Announcements, Reminders

• Creative Project 2 released – due Tuesday, Feb 6\textsuperscript{th}
  • Note: uses Javadoc!
  • Also helpful: section problems, + last week’s \textit{food for thought}

• Resubmission Cycle 2 form released
  • Note: this is the last time C0 is eligible for resubmission!

• \textbf{IPL tips!}

• Mid-Quarter Formative Feedback with Ken Yasuhara for part of class on Wednesday, Feb 7\textsuperscript{th}
(PCM) Conditionals (1/4)

```java
if (test) {
    body (statements to be executed)
}
```

Executes a block of statements if and only if the test is true
(PCM) Conditionals (2/4)

1. If the test is true: execute block of statements
2. If not, execute other block of statements

if (test) {
    statement(s)
} else {
    statement(s)
}
1. If the first test is true, execute that block
2. If not, proceed to the next test, and repeat
3. If none were true, don’t execute any blocks
With a large if-else-if-else chain,

- if there is an ending else, exactly one block will execute
- if there is no ending else, zero or one blocks will execute
public static void main(String[] args) {
    for (int i = 1; i <= 3; i++) {
        System.out.print(mystery(i));
    }
}

public static String mystery(int n) {
    if (n % 2 == 1) {
        return "odd ";
    } else if (n == 1) {
        return "one ";
    }
    return "even ";
}

What does this program output?
A. odd even odd
B. one even odd
C. one even even
D. even even even
public static void main(String[] args) {
    for (int i = 1; i <= 3; i++) {
        System.out.print(mystery(i));
    }
}

public static String mystery(int n) {
    if (n % 2 == 1) {
        return "odd ";
    } else if (n == 1) {
        return "one ";
    }
    return "even ";
}

This else if statement never runs!
mEtAcOgNiTiOn iS ThE KeY To bEiNg sUCCeSSfuL iN cOlLeGe
Common Problem-Solving Strategies

- **Analogy** – Is this similar to another problem you've seen?
- **Brainstorming** – Consider steps to solve problem before jumping into code
  - Try to do an example "by hand" → outline steps
- **Solve sub-problems** – Is there a smaller part of the problem to solve?
- **Debugging** – Does your solution behave correctly?
  - What is it doing?
  - What do you expect it to do?
  - What area of your code controls that part of the output?
- **Iterative Development** – Can we start by solving a different problem that is easier?
Common Problem-Solving Strategies

- **Analogy** – Is this similar to another problem you've seen?
- **Brainstorming** – Consider steps to solve problem before jumping into code
  - Try to do an example "by hand" → outline steps
- **Solve sub-problems** – Is there a smaller part of the problem to solve?
- **Debugging** – Does your solution behave correctly?
  - What is it doing?
  - What do you expect it to do?
  - What area of your code controls that part of the output?
- **Iterative Development** – Can we start by solving a different problem that is easier?
Common Problem-Solving Strategies

- **Analogy** – Is this similar to another problem you've seen?
- **Brainstorming** – Consider steps to solve problem before jumping into code
  - Try to do an example "by hand" → outline steps
- **Solve sub-problems** – Is there a smaller part of the problem to solve?
- **Debugging** – Does your solution behave correctly?
  - What is it doing?
  - What do you expect it to do?
  - What area of your code controls that part of the output?
- **Iterative Development** – Can we start by solving a different problem that is easier?
Poll in with your answer!

What is the output produced by executing this code?

A. 7 -1 12
B. -3 -1 13
C. 3 -1 13
D. 12 1 12
E. -14 1 13

```java
int a = 7;
int b = -1;
int c = 12;
if (a < b) {
a *= 2;
} else if (b < a) {
a /= 2;
} else {
a = c;
}
if (c % 2 == 0) {
c += 1;
}
if (b > 0) {
b *= -1;
} else if (a < 0) {
a *= -1;
}
System.out.print(a + " " + b + " " + c);
```
A weekly section where I introduce open problems related to our lecture topic(s) of the week.

Goals:
1. give you “conversational familiarity” with CS terminology
2. see how CS interacts with other fields and people!
3. point you in the direction of more CSE (or adjacent) classes

Note: not tested content. Just food for thought :)
We just learned that methods can call themselves. This is called “recursion”!

Fun fact: this is related to many, many concepts you may have seen in math:
- some keywords: “recurrence relation”, “induction”, “fractals”, “differential equation”

Another fun fact: this is normally a CSE 123 topic, though at other schools (who start with different languages), you learn this before loops.

Even funner (?) fact: some programming languages, including some of Matt’s favourites, do not allow variables or loops. You do everything with recursion!
So is this on the test???

Absolutely not.

But...

• this is (subtly) one of the fundamental ideas of computer science and repeats itself everywhere. Including with Social Networks :)

• if you love thinking about methods calling themselves, CSE (or math!) might be the right major for you!!

• and consider taking CSE 311 and CSE 341 :)