determines if the specified character is a letter or a digit.</td>
</tr>
<tr>
<td><code>static boolean isLowerCase(char ch)</code></td>
<td>Determines if the specified character is a lowercase letter.</td>
</tr>
<tr>
<td><code>static boolean isUpperCase(char ch)</code></td>
<td>Determines if the specified character is an uppercase letter.</td>
</tr>
<tr>
<td><code>static boolean isWhitespace(char ch)</code></td>
<td>Determines if the specified character is <code>whitespace</code> (spaces and tabs).</td>
</tr>
<tr>
<td><code>static char toLowerCase(char ch)</code></td>
<td>Converts <code>ch</code> to its lowercase equivalent, if any. If not, <code>ch</code> is returned unchanged.</td>
</tr>
<tr>
<td><code>static char toUpperCase(char ch)</code></td>
<td>Converts <code>ch</code> to its uppercase equivalent, if any. If not, <code>ch</code> is returned unchanged.</td>
</tr>
</tbody>
</table>
• **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.**
• Strings can be combined with ints, doubles, chars, etc.
Strings

Useful Methods in the **String** Class

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

*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
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"
println("B" + 8 + 4);
// prints “B84”
println("B" + (8 + 4));
// prints “B12”
println(‘A’ + 5 + “ella”);
// prints “70ella (note: ‘A’ corresponds to 65)”
println((char)(‘A’ + 5) + “ella”);
// prints “Fella”
Type Conversion Mayhem

• 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.

```java
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.

```java
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!)

```java
String oldStr = ...;
String newStr = "";
for (int i = 0; i < oldStr.length(); i++) {
    // build up newStr
}
```
```java
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!!
}
```
A **string tokenizer** is an object that chops up strings for you. (Handy!). By default, by spaces, but you can specify custom DELIMITERS.

- You get the tokens back one at a time to use.
- `hasMoreTokens()` returns a boolean
- `nextToken()` returns the next token as a string
• Practice problem: Spongebob! (S1:E17 – Rock Bottom)
• Lets insert “pftth” in between every word in a string. (Because why not?)

```java
String str = "When is the next bus";
println(pftth(str));
// "When pftth is pftth the pftth next pftth
// bus pftth"
```

Challenge: solve this before proceeding to solution!
private String pftth(String str) {
    StringTokenizer t = new StringTokenizer(str);
    String newStr = "";
    while (t.hasMoreTokens()) {
        newStr += t.nextToken() + "pftth";
    }
    return newStr;
}
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!