• Karel
• Java Constructs
• Graphics + Animation
• Memory and Tracing
• 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.)
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
public void run() {
    while (frontIsClear()) {
        if (leftIsClear()) {
            pickUpMail();
        }
        if (frontIsClear()) {
            move();
        }
    }
    if (leftIsClear()) { // maybe house on the last square!
        pickUpMail();
    }
}
private void pickUpMail() {
    turnLeft();
    traverseWalkway();
    takeMail();
    turnAround();
    traverseWalkway();
    turnLeft();
}
private void traverseWalkway() {
    move();
    while (leftIsBlocked() && rightIsBlocked()) {
        move();
    }
}
private void takeMail() {
    while (beepersPresent()) {
        pickBeeper();
    }
}
Java Constructs
Java Constructs

• **Variable types**: primitives (int, double, bool, char) + objects (GRect, GOval, ...)

• **Control statements**: if, while, for
  – What is each useful for?

• **Methods**
  – Parameters
  – Return
Java Constructs

• Variable types: primitives (int, double, bool, char) + objects (GRect, GOval,...)

• Control statements: if, while, for
  – What is each useful for?

• Methods
  – Parameters
  – Return
Variable Types: Primitive or Class?

- **CLASS**
  - Passed by reference to a called method
  - Initialized using `new` keyword
  - Have their own methods you can call

- **PRIMITIVE**
  - Passed by value to a called method
• **Variable types**: primitives (int, double,...) + objects (GRect, Goval,...)

• **Control statements**: if, while, for
  – 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
  – 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

- Parameter
- Method
Java Constructs - Methods
Java Constructs - Methods

return

method
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.

```
int x = readInt("Your guess? ");
```
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? ");
```
• **Variable types**: primitives (int, double,...) + objects (GRect, Goval,...)

• **Control statements**: if, while, for, switch
  – What is each useful for?

• **Methods**
  – Parameters
  – Return
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
}
```

`Inside drawBlueRect, refer to the first parameter value as width...`
Parameters: drawBlueRect

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.
We call `drawBlueRect` and give it some information (the size of the rect we want). The code snippet is:

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

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

drawBlueRect(width, height);
Parameters: drawBlueRect

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

drawBlueRect(70, 40);
Parameters: drawBlueRect

drawBlueRect(70, 40);
Parameters: drawBlueRect

```java
private void drawBlueRect(int width, int height) {
    // use width and height variables
    // to draw a rect at 0, 0
}
```
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
When this method finishes, it will return a `double`.

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

(Dug up from an old program – do not write code like this at home! :) )
public void run() {
    String str = "Boo!! It is halloween."
    println(trickOrTreat(str, 6));
    ...
}

Program Trace

run

Boo!! It is Halloween.

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

Program Trace

trickOrTreat

Boo!! It is Halloween.

str

num1

12

12

21
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)”);
}

run

Boo!! It is Halloween.

str   candy   costume
5     5       6
private int howMuchCandy(int costume, int candy) {
    int num3 = costume + candy / 2;
    return num3 % 3;
}

Program Trace

howMuchCandy

5
6
costume
candy
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)
• **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)

```java
while (CONDITION) {
    updateGraphics();
    performChecks();
    pause(PAUSE_TIME);
}
```
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. This uses the **new** keyword.

• 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.
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

• **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!
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.
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';

// 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.
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 whitespace (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.
```java
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

## 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) or 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*
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"
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

• 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
}
Parting Words
• Try to get to every problem
• Don’t rush to coding too quickly. Read all instructions.
• Pseudocode!
• Look over the practice midterms
• Functionality should be your main goal, but good style often goes hand in hand with good functionality.
• Download the exam software ahead of time
• Don’t forget laptop, charger, printed notes!
Good Luck!