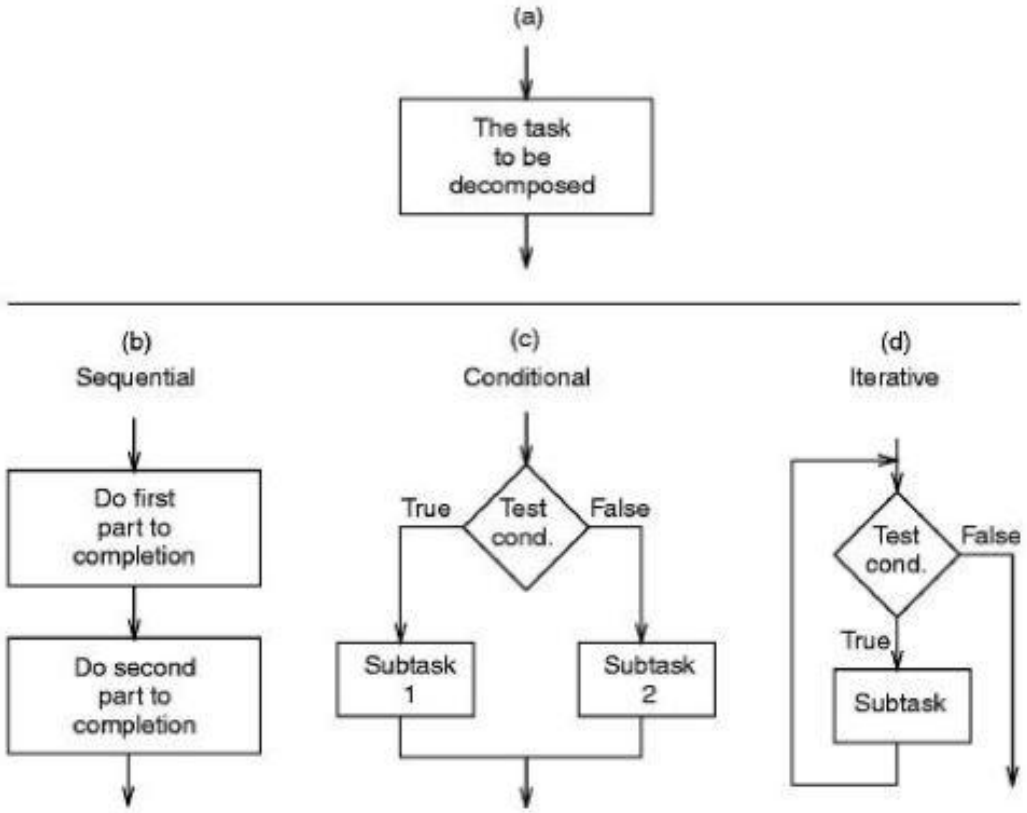
WEEK FOUR

Acknowledgements: Slides created based off material provided by Dr. Travis Doom

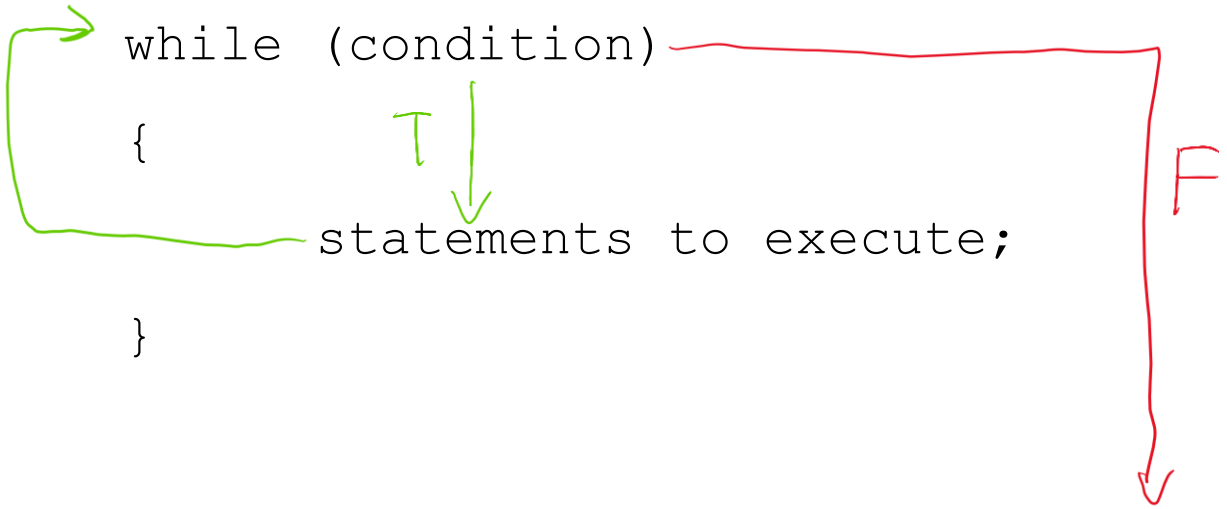
CONTROL STRUCTURES

- Sequential
  - Default
  - Do A -> B -> C -> …
- Selective/Conditional
  - Decision/choice
  - Do A if some condition, otherwise do B
- Iteration
  - Loops
  - Do A repeatedly until a condition is met



ITERATION: WHILE LOOPS

- Continues to execute a section of code while a condition is true



ITERATION: WHILE LOOPS

- While loops that never exit are possible (infinite loops)
- We want to avoid these

```
while (true)
{
    System.out.println("wee");
}
```

```
int counter = 5;

while (counter < 10)
{
    System.out.println(counter);
    counter = counter - 1;
}
```

counter = 5, 4, 3, 2...

ITERATION: WHILE LOOPS

```
int counter = 5;
while (counter > 0)
{
      System.out.println(counter);
      counter = counter - 1
}
```

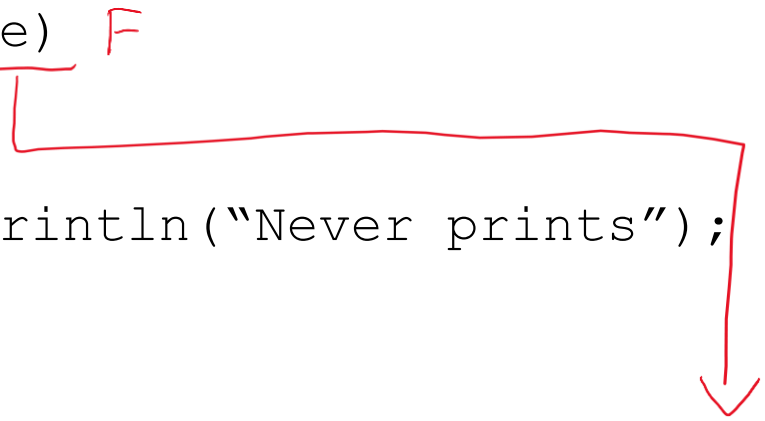
output:

5
4
3
2
1

ITERATION: WHILE LOOPS

- It is also possible for the entire loop to be skipped

```
boolean flag = false;
while (flag == true)
{
      System.out.println("Never prints");
}
```



ITERATION: WHILE LOOPS

- Off by one errors

```
// code to count up to 5
int num = 0;
while (num < 5)
{
        num = num + 1;
        System.out.println(num);
}
```

Output:

1
2
3
4
5

```
int num = 1;
while (num < 5)
{
        System.out.println(num);
        num = num + 1;
}
```

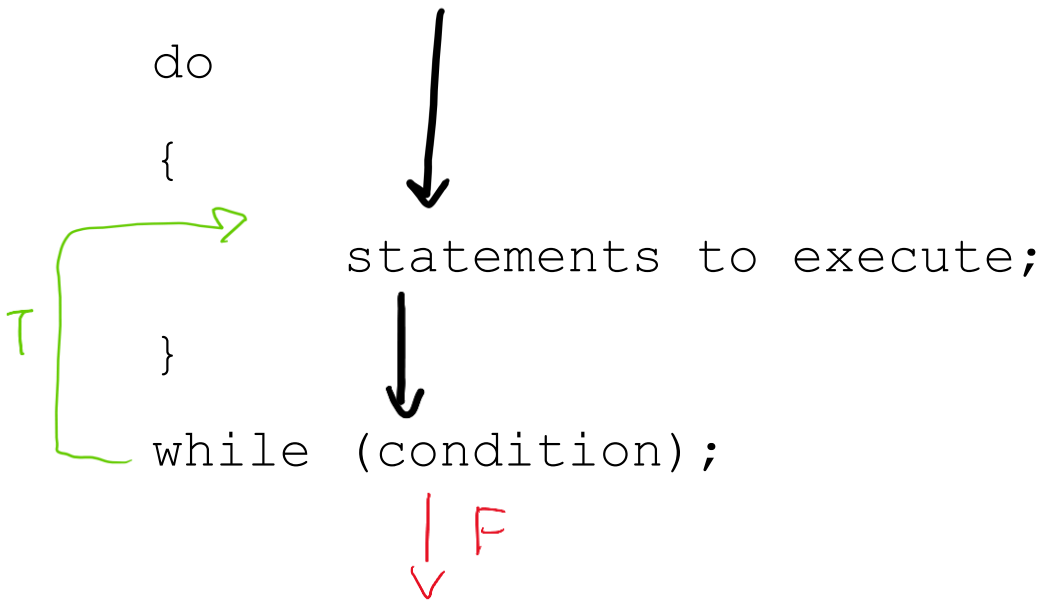
Output:

1
2
3
4

ITERATION: DO WHILE LOOPS

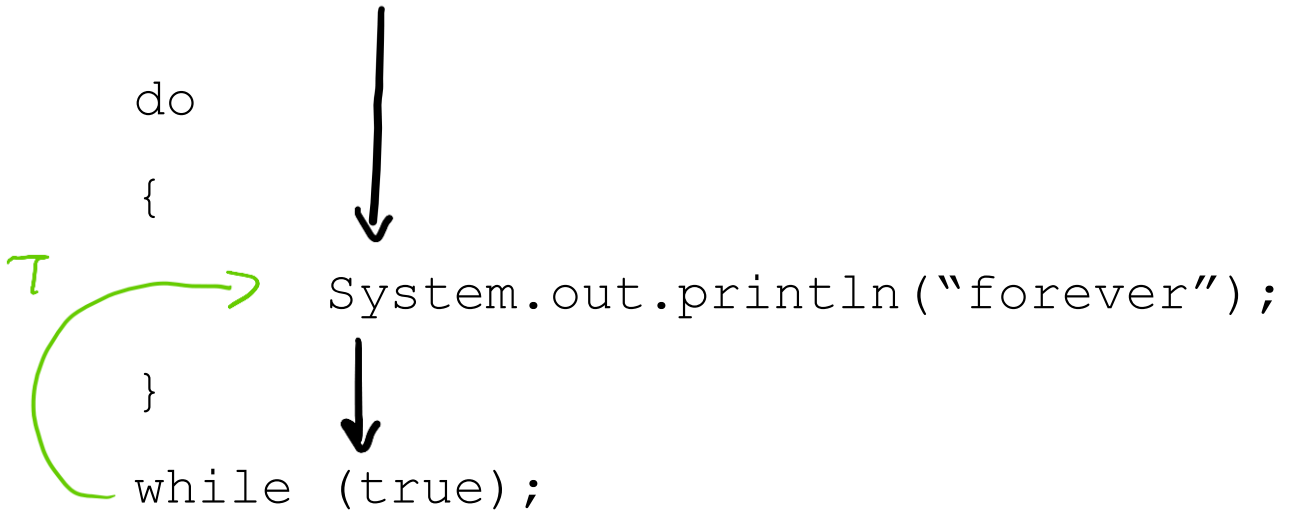
- Same as while loop but we execute our code before checking the condition

```
do
{
        statements to execute;
}
while (condition);
```



ITERATION: DO WHILE LOOPS

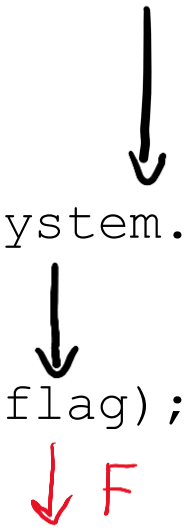
- Infinite loops are still possible



ITERATION: DO WHILE LOOPS

- It is not possible for the code in the do statement to be completely skipped

```
boolean flag = false;
do
{
      System.out.println("prints once but only once");
}
while (flag);
```



ITERATION: FOR LOOPS

- Loops with more power
- Count through iterations

```
for (initialization; condition; update)
{
      statements to execute;
}
```

ITERATION: FOR LOOPS

- Loops with more power
- Count through iterations

initialization conditional update

```
for (int i = 5; i >= 0; i--)
{
        System.out.println(i);
}
```

#1 Initialize integer i and assign the value 5 to it

#2 Check the condition
Is i >= 0?

#3 If true, print out i

#4 Update i by subtracting one

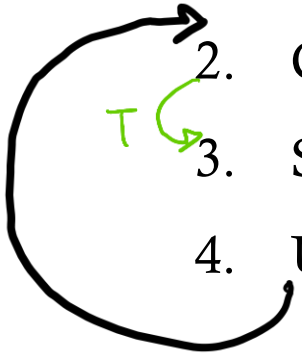
ITERATION: FOR LOOPS

```
for (int i = 5; i >= 0; i--){
        System.out.println(i);
}
```

init cond update

```
int i;
for (i = 5; i >=0; i = i - 1){
        System.out.println(i);
}
```

init assignment cond update



1. Initialization happens first
2. Conditional check happens next
3. Statements inside curly braces
4. Update executes last

IN CLASS ACTIVITY

- Countdown from 10
- For each number print “T-Minus” before the number
- Between 7 and 6, print “MAIN ENGINE START”
- After you get to 1
- Print out “LIFT OFF!!!”

LOOPS FOR INPUT

- Often getting multiple pieces of input from the user
- Re-prompting when the user enters something unexpected
- Useful methods for checking input
  - hasNext()
  - hasNextInt()
  - hasNextDouble()
  - hasNextLine()

IN CLASS ACTIVITY

- Write a loop that will continue to sum numbers that the user entered until they type done
- If they type a different string,
  - Tell them that is unexpected input
  - Prompt the user again
- Print out the sum of all the numbers the user typed

SCOPE

- Describes where a variable is accessible from
- Block scope:
  - Applies to loops and if/else if/else statements
  - If we create a variable inside the block, it will not be accessible outside the block
  - The variable only exists within that block
- If we try to use it outside its scope, we get a "cannot resolve symbol" error
- Best practice is to create and initialize variables early unless we don't want to access them elsewhere

```
int x = 1;

while (x < 5)
{
    int num = 6;
    System.out.println(x + num);
    x++;
}

// x is accessible here
// num is not out of scope here
```

FLOW CONTROL STATEMENTS

- Keywords used for flow control
  - return: used to exit out of method back to the main code (will talk more about later)
  - break: used to exit out of a loop or switch. The code immediately after the loop executes next.
  - continue: skips the rest of the loop and continues with the next loop iteration
  - try/finally blocks: used with error handling (will talk more about later)
- Generally, we aim to avoid using these
  - Easier to read/understand especially once programs get more complicated
- Loop control can be accomplished with intelligent design without using keywords.
  - Using booleans that are updated when certain conditions are met

FLOATING POINT ISSUES

- Our number system (base ten) can't accurately represent some numbers (e.g. 1/3) because they are repeating
- Same thing happens in binary (e.g. 0.1 results in repeating numbers in binary and thus can't be accurately represented)
- This can cause issues with math
- Also, why we don't use double or float values for loop control (i.e. in for loops)

FLOATING POINT ISSUES

```
for (double value = 0.0; value < 1.0; value = value + 0.1)
{
      System.out.println(value);
}
```

Output:

0.0
0.1
0.2
0.30000000000000004
0.4
0.5
0.6
0.7
0.7999999999999999...
0.8999999999999999...
0.9999999999999999...

NESTED LOOPS

- Like if statements, we can stick a loop inside another loop

```
for (int i = 0; i < 5; i++) {
        System.out.println("Outside Loop - A: " + i);
        for (int j = 0; j < 3; j++) {
                System.out.println("Inside Loop: " + j);
        }
        System.out.println("Outside Loop - B: " + i);
}
```