

CS 600.226: Data Structures

Michael Schatz

Sept 5, 2018

Lecture 3: Introduction to Interfaces



Agenda

- 1. Quick Review**
- 2. *Introduction to Java Interfaces***
- 3. *Introduction to Generics, Exceptions and Arrays***

Welcome!

Course Webpage:

<https://github.com/schatzlab/datastructures2018>

Course Discussions:

<https://piazza.com/jhu/fall2018/600226/home>

Office Hours:

Wednesday @ 2:45pm – 4pm, Malone 323
CA office hours throughout the week ☺

Programming Language:

Java with Checkstyle and JUnit
Virtual Machine (Lubuntu) or CS acct.

Accounts for Majors (CS/CE) & Minors:

If you do not already have a personal CS departmental unix account, please complete an account request form ASAP. Check "Linux Undergrad" for account type. (Note - must be declared to be eligible.)

Accounts for Others:

We will need to make accounts. Do people need them?

CS Lab access:

Students must see Steve DiBlasio, with your J-card, in Malone G61A to get CS Lab access. The CS Lab is Malone 122 and that's where course TA/CAs will be available for help.

Piazza! Lecture Notes! Q&A!

The screenshot shows a web browser window with several tabs open. The main content is a Piazza course page for '600.226' titled 'PeterLectureNotes.pdf'. The page displays a note titled 'Lecture notes' by 'Mike' with 0 views. The note content is as follows:

Here are the lecture notes from previous versions of this class taught by Dr. Peter Froehlich. We won't be following this exactly as presented, but this can be a very useful resource with additional examples and discussion of the topics we do cover in class. These are provided as-is, although if you spot any typos let me know and I can forward on to Peter.

[PeterLectureNotes.pdf](#)

Good luck!

Mike

logistics

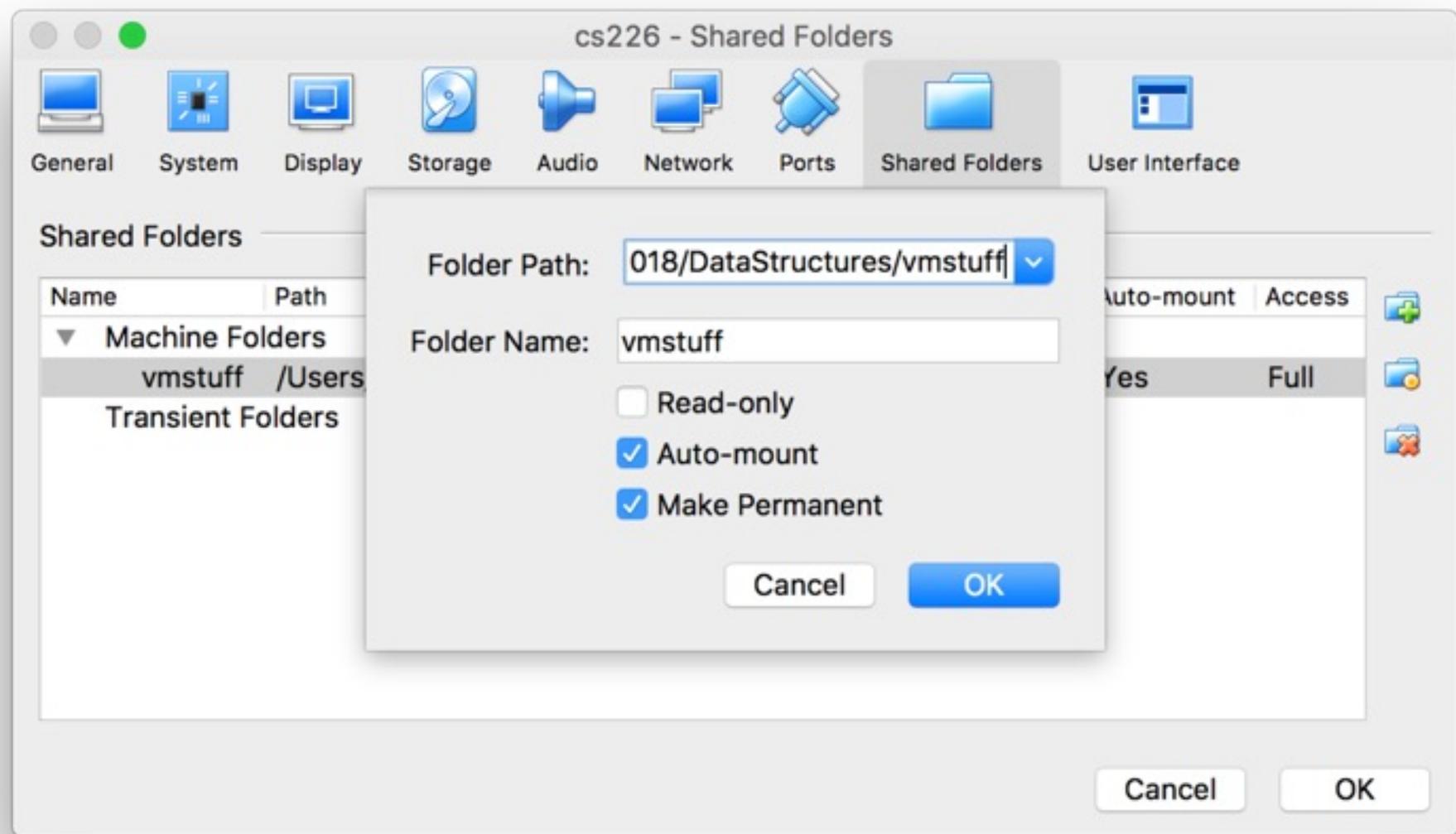
The sidebar on the left shows a list of posts categorized by time: PINNED, FAVORITES, TODAY, and LAST WEEK. The 'TODAY' section contains a post from 'Mike' titled 'Lecture notes' which is highlighted in yellow. Other visible posts include 'Office Hours', 'Resources & Links', 'Welcome to Piazza!', 'Pre-Req Bootcamp Thursday...', and 'PPT From Yesterday?'. The bottom right corner of the page shows statistics: Online Now (9), This Week (111), and Copyright information: © 2018 Piazza Technologies, Inc. All Rights Reserved. Privacy Policy, Copyright Policy, Terms of Use, Blog, Report Bug!

VirtualBox

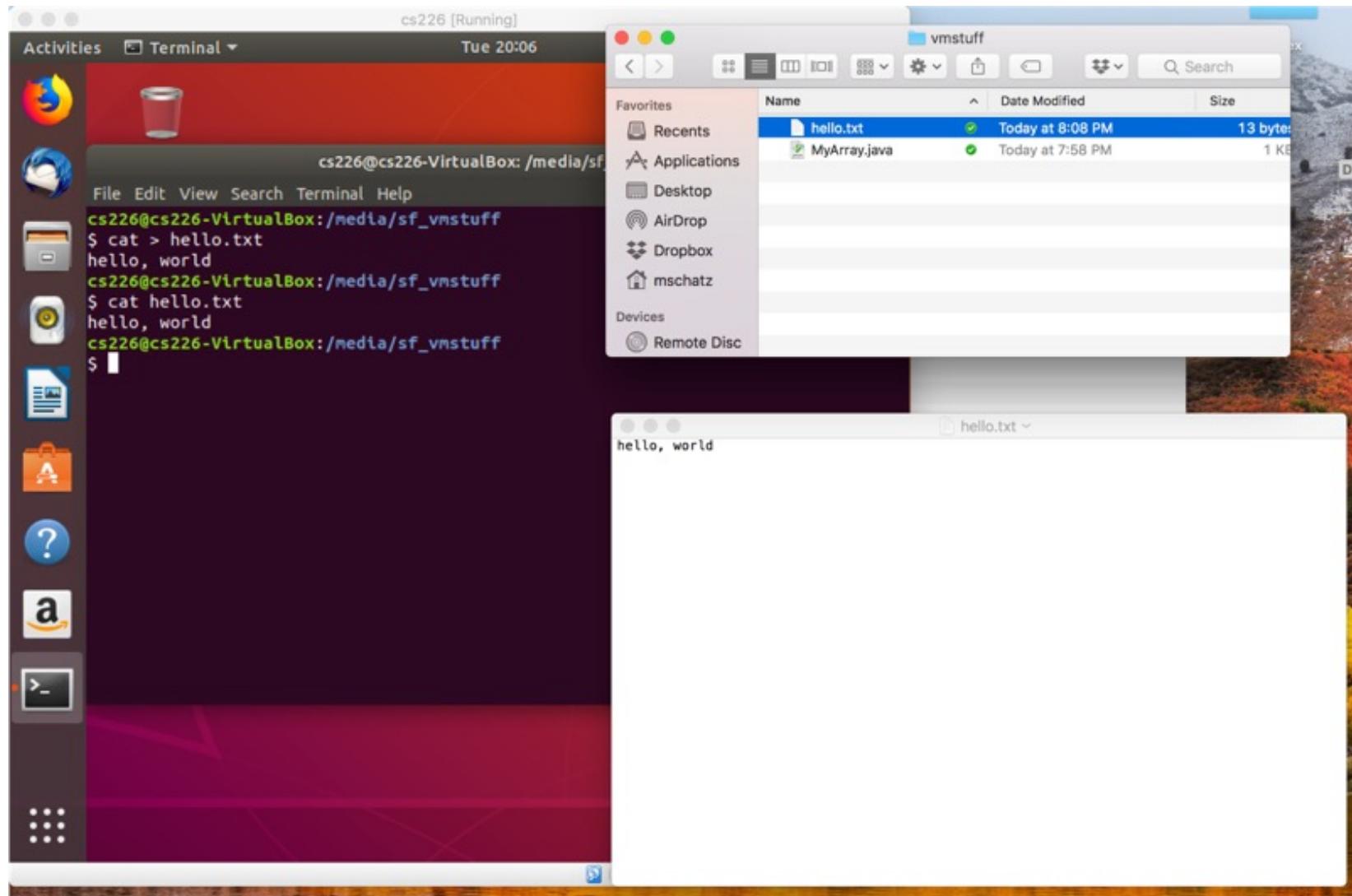


- Client application available for Mac, Windows, Linux
- Available to run our reference virtual machine running linux
 - Guaranteed that your development environment matches testing environment
 - Make sure to install the Extension Pack and Guest Additions too

VirtualBox Shared Folders



VirtualBox Shared Folders



```
$ sudo usermod -aG vboxsf cs226
$ /sbin/shutdown -r now
```

Java Environments

Command Line Everything



```
1. vim
public class SimpleCounter implements Counter {
    // current value of the counter.
    private int value;

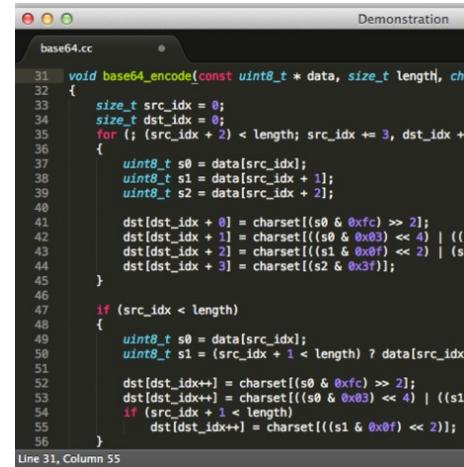
    /**
     * Simple assert-based unit tests for this counter.
     *
     * Make sure you run SimpleCounter with -enableassertions!
     * We'll learn a much better approach to unit testing later.
     */
    public static void main(String[] args) {
        Counter c = new SimpleCounter();
        assert c.value() == 0;
        System.out.println("Counter is now: " + c.value());
        c.up();
        assert c.value() == 1;
        System.out.println("Counter is now: " + c.value());
        c.down();
        assert c.value() == 0;
        System.out.println("Counter is now: " + c.value());
        c.up();
        c.up();
        c.up();
        System.out.println("Counter is now: " + c.value());
        assert c.value() == 2;
    }
}
```

\$ vim HelloWorld.java

\$ javac HelloWorld.java
\$ java HelloWorld

Universal, fast, flexible
Steep learning curve

GUI Editor + Command Line



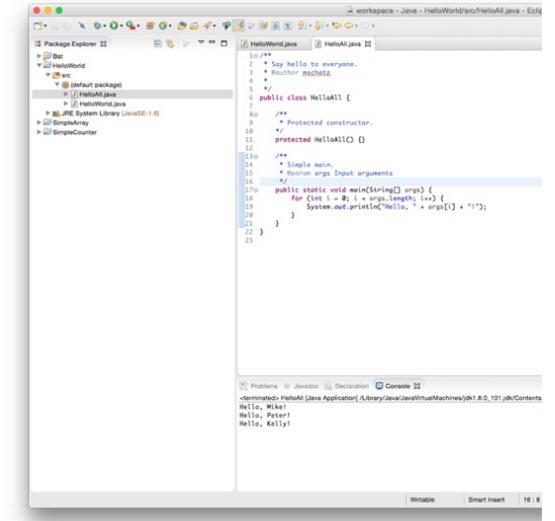
```
Demonstration
base64.cc
31 void base64_encode(const uint8_t * data, size_t length, char
32 {
33     size_t src_idx = 0;
34     size_t dst_idx = 0;
35     for (; (src_idx + 2) < length; src_idx += 3, dst_idx += 4)
36     {
37         uint8_t s0 = data[src_idx];
38         uint8_t s1 = data[src_idx + 1];
39         uint8_t s2 = data[src_idx + 2];
40
41         dst[dst_idx + 0] = charset[((s0 & 0xfc) >> 2)];
42         dst[dst_idx + 1] = charset[((s0 & 0x03) << 4) | ((s1
43         dst[dst_idx + 2] = charset[((s1 & 0x0f) << 2) | (s2
44         dst[dst_idx + 3] = charset[((s2 & 0x3f)];
45
46         if (src_idx < length)
47         {
48             uint8_t s0 = data[src_idx];
49             uint8_t s1 = (src_idx + 1 < length) ? data[src_idx +
50
51             dst[dst_idx++] = charset[((s0 & 0xfc) >> 2)];
52             dst[dst_idx++] = charset[((s0 & 0x03) << 4) | ((s1 &
53             if (src_idx + 1 < length)
54                 dst[dst_idx++] = charset[((s1 & 0x0f) << 2)];
55         }
56     }
Line 31, Column 55
```

Sublime Text

\$ javac HelloWorld.java
\$ java HelloWorld

Nearly universal, flexible
Moderate learning curve

Integrated Development Environment (IDE)



```
workspace - Java - HelloWorld/src/HelloAll.java - Eclipse
HelloAll.java
1 /**
2  * Say Hello to everyone.
3  */
4
5
6 public class HelloAll {
7
8     /**
9      * Protected constructor.
10     */
11     protected HelloAll() {
12
13     /**
14      * Simple main.
15      */
16     public static void main(String[] args) {
17         for (int i = 0; i < args.length; i++) {
18             System.out.println("Hello, " + args[i] + "!");
19         }
20     }
21
22 }
```

Hello, Mike!
Hello, Peter!
Hello, Kelly!

Eclipse / IntelliJ

Most Support
Most “magical”

Code may not work
during grading ☹

Bootcamp: Thursday @ 5:30 Malone 228

Inbox (1) - michael.schatz@gmail.com > Google News > 600.226 > Michael

Secure | https://piazza.com/class/jl345xehd4w3w9?cid=12

JHUMail Daily SL cschl jhu Media shop edit Rm Cookies Remove NYT Cooki... Other Bookmarks

piazza 600.226 Q & A Resources Statistics Manage Class Michael Schatz

hw1 hw2 hw3 hw4 hw5 logistics

Unread Updated Unresolved Following

New Post Search or add a post...

PINNED

9/1/18 Instr Office Hours This thread can be used for office hour updates, so it might be worthwhile to check here if you plan on coming to office

8/31/18 Instr Resources & Links Here is a list of several helpful links for the class so you can get to them easily from piazza. You can also always go

FAVORITES

8/20/18 Instr Welcome to Piazza! Students, Welcome to Piazza! We'll be conducting all class-related discussion here this term. The quicker you begin a

TODAY

12:14PM Instr Pre-Req Bootcamp Thursda... Hi Guys, So I got Malone 228 reserved for us for this Thursday, 9/6, from 5:30-6:30. I'll run through a powerpoint

LAST WEEK

Sat 12:14PM Private PPT From Yesterday? Happy Labor Day Weekend! I wanted to make sure I set up my Virtual Box Linux environment correctly by going through the

Fri 12:14PM Instr Pre-Req Bootcamp UPDATE: The below has been confirmed! -- UPDATE: This is tentatively planned for Thursday 9/6, 5:30-6:30 in Malone 2

WEEK 8/19 - 8/25

8/20/18 Welcome to Piazza! Piazza is a Q&A platform designed to get you great answers from classmates and instructors fast. We've put together thi

Note History:

note 13 views

Pre-Req Bootcamp Thursday 9/6

Hi Guys,

So I got Malone 228 reserved for us for this Thursday, 9/6, from 5:30-6:30. I'll run through a powerpoint with some general notes for developing in this course, as well as a quick Java refresher - covering most of the questions I remember having when I took this course. But, if you have anything specific you want to discuss, feel free to follow up to this post or send me a private message.

See you then.

-Tim

logistics

edit good note 0 Updated 2 hours ago by Tim Kutcher

followup discussions for lingering questions and comments

Resolved Unresolved

Anonymous 2 hours ago Could you email the PowerPoint out, for those of us that won't be able to attend?

Tim Kutcher 2 hours ago Yep! I'll have the final version posted in the class repository (and maybe add some notes for any questions asked afterward) for you all to retrieve it.

Reply to this followup discussion

Start a new followup discussion

Compose a new followup discussion

Agenda

- 1. Quick Review**
- 2. *Introduction to Java Interfaces***
- 3. *Introduction to Generics, Exceptions and Arrays***

Interfaces



Introduction to Java Interfaces

Objects define their interaction with the outside world through the methods that they expose. Methods form the object's interface with the outside world; the buttons on the front of your television set, for example, are the interface between you and the electrical wiring on the other side of its plastic casing. You press the "power" button to turn the television on and off. [...] **An interface is a group of related methods with empty bodies.**

<https://docs.oracle.com/javase/tutorial/java/concepts/interface.html>



```
interface Counter {  
    int value();  
    void up();  
    void down();  
}
```

Interfaces Review

- **Interfaces establish the “contract” between the implementation and any potential client code**
 - Helps to abstract our the key features of a data structure
 - You can trivially replace the use of one implementation with another as long as they implement the same interface
 - Teams can work on different pieces of a large system knowing that everything will work together in the end
- **Java Interfaces are groups of related methods with empty bodies**
- Defines the syntax of what is available

```
interface Counter {  
    int value();  
    void up();  
    void down();  
}
```

Whats wrong with this interface?

Interfaces Review

- **Interfaces establish the “contract” between the implementation and any potential client code**
 - Helps to abstract our the key features of a data structure
 - You can trivially replace the use of one implementation with another as long as they implement the same interface
 - Teams can work on different pieces of a large system knowing that everything will work together in the end
- **Java Interfaces are groups of related methods with empty bodies**
- Defines the syntax of what is available

```
interface Counter {  
    int banana();  
    void orange();  
    void grape();  
}
```

Whats wrong with this interface?

Algebraic Specification of Abstract Data Types

```
# Counter ADT Specification.
```

```
adt Counter
```

```
uses Integer
```

```
defines Counter
```

```
operations
```

```
new: ---> Counter
```

```
up: Counter ---> Counter
```

```
down: Counter ---> Counter
```

```
value: Counter ---> Integer
```

```
# name of specification
```

```
# specification(s) this one needs (imports)
```

```
# type(s) defined by this specification
```

```
# operations(s) defined in this specification
```

```
# constructor, "convert" TO Counter
```

```
# mutators, make Counters from Counters
```

```
# (in Java they'd change their receiver!)
```

```
# observer, "convert" FROM Counter
```

```
axioms
```

```
value(new()) >= 0
```

```
value(up(c)) > value(c)
```

```
value(down(c)) <= value(c)
```

```
# axioms for the operations defined above
```

```
# value of a new Counter is >= 0
```

```
# value of up'd Counter is > value before
```

```
# value of down'd Counter is <= value before
```

Axioms should be read as universally quantified. For example, the second axiom is "for all counters c, the value of up(c) is > the value of c" if read out aloud. The "rule of thumb" for finding axioms is to combine all constructors/mutators with all observers and then stare at that until we figure it out. :-)

Axioms are enforced by asserts and other test cases!

Variables and Types (I)

```
adt Counter
  uses Integer
  defines Counter

  operations
    new: ---> Counter
    up: Counter ---> Counter
    down: Counter ---> Counter
    value: Counter ---> Integer

  axioms
    value(new()) >= 0
    value(up(c)) > value(c)
    value(down(c)) <= value(c)

interface Counter {
  int value();
  void up();
  void down();
}
```

*Does this specification allow for floating point numbers?
How would you fix it?*

Variables and Types (2)

```
adt Counter
  uses Float
  defines Counter

  operations
    new: ---> Counter
    up: Counter ---> Counter
    down: Counter ---> Counter
    value: Counter ---> Float

  axioms
    value(new()) >= 0.0
    value(up(c)) > value(c)
    value(down(c)) <= value(c)
```

```
interface Counter {
  float value();
  void up();
  void down();
}
```

**What if you want to allow the counter to use either floats or ints?
How would you code it?**

Variable Types

```
adt Variable
  uses Any
  defines Variable<T: Any>
    operations
      new: T ---> Variable<T>
      get: Variable<T> ---> T
      set: Variable<T> x T ---> Variable<T>
    axioms
      get(new(t)) = t
      get(set(v, t)) = t
```

“Any” defines a type with = operation
T stands for “Any” type: int, float, String, ...
v and t are values of type T

```
adt Counter
  uses Any
  defines Counter<T: Any>
    operations
      new: T ---> Counter<T>
      up: Counter<T> ---> Counter<T>
      down: Counter<T> ---> Counter<T>
      value: Counter<T> ---> T
    axioms
      value(new(t)) = t
      value(up(c)) > value(c)
      value(down(c)) <= value(c)
```

Using t with new() enables more flexibility than initializing to 0, 3.14 or any other specific value

T must define >, <=, and =
new() takes a starting value t

Java Generics

In a nutshell, generics enable *types* (classes and interfaces) to be parameters when defining classes, interfaces and methods. Much like the more familiar *formal parameters* used in method declarations, type parameters provide a way for you to reuse the same code with different inputs.

- ***Stronger type checks at compile time.***

A Java compiler applies strong type checking to generic code and issues errors if the code violates type safety. Fixing compile-time errors is easier than fixing runtime errors, which can be difficult to find.

- ***Elimination of casts.***

The following code snippet without generics requires casting:

```
List list = new ArrayList();
list.add("hello");
String s = (String) list.get(0);
```

When re-written to use generics, the code does not require casting:

```
List<String> list = new ArrayList<String>();
list.add("hello");
String s = list.get(0); // no cast
```

- ***Enabling programmers to implement generic algorithms.***

By using generics, programmers can implement generic algorithms that work on collections of different types, can be customized, and are type safe and easier to read.

Implementing Variable Types with Generics

Variable.java

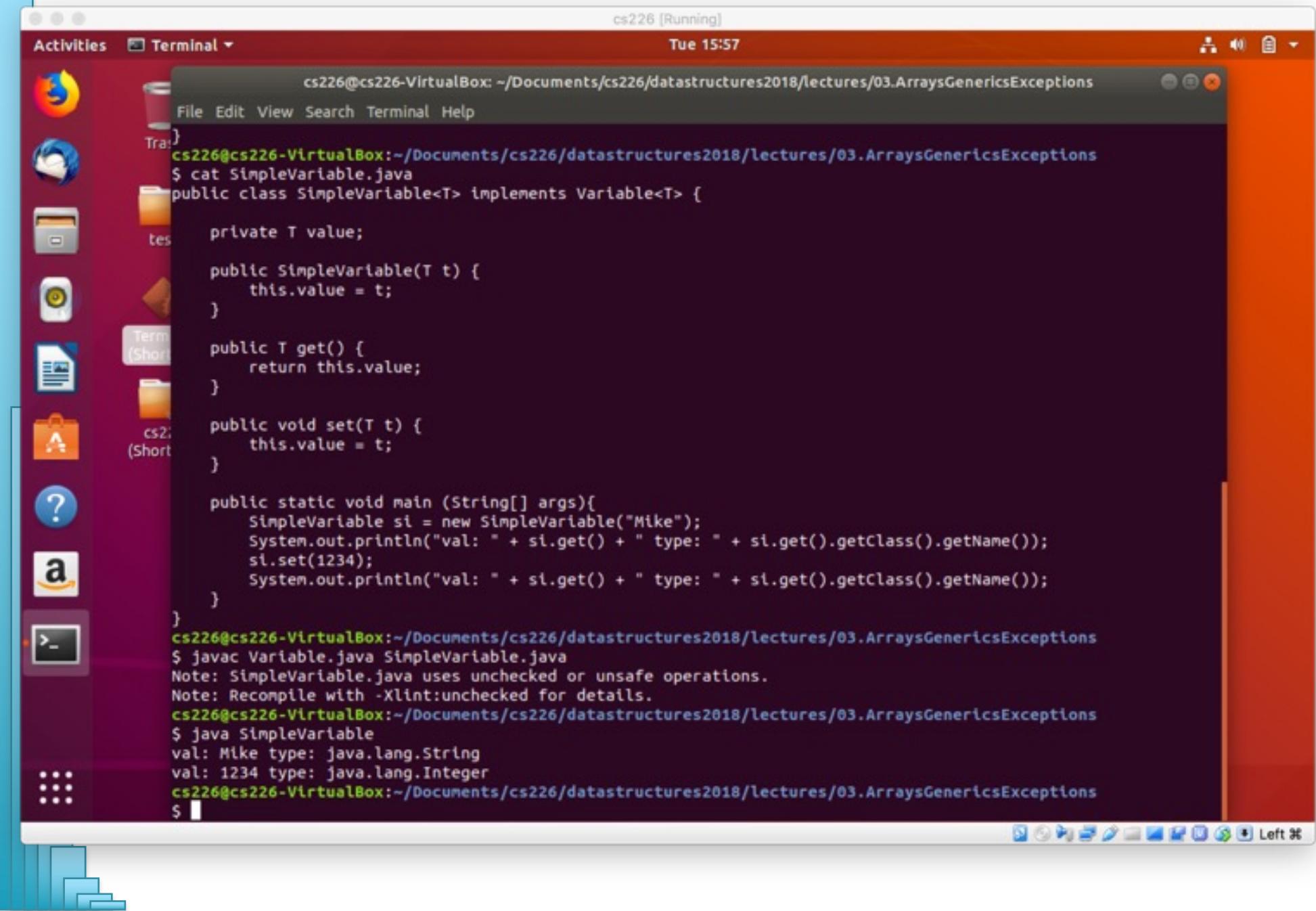
```
public interface Variable <T> {  
    public T get();  
    public void set(T t);  
}
```

SimpleVariable.java

```
public class SimpleVariable<T> implements Variable<T> {  
    private T value;  
    public SimpleVariable(T t) {  
        this.value = t;  
    }  
    public T get() {  
        return this.value;  
    }  
    public void set(T t) {  
        this.value = t;  
    }  
    public static void main (String[] args){  
        SimpleVariable si = new SimpleVariable("Mike");  
        System.out.println("val: " + si.get() +  
                           " type: " + si.get().getClass().getName());  
        si.set(1234);  
        System.out.println("val: " + si.get() +  
                           " type: " + si.get().getClass().getName());  
    }  
}
```

At compile time it will
automagically define
set(String) and set(Integer)

Implementing Variable Types with Generics

A screenshot of an Ubuntu desktop environment. On the left is a dock with various icons: Firefox, Dash, Activities, Terminal, LibreOffice, LibreOffice Calc, LibreOffice Impress, LibreOffice Writer, LibreOffice Draw, LibreOffice Base, LibreOffice Math, LibreOffice Calc (Short), LibreOffice Impress (Short), LibreOffice Writer (Short), LibreOffice Draw (Short), LibreOffice Base (Short), LibreOffice Math (Short), Help, Amazon, and a terminal icon. A red vertical bar is positioned on the right side of the screen. In the center, a terminal window titled "Activities Terminal" is open, showing the following Java code and its execution:

```
cs226@cs226-VirtualBox:~/Documents/cs226/datastructures2018/lectures/03.ArraysGenericsExceptions
$ cat SimpleVariable.java
public class SimpleVariable<T> implements Variable<T> {

    private T value;

    public SimpleVariable(T t) {
        this.value = t;
    }

    public T get() {
        return this.value;
    }

    public void set(T t) {
        this.value = t;
    }

    public static void main (String[] args){
        SimpleVariable si = new SimpleVariable("Mike");
        System.out.println("val: " + si.get() + " type: " + si.get().getClass().getName());
        si.set(1234);
        System.out.println("val: " + si.get() + " type: " + si.get().getClass().getName());
    }
}
cs226@cs226-VirtualBox:~/Documents/cs226/datastructures2018/lectures/03.ArraysGenericsExceptions
$ javac Variable.java SimpleVariable.java
Note: SimpleVariable.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
cs226@cs226-VirtualBox:~/Documents/cs226/datastructures2018/lectures/03.ArraysGenericsExceptions
$ java SimpleVariable
val: Mike type: java.lang.String
val: 1234 type: java.lang.Integer
cs226@cs226-VirtualBox:~/Documents/cs226/datastructures2018/lectures/03.ArraysGenericsExceptions
$
```

The terminal window has a dark background with light-colored text. The status bar at the bottom shows "Left 96".

Implementing Variable Types with Generics

SimpleVariable.java

```
public static void main (String[ ] args){
    SimpleVariable si = new SimpleVariable("Mike");
    System.out.println("val: " + si.get() +
                       " type: " + si.get().getClass().getName());

    SimpleVariable<String> ssi = si;
    String vals = ssi.get();
    System.out.println(vals);

    si.set(1234);
    System.out.println("val: " + si.get() +
                       " type: " + si.get().getClass().getName());

    SimpleVariable<Integer> isi = si;
    System.out.println(isi.get() + 10);
}
```

Defining the variable type
as `SimpleVariable<String>`
allows us to skip the cast!

Interfaces Review

- **Interfaces establish the “contract” between the implementation and any potential client code**
 - Helps to abstract our the key features of a data structure
 - You can trivially replace the use of one implementation with another as long as they implement the same interface
 - Teams can work on different pieces of a large system knowing that everything will work together in the end
- **Java Interfaces are groups of related methods with empty bodies**
- Defines the syntax of what is available

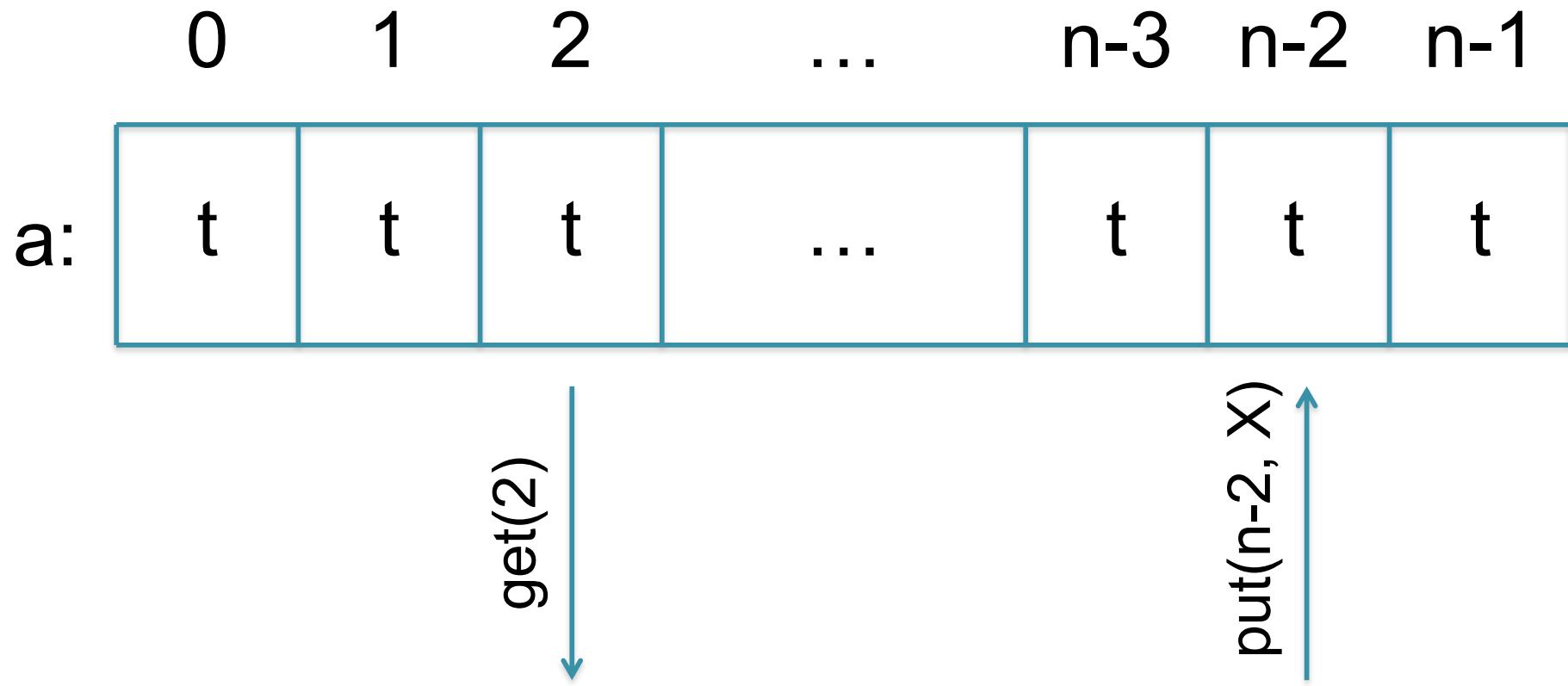
```
interface Counter {  
    int value();  
    void up();  
    void down();  
}
```

- Use an algebraic specification to define the semantics
- Use genetics to allow flexibility across types

Agenda

- 1. Quick Review**
- 2. *Introduction to Java Interfaces***
- 3. *Introduction to Generics, Exceptions and Arrays***

Our first data structure: Arrays



- Fixed length data structure
- Constant time `get()` and `put()` methods
- Definitely needs to be generic ☺

Array ADT

adt Array

uses Any, Integer

defines Array<T: Any>

Uses two related ADTs

operations

new: Integer x T ---> Array<T>

get: Array<T> x Integer ---> T

put: Array<T> x Integer x T ---> Array<T>

length: Array<T> ---> Integer

Defines method signatures

axioms

get(new(n, t), i) = t

Enforced by asserts

get(put(a, i, t), j) = (if i = j then t else get(a, j))

length(new(n, t)) = n

length(put(a, i, t)) = length(a)

preconditions

new(n, t): 0 < n

Enforced by exceptions

get(a, i): 0 <= i < length(a)

put(a, i, t): 0 <= i < length(a)



Array Interface

```
/**  
 * Arrays with integer positions.  
  
 * The constructor should take a length > 0 as well as a default  
 * value to "plaster" all over the new array. The constructor should  
 * throw LengthException if length ≤ 0.  
  
     Array(int length, T default) throws LengthException  
  
     @param <T> Element type.  
 */  
  
public interface Array<T> {  
    /**  
     * Change value at index.  
  
     * @param i Index to write value at.  
     * @param t Value to write at index.  
     * @throws IndexException if i < 0 or i > length-1.  
     */  
    void put(int i, T t) throws IndexException;  
  
    ...  
}
```

Array Interface

```
...  
  
    /**  
     * Value at index.  
  
     * @param i Index to read value at.  
     * @return Value read at index.  
     * @throws IndexException if i < 0 or i > length-1.  
     */  
    T get(int i) throws IndexException;  
  
    /**  
     * Length of array.  
  
     * @return Length of array, always > 0.  
     */  
    int length();  
}
```

Array Exceptions

IndexException.java

```
/**  
 * Exception for invalid index.  
  
 * Data structures using (integer) indices throw IndexException  
 * if a given index is out of range.  
 */  
public class IndexException extends RuntimeException {  
    private static final long serialVersionUID = 0L;  
}
```

LengthException.java

```
/**  
 * Exception for invalid length.  
  
 * Data structures that have a fixed (integer) length throw  
 * LengthException if a given length is out of range.  
 */  
public class LengthException extends RuntimeException {  
    private static final long serialVersionUID = 0L;  
}
```

The type is the main item of interest, but other information could be returned

Simple Array I

SimpleArray.java

```
/**  
 * Array implementation on top of basic Java array.
```

The obvious implementation of the `Array` interface, absolutely positively nothing fancy going on here.

There are two reasons for this class to exist: First it's an example for the style of code we're about to write a lot of. Second it's useful because Java's generics don't really play well with Java's basic arrays; we'll use `SimpleArray` in lots of places where Java's arrays would give us generic grief.

```
@param <T> Element type.  
*/  
public class SimpleArray<T> implements Array<T> {  
    // The underlying data structure of our abstract Array.  
    private T[ ] data;  
  
    /**  
     * Constructs a new SimpleArray.  
  
     * @param n Length of array, must be n > 0.  
     * @param t Default value to store in each slot.  
     * @throws LengthException if n <= 0.  
    */  
public SimpleArray(int n, T t) throws LengthException {  
    ...
```

Why bother?

Simple Array 2

...

```
public SimpleArray(int n, T t) throws LengthException {
    if (n <= 0) {
        throw new LengthException();
    }

    // This cast works around Java's problems with generic arrays.
    // The resulting warning is acceptable because there simply is
    // no better way of doing this.
    this.data = (T[]) new Object[n];

    // Array slots are null by default.
    if (t == null) {
        return;
    }

    for (int i = 0; i < n; i++) {
        this.data[i] = t;
    }
}
```

Workaround for java syntax

Simple Array 3

```
...
// If we let ArrayIndexOutOfBoundsException propagate, we leak an
// implementation detail we should probably hide. (Also that name
// is so horrible, it deserves to live in a dark cave in Mordor.)

@Override
public T get(int i) throws IndexException {
    try {
        return this.data[i];
    } catch (ArrayIndexOutOfBoundsException e) {
        throw new IndexException();
    }
}

@Override
public void put(int i, T t) throws IndexException {
    try {
        this.data[i] = t;
    } catch (ArrayIndexOutOfBoundsException e) {
        throw new IndexException();
    }
}

@Override
public int length() {
    return this.data.length;
}
...
```

These let us “hide” the exceptions from our underlying datatypes

Simple Array 4

```
...
public static void main(String [] args) throws
IndexException, LengthException {
    Array<String> a = new SimpleArray<String>(4, "226");
    assert a.length() == 4;
    for (int i =0; i <a.length(); i++){
        assert a.get(i).equals("226");
    }
    a.put(2, "Peter");
    assert a.length() == 4;
    assert a.get(2).equals ("Peter");
    assert a.get(0).equals ("226");
    assert a.get(1).equals ("226");
    assert a.get(3).equals ("226");

    System.out.println("Passed the value assertions");
...
}
```

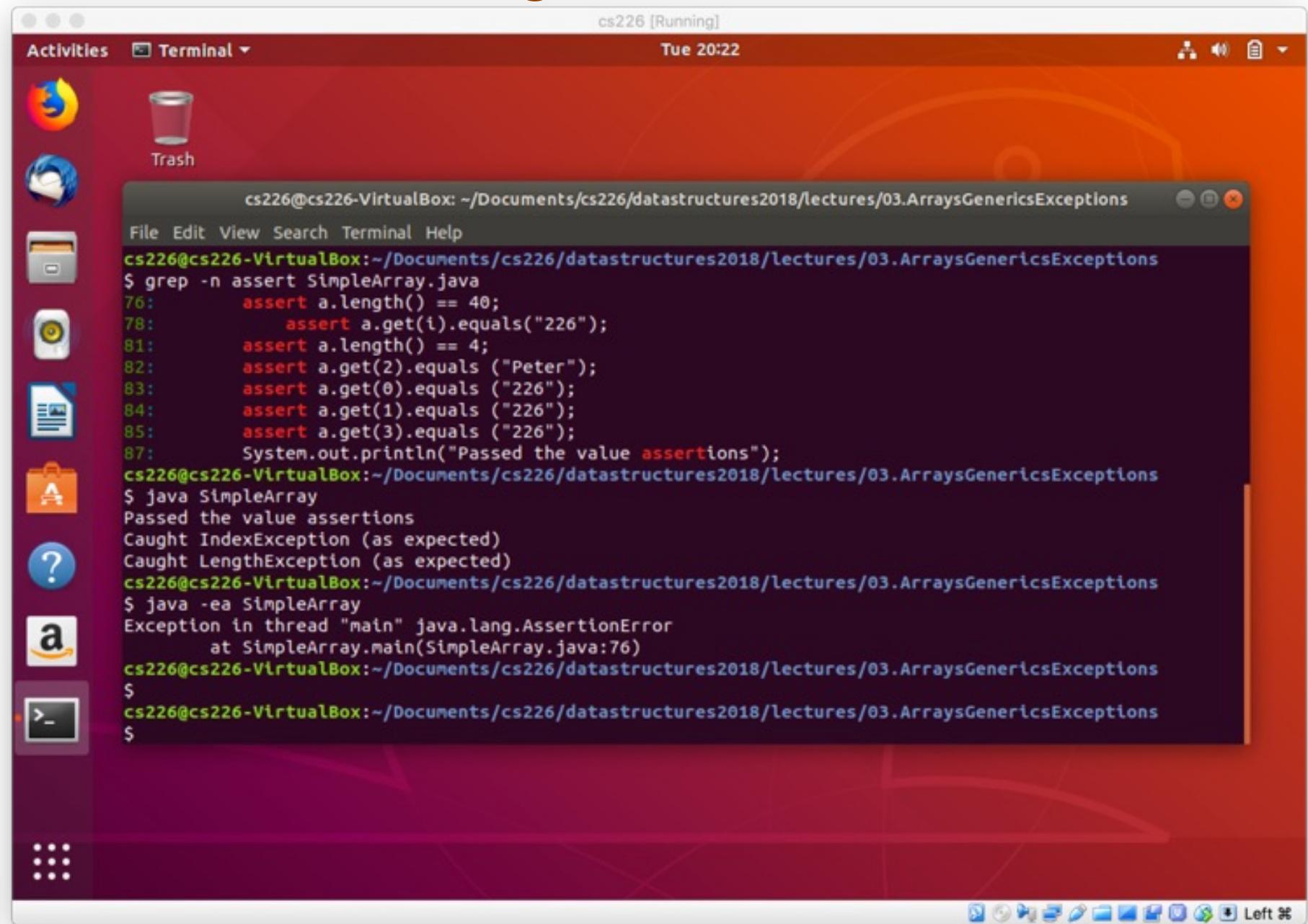
Simple Array 5

...

```
try {
    a.put(a.length(), "Paul");
    System.out.println("Didnt get the exception");
}
catch (IndexException e)
{
    System.out.println("Caught IndexException (as expected)");
}

try {
    Array<String> b = new SimpleArray<String>(0, "Mike");
    System.out.println("No exception after creating second array");
}
catch (LengthException e)
{
    System.out.println("Caught LengthException (as expected)");
}
}
```

Running with Assertions



A screenshot of an Ubuntu desktop environment. The terminal window title is "cs226 [Running]" and the date and time are "Tue 20:22". The terminal content shows the execution of a Java program named SimpleArray. It first runs "grep -n assert SimpleArray.java" to show the code containing several assertions. Then it runs "java SimpleArray", which prints "Passed the value assertions" and catches two expected exceptions: IndexException and LengthException. Finally, it runs "java -ea SimpleArray", which results in an uncaught AssertionException due to one of the assertions failing.

```
cs226@cs226-VirtualBox: ~/Documents/cs226/datastructures2018/lectures/03.ArraysGenericsExceptions
$ grep -n assert SimpleArray.java
76:         assert a.length() == 40;
78:             assert a.get(i).equals("226");
81:             assert a.length() == 4;
82:             assert a.get(2).equals ("Peter");
83:             assert a.get(0).equals ("226");
84:             assert a.get(1).equals ("226");
85:             assert a.get(3).equals ("226");
87:             System.out.println("Passed the value assertions");
cs226@cs226-VirtualBox:~/Documents/cs226/datastructures2018/lectures/03.ArraysGenericsExceptions
$ java SimpleArray
Passed the value assertions
Caught IndexException (as expected)
Caught LengthException (as expected)
cs226@cs226-VirtualBox:~/Documents/cs226/datastructures2018/lectures/03.ArraysGenericsExceptions
$ java -ea SimpleArray
Exception in thread "main" java.lang.AssertionError
        at SimpleArray.main(SimpleArray.java:76)
cs226@cs226-VirtualBox:~/Documents/cs226/datastructures2018/lectures/03.ArraysGenericsExceptions
$
```

\$ java -ea SimpleArray

Next Steps

1. Reflect on the magic and power of interfaces, generics, and exceptions 😊
2. Check on Piazza
3. Download class virtual machine, get CS account and/or set up Linux!
4. Get comfortable with a editor (VI rules!) and/or an IDE (Eclipse for Java)
5. Get comfortable with checkstyle



Welcome to CS 600.226

<https://github.com/schatzlab/datastructures2018>

Questions?