On your way in...

Pick-up
1. Lecture 33 Notes

Drop-off
1. HW11 (2 piles, IDs < 50, etc.)
FINISHED LAB 10?

If you finished your Java Lab 10, then you don’t need to come to lab.

We’re done all assignments!
Finishing Up Exam-able Content

• After today’s class, we’ve finished exam-worthy material
• Wednesday → Computing Topics
  • Snacks
• Friday
  • Semester review → Bring your questions!
    • Concept you’re not sure about?
    • HW question you got incorrect and don’t know why?
    • Been replicating what we do in class, but don’t know why we’re doing that?
  • Evaluations
Welcome to CS 134!

Introduction to Computer Science
Iris Howley

- JAVA V -

Spring 2019
JAVÁ

Building our own List data structure.
List mylist = new List(); Initially, we reserve 12 spaces for our List object

Initially, size is set to 0

We write our code to only look at elements after size, so it doesn’t [currently] matter what’s stored at mylist[size] or after
List mylist = new List();  Initially, we reserve 12 spaces for our List object

Initially, size is set to 0

mylist.append(33); When we add an item to the list, we move size

We’re still writing our List class so that it doesn’t look at any of the elements stored after size.
To remove the list item from the end of the list, we just move `size` back one.

There may be items after `size`, but we write our List class to treat that as if it isn’t there.
To remove the list item, from a given index, we need to move all items after that index to the left by one.

There may be items after size, but we write our List class to treat that as if it isn’t there.
Allocation versus Size

mylist →

mylist.append(5);

When size is the same as allocation, we have no more space to append into! So we double the allocation in a new array, and copy our values over to it. That’s the new array.
JAVA

Generic Types and Object Supertype.
public class List<T> {

    private T[] data; // where the data is
    private int allocation; // how many s
    private int size; // how many i

    public List() // construct an empty list
    {
        allocation = 12; // list can go
        size = 0; // logical si
        data = (T[])new Object[allocation]; //
    }

    public int size() // return the number of elements logically
    {
        return size;
    }

    public T get(int i) // get the i-th element from the List.
    {
        assert (0 <= i) && (i < size);
        return data[i];
    }
}
public class List<T> {
    private T[] data;  // where the data is stored
    private int allocation; // how many slots
    private int size;  // how many items

    public List()  // construct an empty list
    {
        allocation = 12; // list can grow
        size = 0;  // logical size
        data = (T[])new Object[allocation];  // allocate memory
    }

    public int size()  // return the number of elements logically
    {
        return size;
    }

    public T get(int i)  // get the i-th element from the List.
    {
        assert (0 <= i) && (i < size);
        return data[i];
    }
}
Generics

```java
public static void main(String[] args)
{
    // put all the runtime arguments into a list
    List<String> l = new List<String>();
    for (int i = 0; i < args.length; i++) {
        l.append(args[i]);
    }
    System.out.print("Here's the list: ");
    System.out.println(l);
}
```
Generics

• We use type \( T \) to represent a generic type
• With a list, this ensures that all elements in that list are of type \( T \)
• The generic type, \( T \), is specified when the list is instantiated
  ▪ Imagine filling in every occurrence of “\( T \)” in the class with the type listed between angled brackets \(<\text{ThisType}>\)
• \texttt{List<String> l = new List<String>();}
  ▪ We’re specifying that every element in \( l \) is a String variable!
  ▪ \( l.get(0) \) will return a String, not an object
Generic vs. Object

• private Object[] dataO vs. private T[] dataT;

• When we iterate over dataO, we don’t know what type of object it stores:
  ▪ Object[] dataO = {new Color(0,0,0), new Color(1,1,1)};
  ▪ dataO[1].rgb()
    o But Objects don’t have an .rgb() method!
  ▪ We have to cast it:
    o (Color) dataO[1].rgb();
    o This is dangerous and may not give useful errors

• With generics (T[] dataT), we know what type of variable is being stored in there
  ▪ Typing errors will be caught at compile time
QUESTIONS?
Programming with Assertions

• See documentation:
  ▪ [https://docs.oracle.com/javase/7/docs/technotes_guides/language/assert.html](https://docs.oracle.com/javase/7/docs/technotes_guides/language/assert.html)
Java – Things to Notice

- `javap java.lang.Math`
  - Instead of pydoc3

- We use the Javadoc API
  - https://docs.oracle.com/javase/8/docs/api/index.html

- Like python, can make comments that generate javadoc for your code
Java: Language Basics

• Java Docs Tutorial:
  • https://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html

• Java Docs API (like pydoc3)
  • https://docs.oracle.com/javase/8/docs/api/index.html?overview-summary.html
Java - Arrays

- \( \text{int}[] \text{ data} = \text{new int[100];} \)
- \text{System.out.println(data.length + " vs 100.";)
- \text{System.out.println(data[50]);

- 100 vs 100.
- 0

- Maybe we want something more like a Python list, where we create a new list and it’s empty!
- Still have to allocate memory for the list
Java - Arrays

This is the memory ALLOCATION

- `int[] data = new int[100];`
- `System.out.println(data.length + " vs 100.");`
- `System.out.println(data[50]);`

- `100 vs 100.`
- `0`

We also want a list SIZE for how many elements are actually in the list

We don’t want to be able to access an item at index 50, if we haven’t added an item there!
List.java

- cd ~/cs134/shared/examples/05.06
- git pull
- emacs List.java
JAVA

Subtypes and Supertypes.
Subtypes

Object

Automobile

Mazda

Tesla

public void drive()

public void charge()
Subtypes

Object

... automobile

Automobile

Mazda

Tesla

public void drive()

public void charge()

Object is the top-most supertype of all objects.

Tesla is a subtype of Automobile. Automobile is Mazda’s supertype.
Subtypes

Object

Automobile

Mazda

Tesla

public void drive()

public void charge()

Mazda mine = new Mazda()

mine doesn’t have a method .charge()!
Subtypes

Object
...
Automobile

Mazda
Tesla

public void drive()

public void charge()

Automobile a = new Tesla();

It’s always legal to assign a subtype value to a supertype variable. (the reverse is dangerous)