CompSci 201, L12: Debugging and Testing

Logistics, Coming up

- Project 3: DNA (Linked List) available, due 10/17

- Wednesday, 10/5 (today)
  - APT 5 Due

- Friday, 10/7
  - Discussion, linked list

- Monday 10/10 – Tuesday 10/11
  - Fall break, no class meeting, no helper/office hours

- Wednesday 10/12
  - APT6 (linked list) due

Today's agenda

1. Wrapping up linked list problems
    - WOTO2 from 10/3
    - Reverse in-place

2. Testing

3. Debugging

Reviewing WOTO 2 from last time

duke.is/zsttj

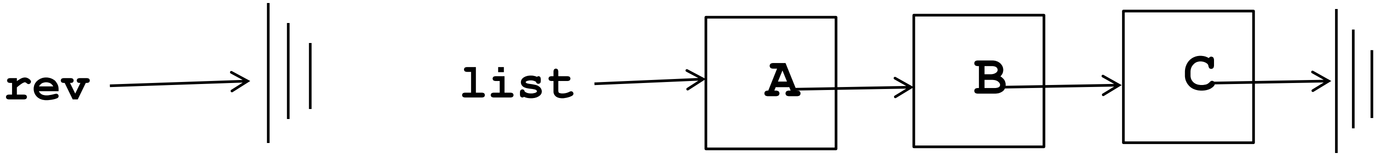
Don’t need to complete again, just reviewing answers.

Canonical Linked List Problem

- How do we reverse nodes in a linked list (without creating a new list)?
  - Go from A->B->C to C->B->A
  - Typical interview style question
  - https://leetcode.com/problems/reverse-linked-list/
  - https://www.hackerrank.com/challenges/reverse-a-linked-list

Methodical Development

- Turn list = ['A', 'B', 'C'] into
  - rev = ['C', 'B', 'A']
- Move one node at a time, ***no new nodes***!
  - Iterative/loop solution with invariant



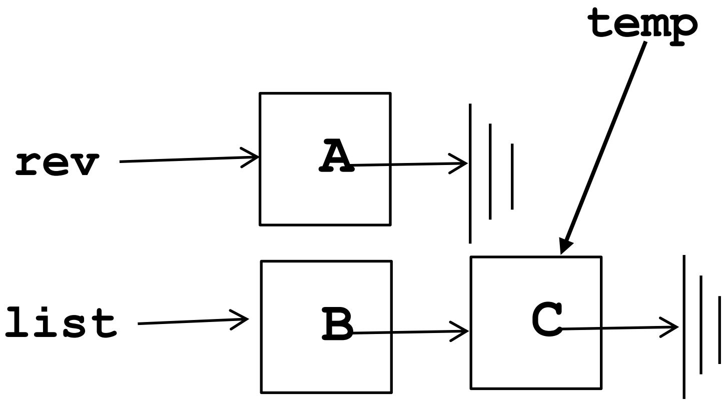
Invariant to help reason about code

- An invariant is something that is true each loop guard check (top of the loop)
  - May become false part way through loop
  - Always re-established before guard check

- rev points to list reversed so far
  - before loop iterates at all? rev = null
  - Then at the end we just return rev

one node at a time, assume invariant!

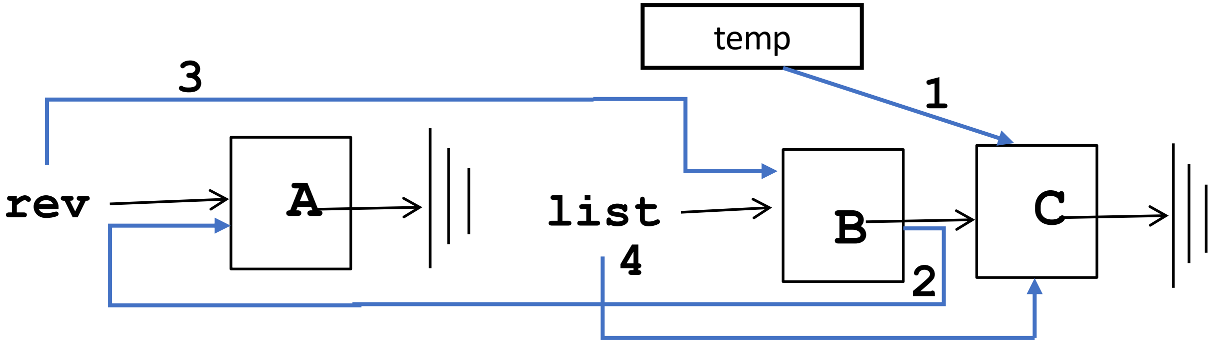
- After one iteration: rev is list reversed so far
  - list has moved to represent [B,C]
    - So rev represents [A]
- How to move B to front?
- Why temp needed?
  - Don't lose C-node!



rev = [A], list = [B,C], change: [B,A], [C]

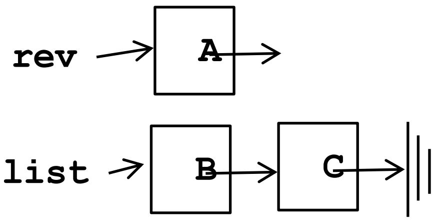
- Pictures and code

```
1.temp = list.next   (so we don't lose ['C'])
2.list.next = rev    (add to front point to ['A'])
3.rev = list         (reestablish invariant)
4.list = temp        (list updated)
```



Working code, check invariant

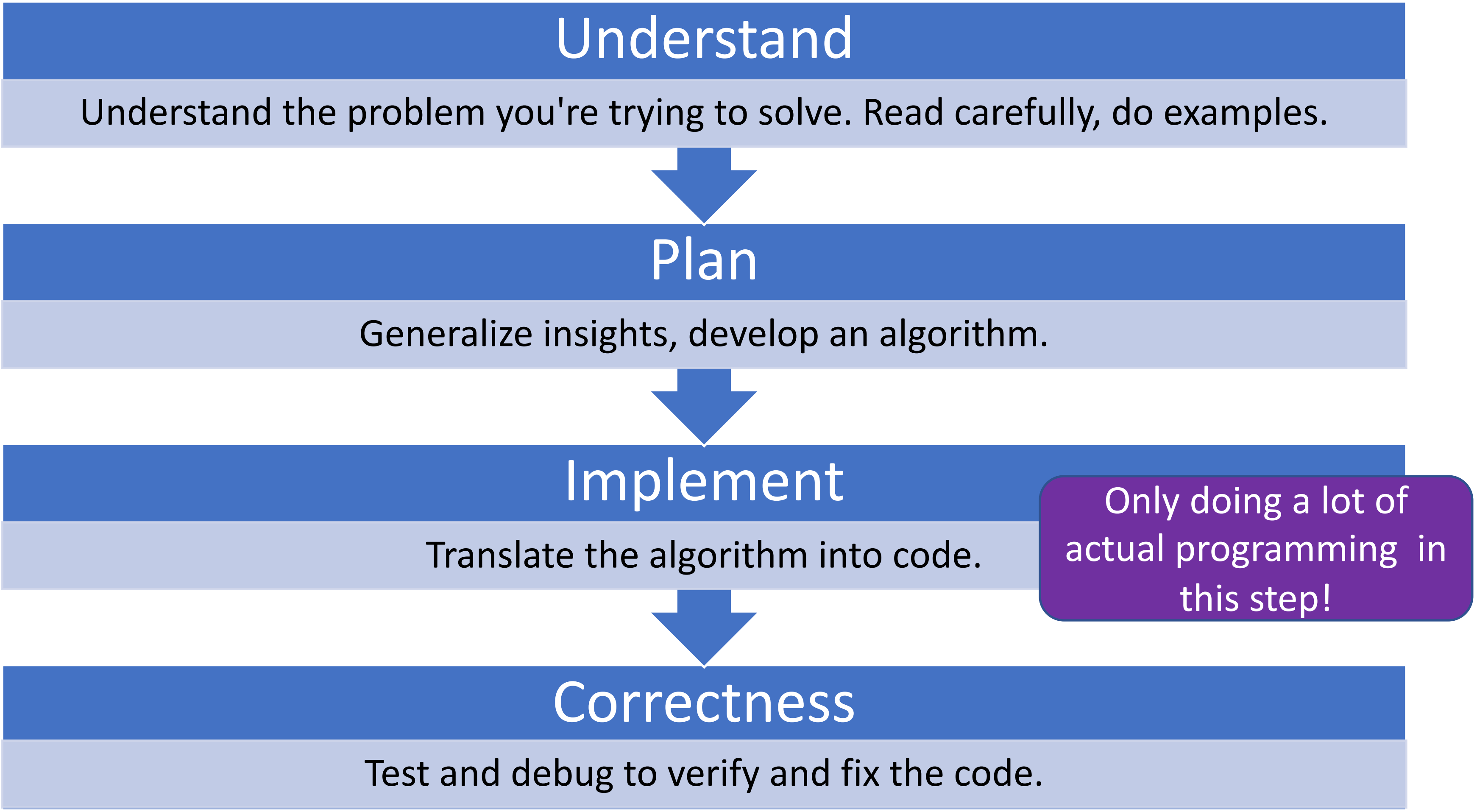
- Initialization, rev?
- Update
    - Check loop



```
public ListNode reverse(ListNode front){
    ListNode rev = null;
    ListNode list = front;
    while (list != null) {
        ListNode temp = list.next;
        list.next = rev;
        rev = list;
        list = temp;
    }
    return rev;  // like front = rev
}
```

Testing and Debugging

An Algorithmic Problem-Solving Process: UPIC



Understand

Understand the problem you're trying to solve. Read carefully, do examples.

Plan

Generalize insights, develop an algorithm.

Implement

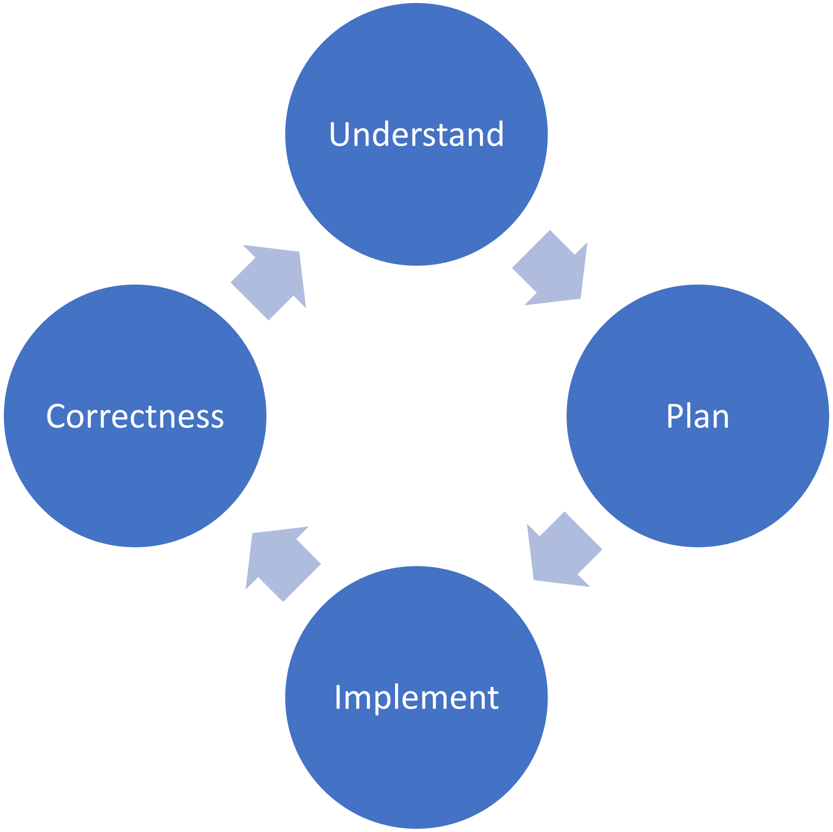
Translate the algorithm into code.

Only doing a lot of actual programming in this step!

Correctness

Test and debug to verify and fix the code.

Not really a linear process

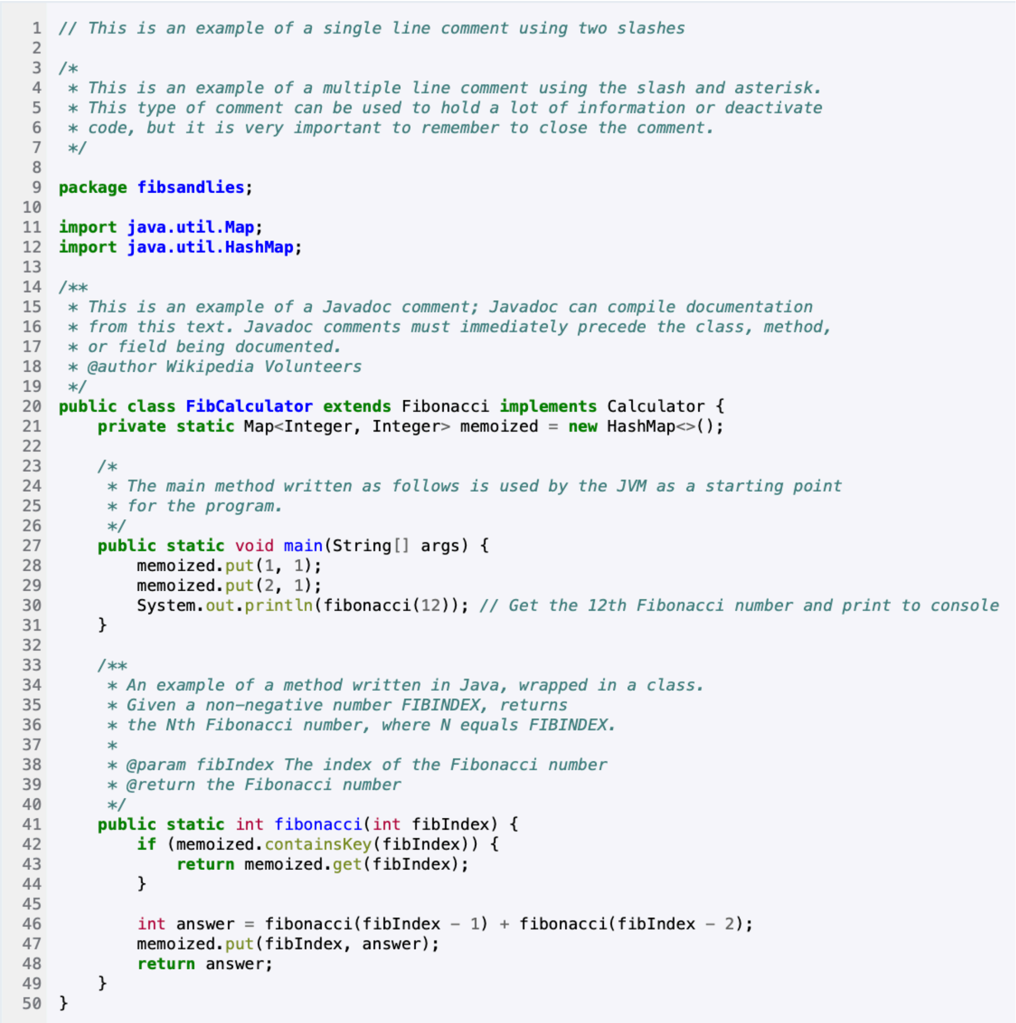


So, something is not correct. Could be…

1. My plan (algorithm) did not match my understanding.
2. My implementation does not do what I wanted my algorithm to do.
3. I did not fully understand the problem.

First approach to correctness

- Natural temptation to rely on reading source code to verify correctness.
- Like editing an essay for a class, read and check that it makes sense, look for typos.
- But…

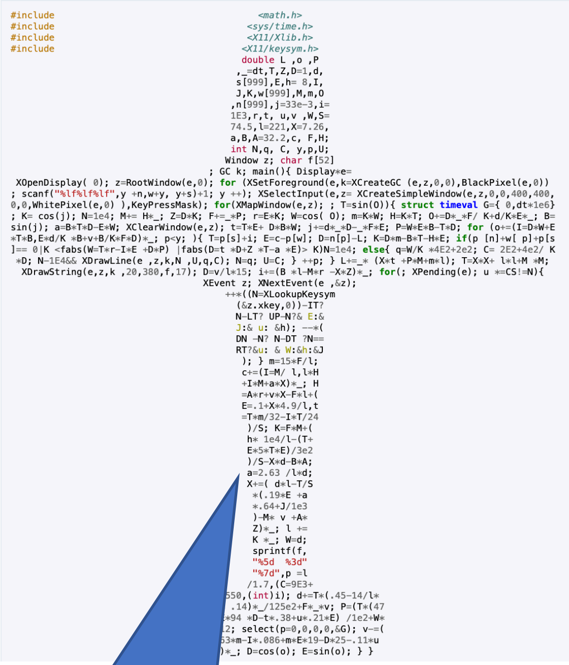


```java
// This is an example of a single line comment using two slashes

/*
 * This is an example of a multiple line comment using the slash and asterisk.
 * This type of comment can be used to hold a lot of information or deactivate
 * code, but it is very important to remember to close the comment.
 */

package fibsandlies;

import java.util.Map;
import java.util.HashMap;

/**
 * This is an example of a Javadoc comment; Javadoc can compile documentation
 * from this text. Javadoc comments must immediately precede the class, method,
 * or field being documented.
 * @author Wikipedia Volunteers
 */
public class FibCalculator extends Fibonacci implements Calculator {
    private static Map<Integer, Integer> memoized = new HashMap<>();

    /*
     * The main method written as follows is used by the JVM as a starting point
     * for the program.
     */
    public static void main(String[] args) {
        memoized.put(1, 1);
        memoized.put(2, 1);
        System.out.println(fibonacci(12)); // Get the 12th Fibonacci number and print to console
    }

    /**
     * An example of a method written in Java, wrapped in a class.
     * Given a non-negative number FIBINDEX, returns
     * the Nth Fibonacci number, where N equals FIBINDEX.
     *
     * @param fibIndex The index of the Fibonacci number
     * @return the Fibonacci number
     */
    public static int fibonacci(int fibIndex) {
        if (memoized.containsKey(fibIndex)) {
            return memoized.get(fibIndex);
        }

        int answer = fibonacci(fibIndex - 1) + fibonacci(fibIndex - 2);
        memoized.put(fibIndex, answer);
        return answer;
    }
}
```

Code is complex and interrelated

Miss something in your essay? The rest of the essay may still make sense?

One thing wrong in the code? Could prevent the whole program from functioning. And code gets complicated!



```c
#include                                      <math.h>
#include                                    <sys/time.h>
#include                                    <X11/Xlib.h>
#include                                   <X11/keysym.h>
                                           double L ,o ,P
                                          ,_=dt,T,Z,D=1,d,
                                          s[999],E,h= 8,I,
                                          J,K,w[999],M,m,O
                                         ,n[999],j=33e-3,i=
                                         1E3,r,t, u,v ,W,S=
                                         74.5,l=221,X=7.26,
                                         a,B,A=32.2,c, F,H;
                                         int N,q, C, y,p,U;
                                        Window z; char f[52]
                                     ; GC k; main(){ Display*e=
 XOpenDisplay( 0); z=RootWindow(e,0); for (XSetForeground(e,k=XCreateGC (e,z,0,0),BlackPixel(e,0))
; scanf("%lf%lf%lf",y +n,w+y, y+s)+1; y ++); XSelectInput(e,z= XCreateSimpleWindow(e,z,0,0,400,400,
0,0,WhitePixel(e,0) ),KeyPressMask); for(XMapWindow(e,z); ; T=sin(O)){ struct timeval G={ 0,dt*1e6}
; K= cos(j); N=1e4; M+= H*_; Z=D*K; F+=_*P; r=E*K; W=cos( O); m=K*W; H=K*T; O+=D*_*F/ K+d/K*E*_; B=
sin(j); a=B*T*D-E*W; XClearWindow(e,z); t=T*E+ D*B*W; j+=d*_*D-_*F*E; P=W*E*B-T*D; for (o+=(I=D*W+E
*T*B,E*d/K *B+v+B/K*F*D)*_; p<y; ){ T=p[s]+i; E=c-p[w]; D=n[p]-L; K=D*m-B*T-H*E; if(p [n]+w[ p]+p[s
]== 0|K <fabs(W=T*r-I*E +D*P) |fabs(D=t *D+Z *T-a *E)> K)N=1e4; else{ q=W/K *4E2+2e2; C= 2E2+4e2/ K
 *D; N-1E4&& XDrawLine(e ,z,k,N ,U,q,C); N=q; U=C; } ++p; } L+=_* (X*t +P*M+m*l); T=X*X+ l*l+M *M;
  XDrawString(e,z,k ,20,380,f,17); D=v/l*15; i+=(B *l-M*r -X*Z)*_; for(; XPending(e); u *=CS!=N){
                                   XEvent z; XNextEvent(e ,&z);
                                        ++*((N=XLookupKeysym
                                          (&z.xkey,0))-IT?
                                          N-LT? UP-N?& E:&
                                          J:& u: &h); --*(
                                          DN -N? N-DT ?N==
                                          RT?&u: & W:&h:&J
                                           ); } m=15*F/l;
                                           c+=(I=M/ l,l*H
                                           +I*M+a*X)*_; H
                                           =A*r+v*X-F*l+(
                                           E=.1+X*4.9/l,t
                                           =T*m/32-I*T/24
                                            )/S; K=F*M+(
                                            h* 1e4/l-(T+
                                            E*5*T*E)/3e2
                                            )/S-X*d-B*A;
                                            a=2.63 /l*d;
                                            X+=( d*l-T/S
                                             *(.19*E +a
                                             *.64+J/1e3
                                             )-M* v +A*
                                             Z)*_; l +=
                                             K *_; W=d;
                                             sprintf(f,
                                             "%5d  %3d"
                                             "%7d",p =l
                                            /1.7,(C=9E3+
                                        550,(int)i); d+=T*(.45-14/l*
                                         .14)*_/125e2+F*_*v; P=(T*(47
                                        *94 *D-t*.38+u*.21*E) /1e2+W*
                                       12; select(p=0,0,0,0,&G); v-=(
                                       63*m-I*.086+m*E*19-D*25-.11*u
                                       )*_; D=cos(o); E=sin(o); } }
```

Working C code from 1998 contest, see wikipedia

A tale of two programmers…

Too confident

“I’m amazing at programming, I don’t need to test my code because I **know** it’s correct.”

The beginning of a security vulnerability, broken app, …

Low confidence

“My code doesn’t work, that must be because I’m personally bad at this. There is no way I could figure this out myself.”

Mistaken expectations, Feeling helpless, not sure what to do

What is testing?

Verifying that an implementation *functions* as *expected*.

- What is functionality is expected?
- Given an input, what output is expected?

Can test at multiple levels: single method (*unit*), class (*integration*), whole project (*integration/functionality*), …

Black box testing (can run program, can’t see source code) and *white box testing* (access to source code).

SandwichBar APT Example

Given:

- `String[] available`, a list of ingredients the sandwich bar can use, and
- `String[] orders`, the types of sandwiches I like, in order of preference (most preferred first)

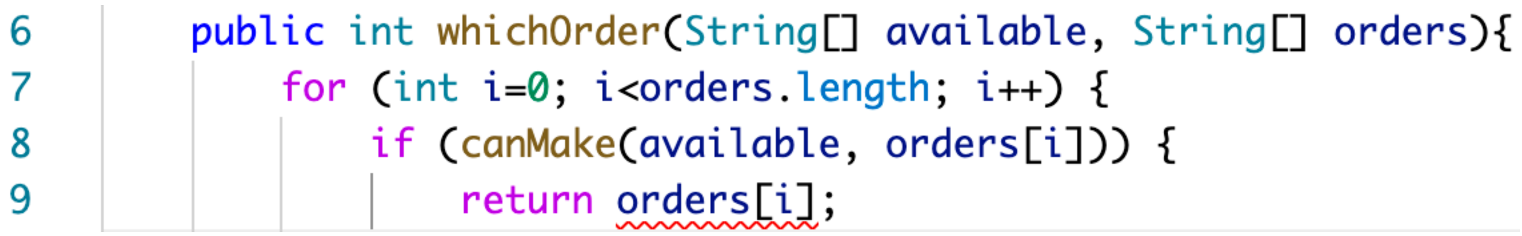
return the 0-based index of the sandwich I will buy. If the bar can make no sandwiches I like, return -1.

Example:

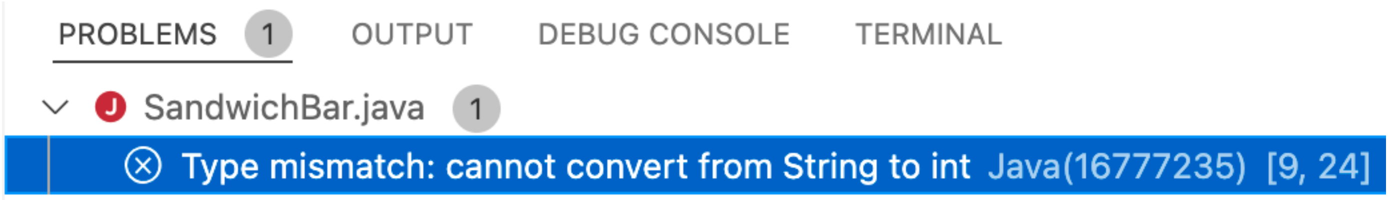
- available: { "ham", "cheese", "mustard" }
- orders: { "ham cheese" }
- Should return: 0

The first test: the compiler

- Compiler performs *static* analysis; check for errors detectable in the source code *before running*.
    - Often *type errors* (e.g., trying to assign a String to an int, trying to treat an Array as a list, …)



```
6      public int whichOrder(String[] available, String[] orders){
7          for (int i=0; i<orders.length; i++) {
8              if (canMake(available, orders[i])) {
9                  return orders[i];
```

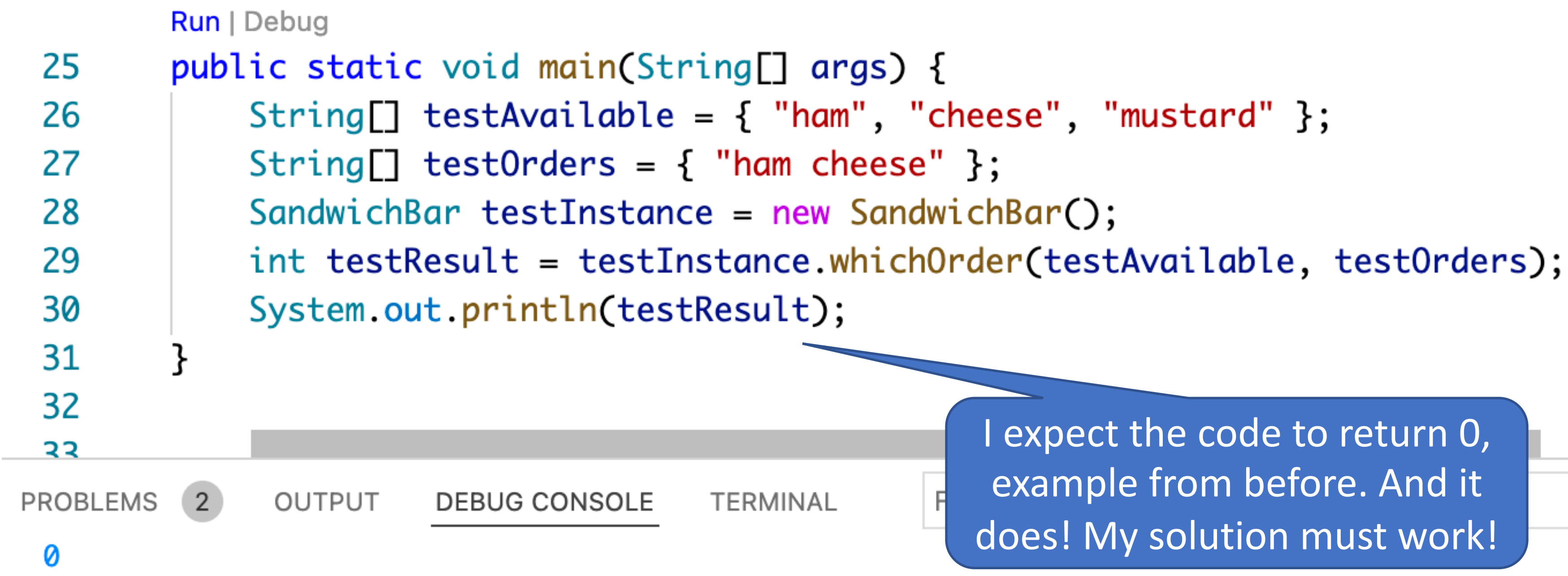


SandwichBar.java 1

Type mismatch: cannot convert from String to int Java(16777235) [9, 24]

Manual test

- Given an input, what is the expected output?
- Run program with expected input. What do you get?

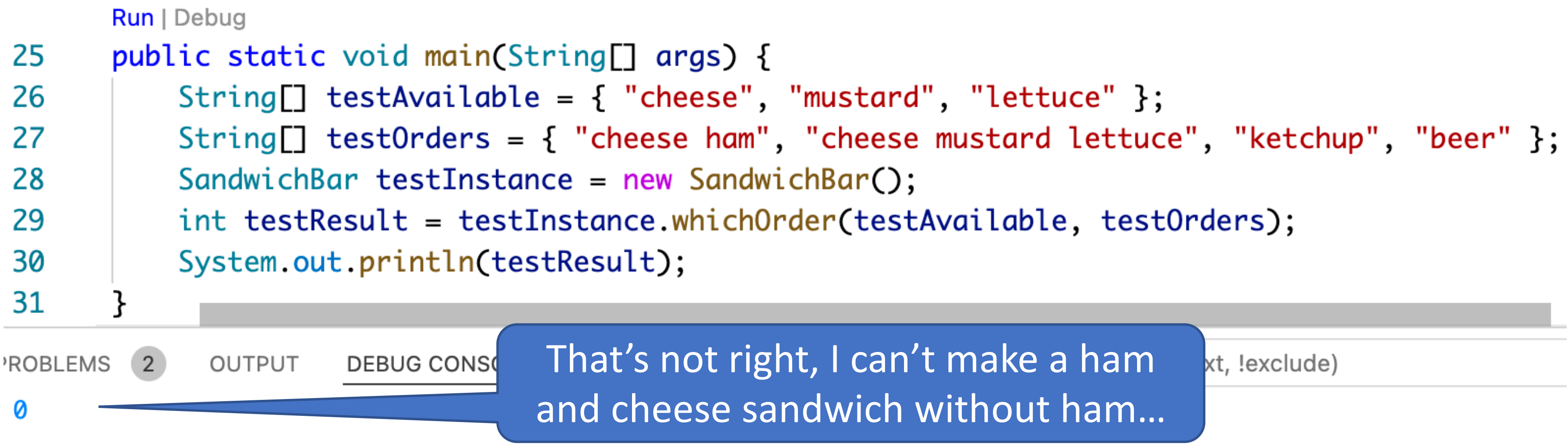


```java
public static void main(String[] args) {
    String[] testAvailable = { "ham", "cheese", "mustard" };
    String[] testOrders = { "ham cheese" };
    SandwichBar testInstance = new SandwichBar();
    int testResult = testInstance.whichOrder(testAvailable, testOrders);
    System.out.println(testResult);
}
```

```
0
```

I expect the code to return 0, example from before. And it does! My solution must work!

How many tests are enough?

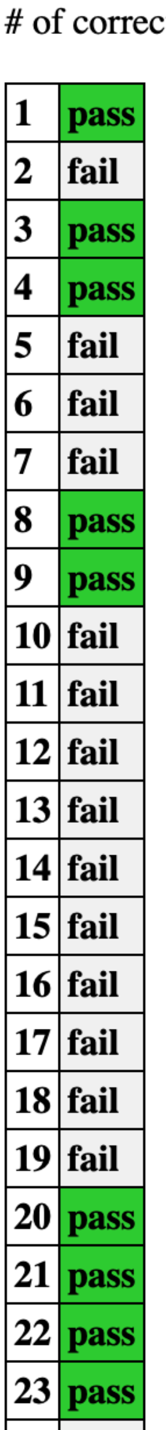


```java
public static void main(String[] args) {
    String[] testAvailable = { "cheese", "mustard", "lettuce" };
    String[] testOrders = { "cheese ham", "cheese mustard lettuce", "ketchup", "beer" };
    SandwichBar testInstance = new SandwichBar();
    int testResult = testInstance.whichOrder(testAvailable, testOrders);
    System.out.println(testResult);
}
```

```
0
```

That’s not right, I can’t make a ham and cheese sandwich without ham…

- Can never have enough tests to guarantee correctness, but…
- More and more diverse tests can help increase confidence.



# of correct: 10 out of 30

1	pass
2	fail
3	pass
4	pass
5	fail
6	fail
7	fail
8	pass
9	pass
10	fail
11	fail
12	fail
13	fail
14	fail
15	fail
16	fail
17	fail
18	fail
19	fail
20	pass
21	pass
22	pass
23	pass

Automated testing?

For when you want to run many tests without doing it manually one at a time…automate it!

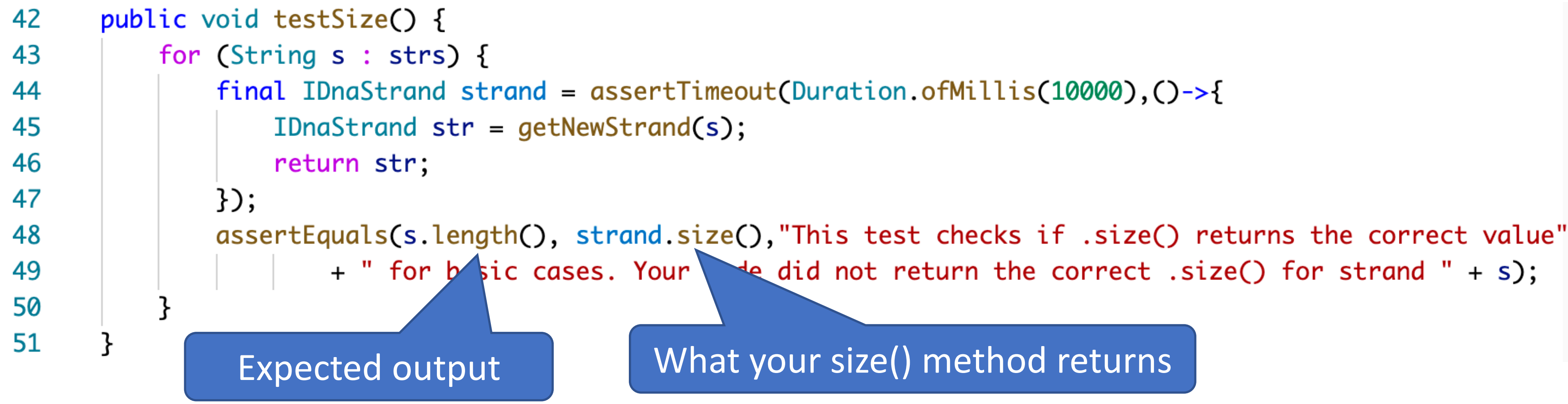
You mostly *use* automated testing in 201 rather than building it yourself:

- JUnit tests
- Gradescope autograder
- APT server

Could learn this if you want, JUnit is a general-purpose Java testing library, not just 201. Examples in projects.

Test early, test small, test often

- **Unit testing**: Term for tests conducted on the smallest *units* of code that take inputs and produce outputs.
    - In Java, typically methods, preferably short ones (10-20 lines). Test as soon as you write, don't wait!
    - Method getting too complex? Helper method!

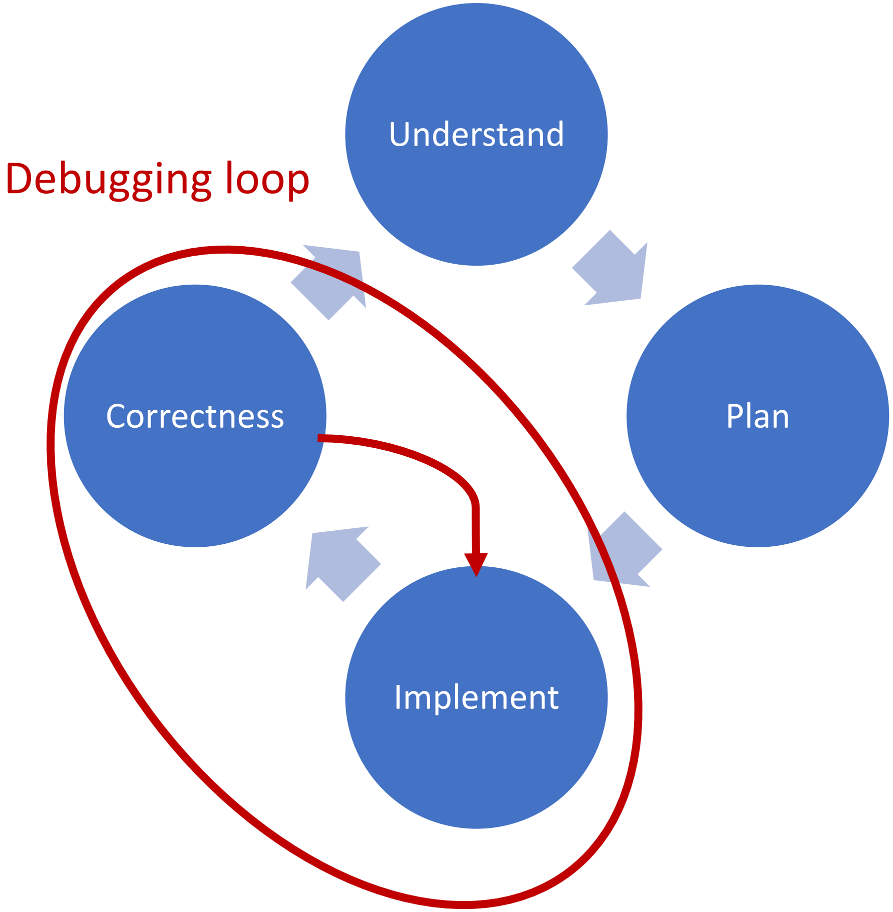


```java
public void testSize() {
    for (String s : strs) {
        final IDnaStrand strand = assertTimeout(Duration.ofMillis(10000),()->{
            IDnaStrand str = getNewStrand(s);
            return str;
        });
        assertEquals(s.length(), strand.size(),"This test checks if .size() returns the correct value"
                + " for b sic cases. Your  de did not return the correct .size() for strand " + s);
    }
}
```

Expected output

What your size() method returns

Debugging



Debugging loop:

1. Detect unexpected behavior through testing.
2. Isolate *cause* of unexpected behavior.
3. Change implementation.
4. Test again.

How to isolate the cause of unexpected behavior

- Want to identify the *first point of divergence from expected behavior.*
    - May have started long before your test result!

- Try to answer the question:
    - What is the *first* line of code in which method of which class that first did something different than I expected?
    - Never fixate on line 30 if you're not sure lines 1-29 are working.

Debugging Methods

- Three common methods:
  - Examine code and small examples by hand
  - Add print statements to code
  - Use a debugger tool

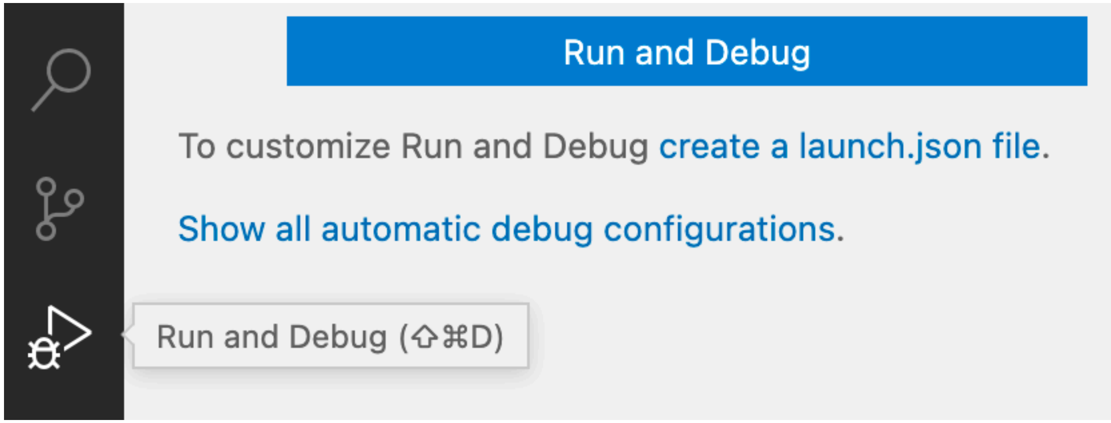
Good start, might get complicated

Allows you to see the *state* of the program while running. Tip: Can add print statements for APTs, will show up on server!

To step through execution line by line

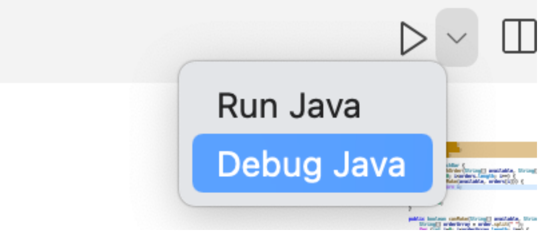
- Today we will look at the basic debugger tool built into an extension on your visual studio code.

Debugger tool



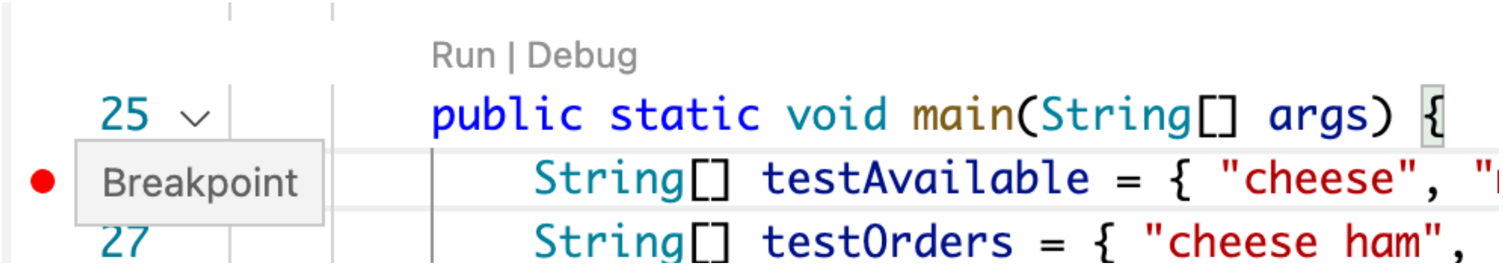
To customize Run and Debug create a launch.json file.

Show all automatic debug configurations.



- Instead of run? Choose debug!
- Walk through execution of program *line by line*.
- See current state of all variables *line by line*.

Set a breakpoint

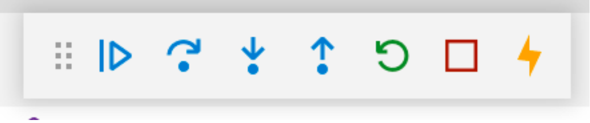


```
public static void main(String[] args) {
    String[] testAvailable = { "cheese", "
    String[] testOrders = { "cheese ham",
```

- Start by setting a *breakpoint* in your code.
- Says “run the program until the first time this line executes, then pause to step line by line.”
- If you want to go line by line from the beginning? Set to first line in main.

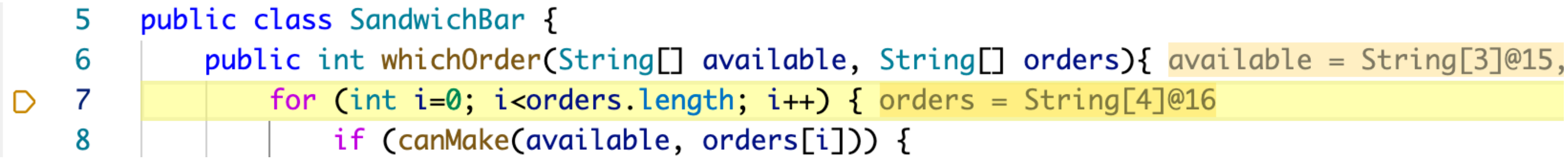
Debug options

Will see a menu like this:

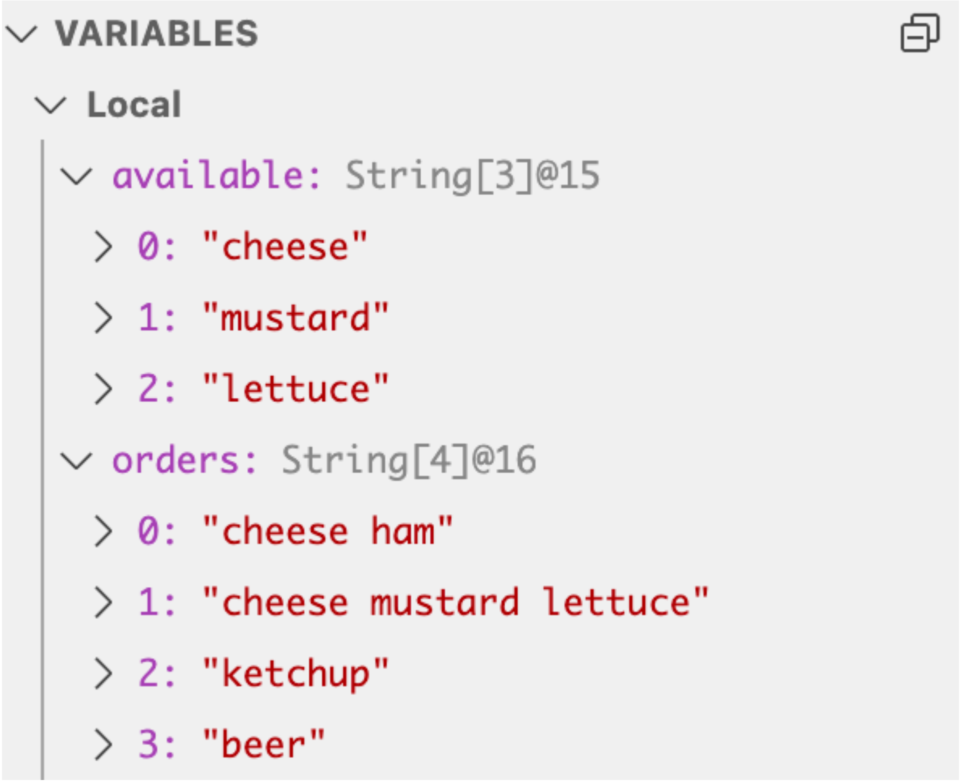


- Continue: Go to next breakpoint
- **Step over**: Execute line, go to next. Run whole methods.
- **Step into**: Same as over *unless method call*. Steps into methods, jumping to first line of method code.
- Step out: Break out of method back to where called
- Restart: Start over again at first breakpoint
- Stop: Stop debugging session

State of program



```
public class SandwichBar {
    public int whichOrder(String[] available, String[] orders){ available = String[3]@15,
        for (int i=0; i<orders.length; i++) { orders = String[4]@16
            if (canMake(available, orders[i])) {
```



```
VARIABLES
  Local
    available: String[3]@15
      0: "cheese"
      1: "mustard"
      2: "lettuce"
    orders: String[4]@16
      0: "cheese ham"
      1: "cheese mustard lettuce"
      2: "ketchup"
      3: "beer"
```

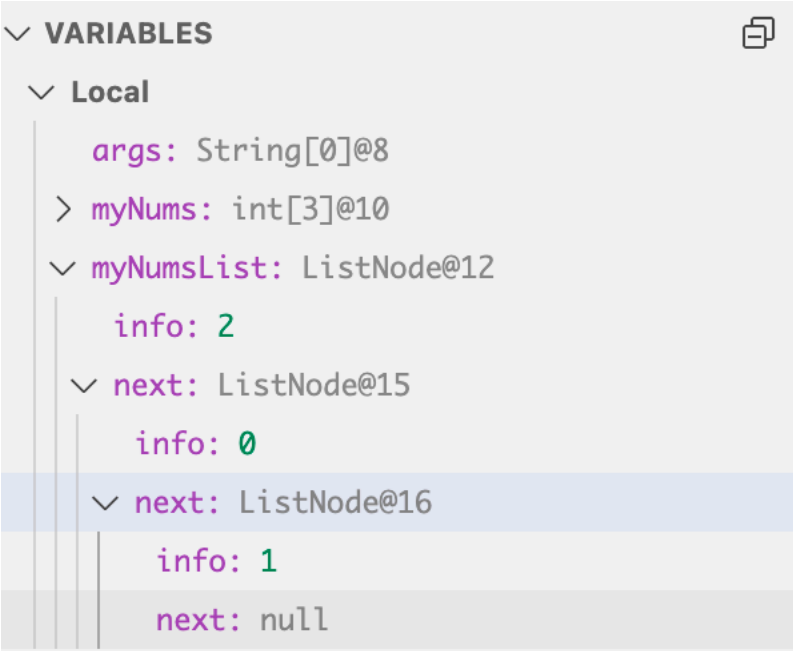
Can see all values of all local variables while executing at highlighted line.

Can step through to determine *first* time values diverge from expectations.

Testing & Debugging SandwichBar

Live coding

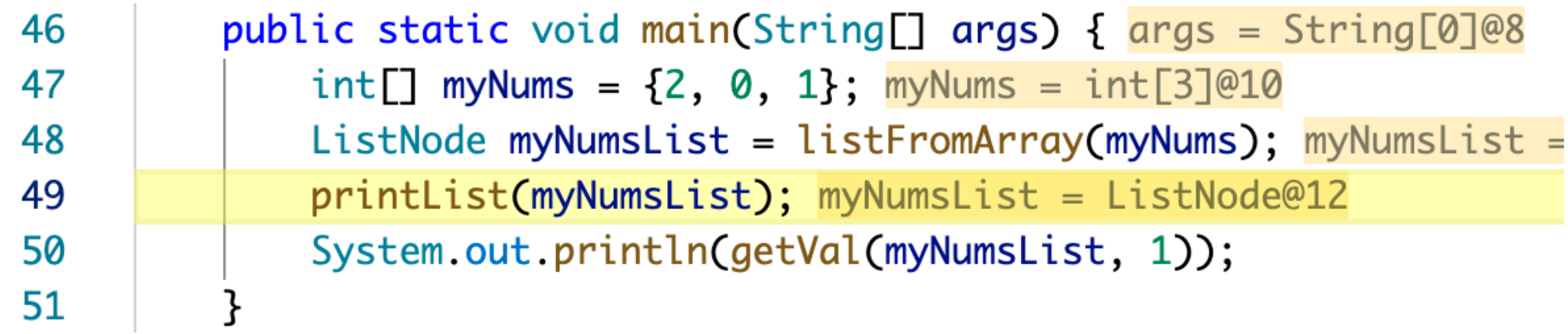
Debugging linked list?



```
VARIABLES
  Local
      args: String[0]@8
    > myNums: int[3]@10
    v myNumsList: ListNode@12
        info: 2
      v next: ListNode@15
          info: 0
        v next: ListNode@16
            info: 1
            next: null
```

- Appears as a nested “list” of object references.

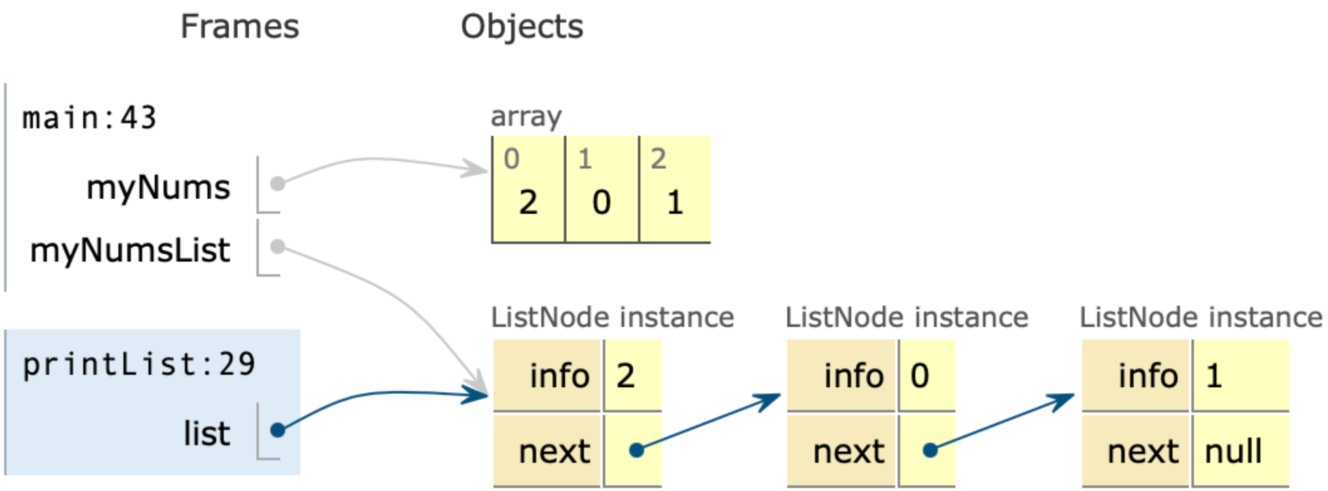
- Expand one node at a time.



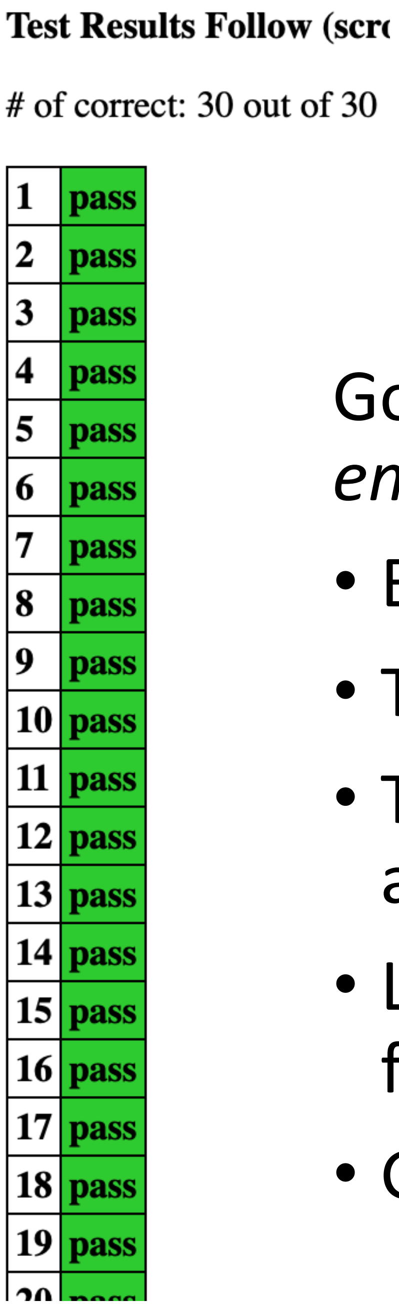
```
    public static void main(String[] args) { args = String[0]@8
        int[] myNums = {2, 0, 1}; myNums = int[3]@10
        ListNode myNumsList = listFromArray(myNums); myNumsList =
        printList(myNumsList); myNumsList = ListNode@12
        System.out.println(getVal(myNumsList, 1));
    }
```

Want something more visual?

pythontutor.com/java.html



Can use if you need to visualize stepping through some pointer code.



Test Results Follow (scr

# of correct: 30 out of 30

1	pass
2	pass
3	pass
4	pass
5	pass
6	pass
7	pass
8	pass
9	pass
10	pass
11	pass
12	pass
13	pass
14	pass
15	pass
16	pass
17	pass
18	pass
19	pass
20	pass

Debugging reflection

Goal is to become a more *active* and *empowered* tester and debugger.

- Build evidence the code *as you develop*.
- Take *active* steps to isolate the problem
- Test, use the debugger, gather data, reason about it
- Less time staring at the code, feeling frustrated
- Gain confidence, gain independence

Person in CS: Barbara Liskov

- Turing Award Winner in 2008 for contributions to practical and theoretical foundations of programming language and system design, especially related to data abstraction, fault tolerance, and distributed computing.
- “The advice I give people in general is that you should **figure out what you like to do**, **and what you can do well**—and the two are not all that dissimilar, because you don’t typically like doing something if you don’t do it well. ... So you should instead watch—**be aware of what you’re doing, and what the opportunities are, and step into what seems right**, and see where it takes you.”