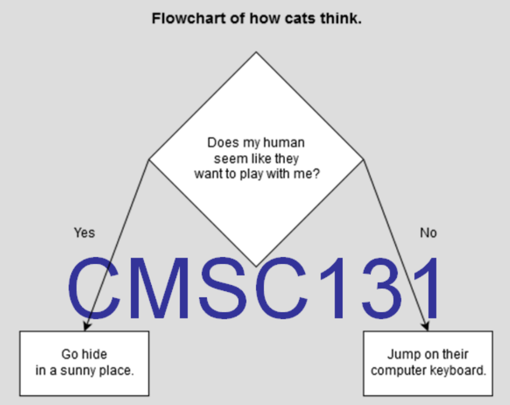

Flowchart of how cats think.
Does my human seem like they want to play with me?
Yes
No
Go hide in a sunny place.
Jump on their computer keyboard.

CMSC131

Conditional Statements and Logical Operators

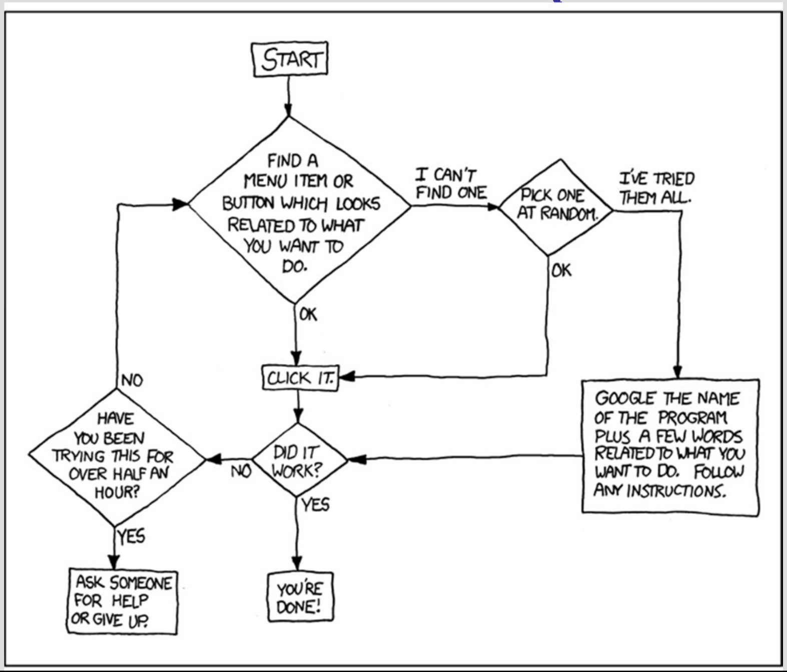
Flow of Control

The default "flow" through a program is going top-to-bottom, with each of the statements being executed in turn, one after the other.

We can alter this flow!

- Method calls {kind of, will discuss in more detail}
- Conditional statements (this slide set)
- Iteration (we will see this soon)

xkcd #627: Be a computer expert.



START
FIND A MENU ITEM OR BUTTON WHICH LOOKS RELATED TO WHAT YOU WANT TO DO.
I CAN'T FIND ONE
PICK ONE AT RANDOM.
I'VE TRIED THEM ALL.
OK
OK
CLICK IT.
NO
GOOGLE THE NAME OF THE PROGRAM PLUS A FEW WORDS RELATED TO WHAT YOU WANT TO DO. FOLLOW ANY INSTRUCTIONS.
HAVE YOU BEEN TRYING THIS FOR OVER HALF AN HOUR?
NO
DID IT WORK?
YES
YES
ASK SOMEONE FOR HELP OR GIVE UP.
YOU'RE DONE!

Conditional Statements

We can use a conditional statements to test whether something is true and then decide what to execute based on that.

- **if** statements
- **if-else** statements

if

if (*condition*) {
   statement(s) to execute…
}
next_statement_in_the_code;

- The **condition** is tested.
- **IF** it evaluates to **TRUE**, then the statements are executed and then control moves on to the next statement in the code.
- Otherwise (it evaluated to **FALSE)** control skips right to that next statement in the code without executing the statements inside the braces.

NOTE: For style purposes, we will *ALWAYS* place the statement(s) to execute within a { } block.

if-else

if (*condition*) {
   first group of statements to execute…
}
else {
   second group of statements to execute…
}
next_statement_in_the_code;

- The **condition** is tested.
- **IF** it evaluates to **TRUE**, then the first group of statements are executed after which control moves on to the next statement in the code.
- **ELSE** (it evaluated to **FALSE**) the second group of statements are executed after which control moves on to the next statement in the code.

NOTE: the first or second group are executed, not both, not neither.

Which prompt will this code display if I were to execute it now?

0% A. Click 1

0% B. Click 2

0% C. Both will print

```
if (rightNow.get(Calendar.DAY_OF_WEEK)==Calendar.MONDAY) {
    System.out.println("Click 1");
}
else {
    System.out.println("Click 2");
}
```

Response Counter

IsGreaterTest.java example

```
public static void main(String[] args) {
     final int THRESHOLD = 117;

     int value;
     Scanner sc = new Scanner(System.in);

     System.out.print("Enter a number: ");
     value = sc.nextInt();
     if ( value > THRESHOLD ) {
         System.out.println("Yay. " + value +
                    " is greater than our threshold.");
     }
     else {
         System.out.println("Too bad...");
     }

     sc.close();
}
```

SimpleConditional.java example

```
public static void main(String[] args) {
    int value;
    Scanner sc = new Scanner(System.in);

    System.out.print("Enter an odd number: ");
    value = sc.nextInt();
    if ( value%2 == 1 ) {//the % op returns the remainder
       System.out.println("That's great, thanks!");
    }
    else {
       System.out.println("That number was EVEN.");
    }
}
```

Will `(value%2==1)` always be true when `value` is an odd number?

0% 1. Yes

0% 2. No

0% 3. I'm not sure.

Response Counter

How would you fix this?

```
System.out.print("Enter an odd number: ");
value = sc.nextInt();
if ( value%2 == 1 ) {
   System.out.println("That's great, thanks!");
} else {
   System.out.println("That number was EVEN.");
}
```

Static Methods

Imagine you wanted to have the logic of determining whether an integer was odd in a single place.

We could create a static method in a class that takes a single integer as a parameter:

```
 public static boolean isOdd (int num) {
       return (num%2)!=0;
 }
```

An advantage is that if we put a piece of complex logic into a method such as this, if we later discover an error or a better way to do it we only have to update code in one place.

Some Logical Operators

We can create more detailed conditions using Boolean logic.

There are several operators available.

- and **&&** in Java
- or || in Java
- not **!** In Java

NOTE: Parenthesis are your friend if you are concerned about order of operations.

CompoundConditional.java "excerpts"

```
int num;
final int LOWER = 35; //Note the use of constants.
final int UPPER = 70;
…
if ((num > LOWER) && (num < UPPER)) {
   System.out.println("Thank you.");
}
else {
   System.out.println(
      "That's not between "+LOWER+" and "+UPPER+"!"
   );
}
```

CompoundConditional.java "excerpts"

```
int months, miles;
final int MONTH_BOUNDARY=3;
final int MILES_BOUNDARY=3000;
…
if (
       (months>=MONTH_BOUNDARY)
                    ||
       (miles>MILES_BOUNDARY))
{
   System.out.println("Get an oil change!");
}
else {
   System.out.println("Keep on driving...");
}
```

Constants in Examples

In some class examples I will use literal values where stylistically ***named constants*** would normally be used.

This is so that things fit well in the PowerPoint slides on-screen in these examples.

Nested/Cascading Conditionals

The "nesting" of conditionals is when the block of statements within an **if** or **else** block itself contains a conditional statement.

The "cascading" of conditionals is when you start an **else** by asking another **if** question.

```
if (n<10) {
   System.out.println("Less than 10");
}
else if (n<20) {
   System.out.println("10 or more but less than 20");
}
else {
   System.out.println("20 or more");
}
```

NestedConditional.java excerpt

```
if (numberOwned < 0) {
   System.out.println(
      "How can you own a negative number of animal + "s?");
}
else if (numberOwned == 0) {
   System.out.println("That's a shame :(");
}
else if ( (
            animal.equals("dog") ||
            animal.equals("cat") ||
            animal.equals("hamster")
          ) &&
          numberOwned < 4 ) {
   System.out.println("You are a typical "+animal+" owner.");
}
else {
   System.out.println("That's unusual!");
}
```

Conditionals and Values

What is a danger in the following code and how would you try to fix it?

```
public static void main(String[] args) {
   float taxrate;

   Scanner sc = new Scanner(System.in);
   String s = sc.next();

   if (s.equals("MD")) {
     taxrate = 0.06F;
   }
   System.out.println("Tax Rate is " + taxrate);
}
```

Coding Style

Projects might have some points attached to programming style.

Even if they don't, you should still get into the habit of writing well-styled code.

"Habits Eat Will-Power for Breakfast"[1]

The next few slides demonstrate POOR style to show you what NOT to do.

[1] http://sheridacon.com/2016/02/19/change-your-habits-will-power/

Which should you use for money?

0% A. float

0% B. double

0% C. int

0% D. long

Response Counter

Discuss and revote.
Which should you use for money?

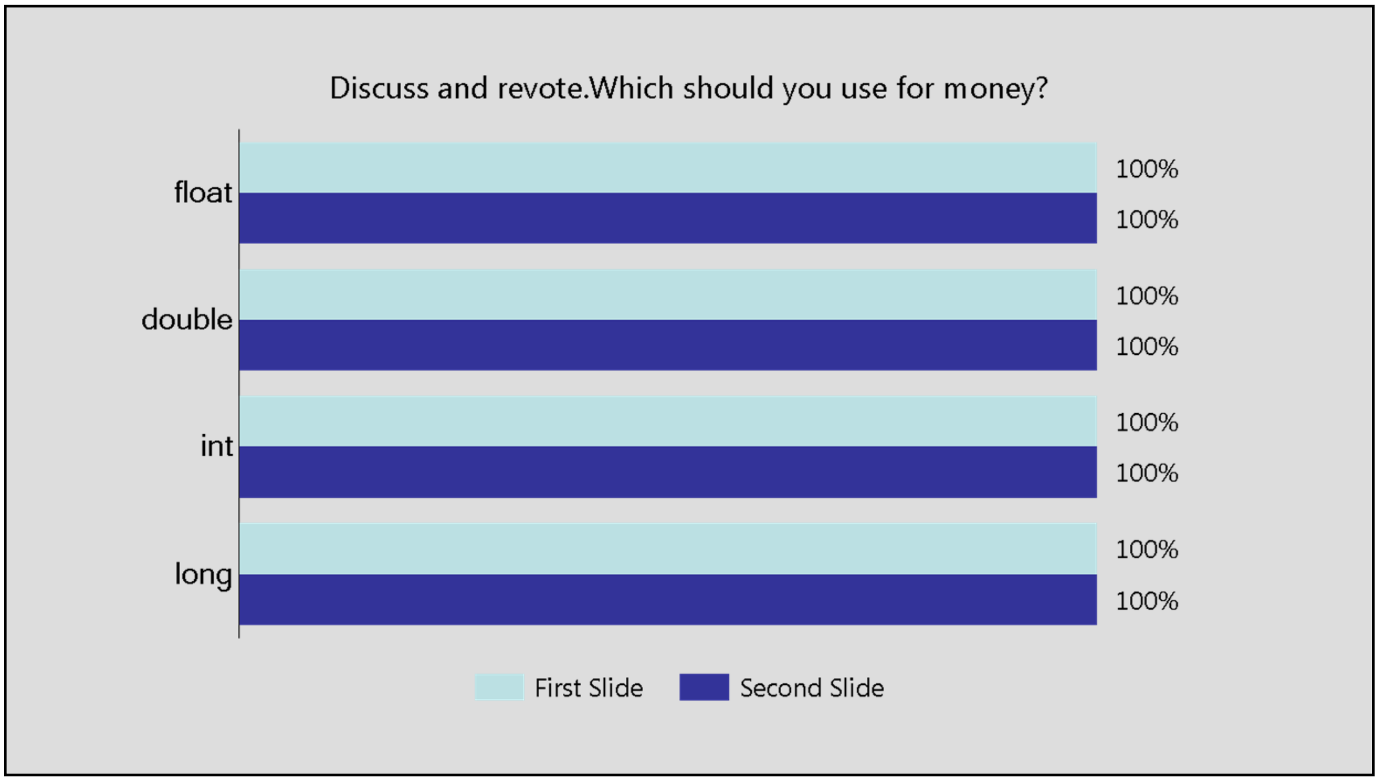
0% A. float

0% B. double

0% C. int

0% D. long

Response Counter



Testing something that must be so…

```
if (x > 20) {
      …
}
else if (x <= 20) {
      …
}
```

There is no need to test again in the else since the only way the program will get to that else is when "x > 20" was false which logically means that "x <= 20" ***must be true*** at that point.

== true

```
boolean flag;
…
if (flag == true) {
      …
}
```

The conditional statement should just be

```
      if (flag) {
```

in this type of situation.

== false

```
boolean flag;
…
if (flag == false) {
      …
}
```

The conditional statement should just be

```
      if (!flag) {
```

in this type of situation.

The ternary operator

The ternary operator is of the form

```
(boolean_expression)?if_true:if_false;
```

A simple example using assignment

```
String s=(x<0)?"Negative":"Not Negative";
```

Applications could include things such as

```
   minVal = (a < b) ? a : b;
   absValue = (a < 0) ? -a : a;
```