

Welcome!

Whole Program Assertions

CSE 331: Software Design and Implementation, Autumn 2026

Working means meeting the specification, on every run.

Welcome! Stoked for this quarter 🎉

Ten weeks to learn software design and implementation.

Our motto:

Your job is to deliver code that you know works.

What is going to happen?

- Bad course summary:
 - "Learn more Java, and some JavaScript"
- Better course summary:
 - Say what code must do: specifications
 - Structure and design programs
 - Know code is correct: testing and proof
 - Build event-driven programs, like user interfaces

Today

1. Course mechanics
2. Our first program

TODO

- Read the [course website](#), especially the [syllabus](#)
- Do [Homework 0](#) and [Reading 1](#)
 - Both due Friday, October 2, at 11:59pm
- Come to your first section, Tuesday, October 6
- Sign in to the [message board](#), on Ed

About us



Zach Tatlock

[PLSE](#): compilers, e-graphs, numerics, fabrication
Helps organize and lead [Race Condition Running](#)
Listens to [Elder](#), watches [Blondihacks](#) machining
Reads [Greg Egan](#) novels, practices [mindfulness](#)



James Wilcox

Languages, verification, distributed systems
Likes puzzles, especially: is this code correct?
Running and [singing](#), but not at the same time
Good coffee, small planes; no cars of any size

Our goal is to help you become great computer scientists.

Our TAs

Our TAs make this course run! They:

- Teach sections every Tuesday
- Hold office hours each week
- Answer your questions on Ed
- Grade your work and give you feedback

You'll meet your TAs in section on Tuesday

Learn more about them on the [staff page](#)

Lecture and section

- Lectures are recorded, on Panopto via Canvas
- Most have an activity: talk, then write on paper
 - Graded on completion: hand it in at the door
- Sections are TA-led, on Tuesdays, and not recorded
- Take notes on paper 

Using AI

- Use it to learn: explanations, questions, practice
- On graded work, only where the [syllabus](#) allows:
 - Reading reflections: light editing only, and say so
 - Programming tasks whose handout says so
- No AI on Homework 0 or Homework 1
 - AI built into your editor counts
- Exams set your grade, and you take them on your own

Questions about mechanics?

Our motto

Your job is to deliver code that you know works.

Your job: can a tool do it for you?

Deliver: to whom?

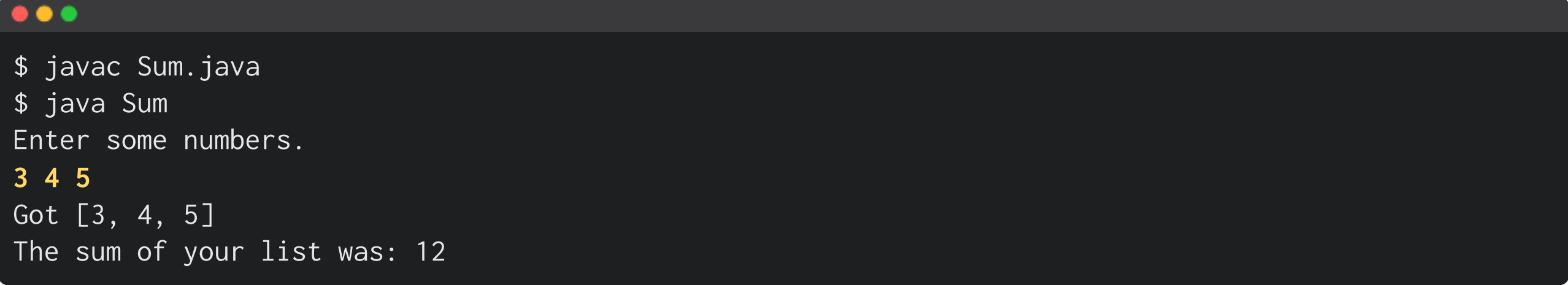
Know: how?

Works: meaning what?

What does this program do?

```
5  public class Sum {
6      public static void main(String[] args) {
7          Scanner in = new Scanner(System.in);
8          System.out.println("Enter some numbers.");
9          List<Integer> list = new ArrayList<>();
10         while (in.hasNextInt()) {
11             list.add(in.nextInt());
12         }
13         // (1)
14         System.out.println("Got " + list);
15         int sum = 0;
16         for (int i = 0; i < list.size(); i++) {
17             sum += list.get(i);
18         }
19         // (2)
20         System.out.println("The sum of your list was: " + sum);
21     }
22 }
```

Let's run it



```
$ javac Sum.java
$ java Sum
Enter some numbers.
3 4 5
Got [3, 4, 5]
The sum of your list was: 12
```

End the input with Control-D; on Windows, Control-Z, then Enter.

A word among the numbers



Enter some numbers.

1 2 three 4

Got [1, 2]

The sum of your list was: 3

Wrong about this run, or silent?

Nothing at all



```
Enter some numbers.  
Got []  
The sum of your list was: 0
```

Type nothing; end the input at once.

The sentence is true of this run, if an empty list has a sum.

Two numbers



```
Enter some numbers.
```

```
2147483647 1
```

```
Got [2147483647, 1]
```

```
The sum of your list was: -2147483648
```

A Java `int` holds only -2^{31} to $2^{31} - 1$,

that is, -2147483648 to 2147483647.

The sentence promised the sum.

Every run

- **Run:** one execution of a program, on one input
 - 3 4 5 is one, 1 2 3 another, nothing at all a third
- Our sentence is a claim about **every** run
 - One run can confirm it only for that run
 - One run where it is false refutes it: a **counterexample**

Repairing the sentence

The program asks for numbers.

It reads whole numbers between -2^{31} and $2^{31} - 1$ from the user, in order, until the input ends or the next thing typed is not such a number; anything from that point on is ignored. It prints the numbers it read, then their sum, wrapped around into that same range if the true sum falls outside it.

1 2 three 4: reads 1 and 2, ignores the rest

Nothing typed: Got [] and a sum of 0

2147483647 1: the sum wraps to -2147483648

2147483648 alone: too big for an int, so nothing is read

What a specification is

A **specification** says what a piece of code *must* do on *every run*, as seen from *outside* it.

It says nothing about how the code does it.

One run where it is false refutes it

It never mentions `list`, `sum`, or `line 15`

The code meets it or fails it

- For a whole program, outside means the **user**
 - Its specification is a precise version of the **user manual**

What is true at (1) and (2)?

```
10  while (in.hasNextInt()) {
11      list.add(in.nextInt());
12  }
13  // (1)
14  {{ ? }}
15  System.out.println("Got " + list);
16  int sum = 0;
17  for (int i = 0; i < list.size(); i++) {
18      sum += list.get(i);
19  }
20  // (2)
21  {{ ? }}
22  System.out.println("The sum of your list was: " + sum);
```

Half sheet, question 2: one claim at each point, true on every run.

What an assertion is

- **Assertion:** a true or false claim about the program's state
 - Its variables and memory, at one point in the program
- It **holds** if the claim is true there on every run
 - One run where it is false, and it does not hold
- Not Java's `assert`, a print, or a test
 - Those check only the runs you try

Sum with assertions

```
10 while (in.hasNextInt()) {
11     list.add(in.nextInt());
12 }
13 // (1)
    {{ list holds, in order, every number the user typed before
        the input ended or something that was not an int appeared }}
14 System.out.println("Got " + list);
15 int sum = 0;
16 for (int i = 0; i < list.size(); i++) {
17     sum += list.get(i);
18 }
19 // (2)
    {{ sum is the sum of the elements of list, wrapped into the
        int range if the true sum falls outside it }}
    {{ list still holds, in order, every number the user typed
        before the input ended or something that was not an int
        appeared }}
20 System.out.println("The sum of your list was: " + sum);
```

Outside and inside

- **Specification:** what the user will see
 - Outside the program, on every run
- **Assertions:** why the code delivers it
 - Inside the program, at one point, on every run
 - The user never sees them

Reasoning between assertions

```
{ { P } }  
some code  
{ { Q } }
```

- Before: P holds
- The code: takes any P state to a Q state
- After: Q holds
- Needs only P : whatever came before is summarized by P

The edge of the program

If line 10 used `hasNext` instead of `hasNextInt`:



Enter some numbers.

3 4 hello

```
Exception in thread "main" java.util.InputMismatchException
    at java.base/java.util.Scanner.throwFor(Scanner.java:977)
    at java.base/java.util.Scanner.next(Scanner.java:1632)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2297)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2251)
    at Sum.main(Sum.java:11)
```

A message meant for someone else

Our standard, all quarter

- **Every input** a user can produce gets a response
 - One the specification describes
- A **stack trace** is never that response
 - It is a message meant for someone else

How do you come to know?

- **Testing:** 3 4 5, a word, nothing, the top of the range
 - Can refute a claim about every run, and never confirm it
- **Tools:** the type checker ran when we compiled
- **Reasoning:** what is true at (1) and (2)
 - Can confirm a claim about every run, if the argument is right
 - An argument can be wrong
 - A test that refutes the claim refutes the argument too

Our motto, again

Your job is to deliver code that you know works.

Your job: not the compiler's, a teammate's, or a tool's

Deliver: to people who type things you did not imagine

Know: reasons, from tests, from tools, and from arguments

Works: meets its specification, on every run

Coming up

- Friday: [Sum](#), split into methods
- Its assertions become promises between them
- Homework 0 and Reading 1: due Friday, 11:59pm
- Hand in your half sheet at the door