Introduction to Computer Science

PTI CS 103 Lecture Notes

The Prison Teaching Initiative

This version: April 17, 2019

Acknowledgements

TK.

Contents

1

Introduction

1.1 Overview of course

Knowing just a little bit of computer science can get you started right away in actual applications. One of the goals of this course is to learn about the fascinating subject of computer science. Another is to develop algorithmic thinking skills that will help with day-to-day critical problem-solving skills. But perhaps the most important goal of the course is to develop coding skills, which will not only open up new job opportunities but also make you more effective in most areas of business.

In the first semester, we will spend the first two classes of each week on computer science theory and special topics. The final day of each week will be a lab day, where we actually start practicing coding skills.

In the second semester, we will start focusing more on practical coding, with a single day a week for theory and 2 lab periods per week for coding.

Broadly, we will cover the following topics:

- How modern computers work
  - Hardware
  - Software
  - Computer networks and information systems
- Algorithms for quickly solving complex problems
  - Searching
  - Sorting
- Data structures
  - Arrays
  - ArrayLists
- Applications of Computer science
  - Basic coding in Java
  - How to use productivity software

1.2 Brief history of computer science

Timeline (credit: https://www.worldsciencefestival.com/infographics/a_history_of_computer_science/):

- Invention of the abacus (2700-2300 BC, Sumerians)
- Design of first modern-style computer (Charles Babbage, 1837)

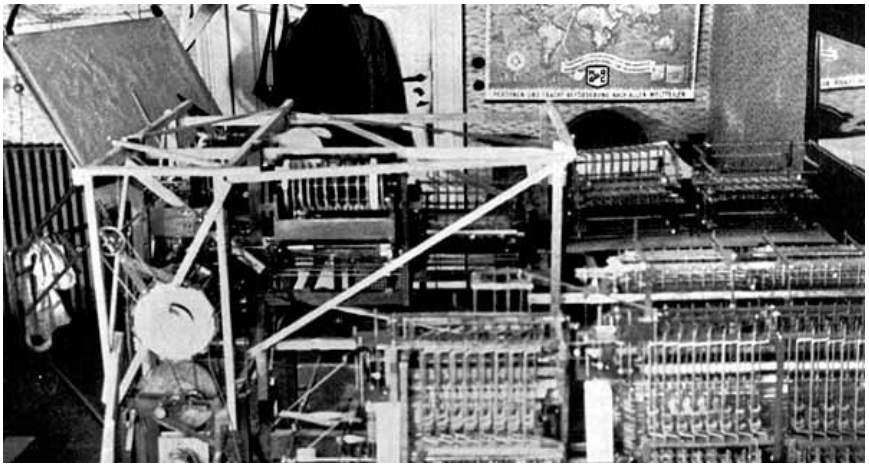

Figure 1.1: Construction of Konrad Zuse's Z1, the first modern computer, in his parents' apartment. Credit: https://history-computer.com/ModernComputer/Relays/Zuse.html

- Design of first computer algorithm (Ada Lovelace, 1843)
- Invention of first electronic digital computer (Konrad Zuse, 1941)
- Invention of the transistor (Bell labs, 1947)
- Invention of the first computer network (early Internet) (DARPA, 1968)
- Invention of the World Wide Web (Sir Tim Berners-Lee, 1990)

1.3 Components of a computer

A computer is an electronic device used to process data. Its basic role is to convert data into information that is useful to people.

There are 4 primary components of a computer:

- Hardware
- Software
- Data
- User

1.3.1 Hardware

Computer hardware consists of physical, electronic devices. These are the parts you actually can see and touch. Some examples include

- Central processing unit (CPU)
- Monitor
- CD drive
- Keyboard
- Computer data storage
- Graphic card
- Sound card
- Speakers
- Motherboard

We will discuss these components in more detail in lecture 3.

Figure 1.2: Examples of hardware components of a personal computer. Credit: https://www5.cob.ilstu.edu/dsmath1/tag/computer-hardware/

1.3.2 Software

Software (otherwise known as "programs" or "applications") are organized sets of instructions for controlling the computer.

There are two main classes of software:

- Applications software: programs allowing the human to interact directly with the computer
- Systems software: programs the computer uses to control itself

Some more familiar applications software include

- Microsoft Word: allows the user to edit text files
- Internet Explorer: connects the user to the world wide web
- iTunes: organizes and plays music files

While applications software allows the user to interact with the computer, systems software keeps the computer running. The operating system (OS) is the most common example of systems software, and it schedules tasks and manages storage of data.

We will dive deeper into the details of both applications and systems software in lecture 4.

1.3.3 Data

Data is fundamentally information of any kind. One key benefit of computers is their ability to reliably store massive quantities of data for a long time. Another is the speed with which they can do calculations on data once they receive instructions from a human user.

While humans can understand data with a wide variety of perceptions (taste, smell, hearing, touch, sight), computers read and write everything internally as "bits", or 0s and 1s.

Computers have software and hardware which allow them to convert their internal 0s and 1s into text, numerals, and images displayed on the monitor; and sounds which can be played through the speaker.

Similarly, humans have hardware and software used for converting human signals into computer-readable signals: a microphone converts sound, a camera converts pictures, and a text editor converts character symbols.

1.3.4 Users

Of course, there would be no data and no meaningful calculations without the human user. Computers are ultimately tools for making humans more powerful.

As we will see in the next section, however, different types of computers have different roles for the user.

1.4 Types of computers

1.4.1 Supercomputers

These are the most powerful computers out there. The are used for problems that take along time to calculate. They are rare because of their size and expense, and therefore primarily used by big organizations like universities, corporations, and the government.

The user of a supercomputer typically gives the computer a list of instructions, and allows the supercomputer to run on its own over the course of hours or days to complete its task.

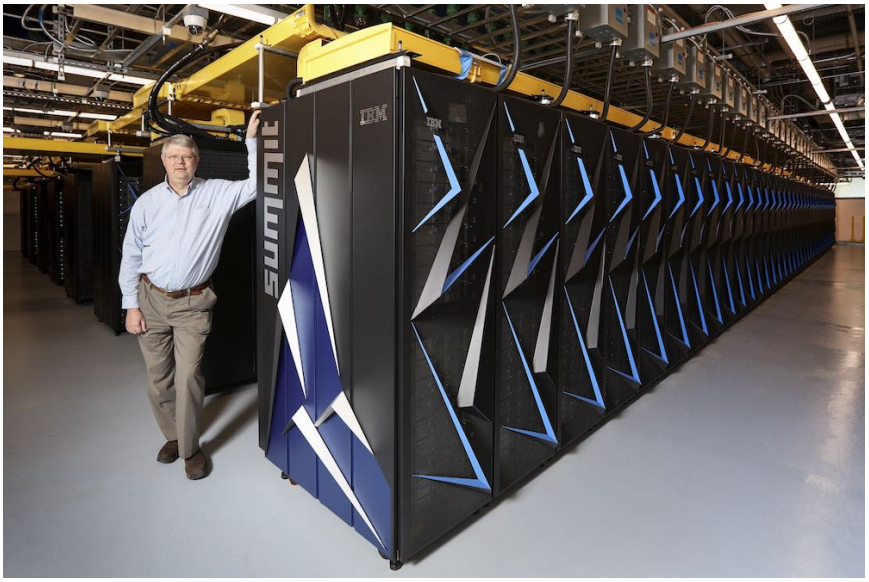

Figure 1.3: Summit, a world-class supercomputing cluster at Oak Ridge National Laboratory in Tennessee. Credit: https://insidehpc.com/2018/11/new-top500-list-lead-doe-supercomputers/

1.4.2 Mainframe computers

Although not as powerful as supercomputers, mainframe computers can handle more data and run much faster than a typical personal computer. Often, they are given instructions only periodically by computer programmers, and then run on their own for months at a time to store and process incoming data. For example, census number-crunching, consumer statistics, and transactions processing all use mainframe computers

1.4.3 Personal computers

These are the familiar computers we use to interact with applications every day. Full-size desktop computers and laptop computers are examples

1.4.4 Embedded computers

In the modern "digital" age, nearly all devices we use have computers embedded within them. From cars to washing machines to watches to heating systems, most everyday appliances have a computer within them that allows them to function.

1.4.5 Mobile computers

In the past 2 decades, mobile devices have exploded onto the scene, and smartphones have essentially become as capable as standalone personal computers for many tasks.

1.5 Why computers are useful

Computers help us in most tasks in the modern age. We can use them, for example, to

- write a letter
- do our taxes
- play video games
- watch videos
- surf the internet

- keep in touch with friends
- date
- order food
- control robots and self-driving cars

Example 1.5.1: What are some other tasks a computer can accomplish?

This is why the job market for computer scientists continues to expand, and why computer skills are more and more necessary even in non-computational jobs.

According to a Stackoverflow survey from 2018 (https://insights.stackoverflow.com/survey/2018/), 9% of professional coders on the online developer community have only been coding for 0-2 years. This demonstrates two things:

1. The job market for people with coding skills is continually expanding
2. It doesn't take much to become a coder

Some examples of careers in computer science include

- IT management / consulting
- Game developer
- Web developer
- UI/UX designer
- Data analyst
- Database manager

References

Computer Science: An Interdisciplinary Approach, Robert Sedgewick and Kevin Wayne.

University of Wisconsin-Madison CS 202 Lectures, Andrea Arpaci-Dusseau. (http://pages.cs.wisc.edu/ dussea F11/)

2

Hardware

Modern computers are a complex group of devices working together to perform a common task (or tasks). A user will interact with a computer through a variety of input and output devices (e.g. keyboards, mice, speakers, microphone, and monitors). A user's input will be processed, some computations will be performed, and then the resulting output will be displayed to the user. When most of us thinking about computers, we often think of a desktop or laptop computer, that come equipped with a keyboard, mouse, and monitor; however, many things we interact with daily are computerized, including cell phones, cars, traffic lights, smart watches, televisions, and manufacturing lines. Today each of these items have sensors to perceive the real world, use an embedded computing device to understand the sensory input, and use a combination of display and mechanical devices to interact with the real world.

Example 2.0.1: For intersections across busy roadways, some traffic lights are computerized to optimize road traffic. These lights will stay green along the busier of the two roads, and use cameras or pressure sensors to detect the presence of cars along the less busy of the two roads, thus switching to allowing the cars on the less busy road to cross when it arrives. Overall, providing a less congested intersection by relying on an embedded computer.

Today we will introduce three fundamental parts of computer systems: input and output devices, memory, and the central processing unit (CPU). These components work together to perform the basic building blocks of input processing, storage, control, and output. Understanding how the three parts work together will allow us to create powerful information processing tools. We will introduce each of these parts in turn. In figure 2.1, we see how these parts come together to form a computer system (similar to the ones you'll use to program in this course).

2.1 Input and Output Devices

We will begin by discussing input and output devices. These devices allow the computer to interact with users and the world directly. Without these devices, a computer system would be very boring, always performing the same computation each time it's used. Even if it did compute a different value we would be unable to examine the value. A computer needs to be able to accept input and allow output. The first computers would occupy a large room in an office building and connect to a terminal (a keyboard and a text screen) in another room for users to interact with. Thanks to Moore's Law, computers many orders of magnituted more powerful can fit in the palm of your hand. Likewise, the variety of input and output devices has multipled. We still have the keyboard and monitor, we've added the mouse for interacting with graphical displays. Today's phones are more computer than phone, coming equipped with: speakers, microphones, touch screens, cameras, fingerprint scanners, radio transmitters, and much more. Computers

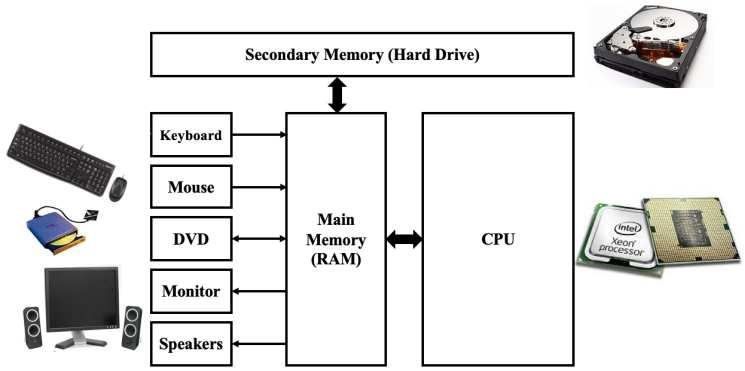

Figure 2.1: Interconnected parts of a computer system (keyboard, mouse, monitor, DVD player / burner, speaker, hard drive, CPU).

even come embedded in other devices like cars, traffic lights, X-ray machines, and thermostats to both control and monitor the devices. As shown in Figure 2.1, these devices connect to the rest of the computer through the computer's memory. This kind of input and output is called memory mapped I/O (input and output). Creaters of input (or output) devices are assigned a section of the computer's memory to write (read resp.) data. The computer will then read (write resp.) data to those locations to communicate with the given device. The creaters of these devices, agree upon a known format to read and write data.

Example 2.1.1: You can think of this communication between devices and computer similar to leaving messages for a friend in a locker. Only you and your friend have access to this locker, which only holds space for one message. The format you agree upon is which language you'll use to speak (e.g. English) and any special keywords or phrases. You might agree with your friend, that if either of you write a message saying "The morning is upon us" that the other will wait until "The night has come" before leaving any new messages.

The format that devices and computers communicate in are generally very simple and structured to permit fast and easily understandable communication for computers and devices.

Example 2.1.2: A monitor is a graphical display for computers. Let's consider a monitor connected to a computer that only displays in black and white images that are 20 x 20 pixels large. The monitor and keyboard agree upon using the following format to communicate. The format is black and white images that are 20 x 20 pixels large. Each pixel's value is represented at 0 for black and 1 for white. Then an image is represented as a 400 = (20 x 20) long sequence of pixel values. The sequence is ordered left to right, top to bottom. Now that both the monitor and computer agree upon the communication format, the computer can write images to the section of memory dedicated to the monitor and the monitor will read the image and display the image on it's screen.

Note: while this is a simplified example, this is similar to how modern graphical displays communicate with computers.

2.2 Memory

Another fundamental part of a computer, is the memory. There are two major types of memory, Main Memory (RAM), and Secondary Memory (e.g. hard disks, solid-state

drves, tape drives, etc.) Main memory is volatile, meaning that the contents of the memory is not preserved when a computer is turned off and back on. On the other hand Secondary Memory, is meant to be persistent (the opposite of volatile). Main Memory can be thought of as the "scratch paper" the computer uses for computations. Computers will also use Main Memory as a conduit for communicating between the CPU and all other parts of the computer. In most modern computers, programs are treated as data. That is the individual instructions that combine to form a program are stored in memory just as data is. It is the job of the computer to properly understand if a segment of memory is data or a program. The computer is able to fetch data from Secondary Memory to Main Memory or persist data in Main Memory to Secondary Memory when needed.

2.3 Central Processing Unit

The final part of a computer we will introduce today is the central processing unit (CPU) — processor, main processor, etc. The CPU is the physical circuitry of a computer that performs instructions. The CPU is responsible for fetching, decoding, and executing all instructions. These instructions form the basic bulding blocks of all programs. These instructions vary between different brands of CPUs but in general they will include arithmetric, control, input, and output functionalities. Example 2.3.1 shows several instructions that together would perform `x = x + y`, given `x` is stored in memory location 16 and `y` at memory location 20. These instructions are quite low level, and harder for humans to read than the programs we will write in this course. However, the programs we write will be translated into these instructions as they are easily understood by the CPU and match directly with the operations the CPU can handle. We will not focus further on programs at this level, but rather try to understand that when we program, we will build upon these pre-existing building blocks.

Example 2.3.1:

```
load R1 16    -- Load value at memory location 16 into register 1
load R2 20    -- load value at memory location 20 into register 2
add  R1 R2 R1 -- add the value in register 1 to the value in register 2
                 and store in register 1
store R1 16   -- store the value in register 1 to memory location 16
```

In most computers, the CPU is responsible for all computing needs of the computer. In some select systems, there will be additional hardware to perform specialized operations (e.g. graphics processing units for processing / producing images). It is the CPU's responsbility to control the computer and coordinate with devices to execute programs.

2.4 Conclusion

In this chapter, we covered the three fundamental parts of a computer system: input and output devices, Main and Secondary Memory, and the CPU. We discussed their roles, relationships, and basic capabilities. I hope that this will help you better understand how hardware works at a high level to better improve how to write programs that will eventually run on these computer system. In the next chapter we will begin discussing the concept of Software and how its similarities and differences to hardware and where the boundary between the two lies.

3

Arrays

Consider this snippet of code:

```
if      (day ==  0) System.out.println("Monday");
else if (day ==  1) System.out.println("Tuesday");
else if (day ==  2) System.out.println("Wednesday");
else if (day ==  3) System.out.println("Thursday");
else if (day ==  4) System.out.println("Friday");
else if (day ==  5) System.out.println("Saturday");
else if (day ==  6) System.out.println("Sunday");
```

What does this code do? It prints the day of the week after conditioning on the value of an integer `day`. But this code is repetitive. It would be useful if we had some way of creating a list of days of the week, and then just specifying which of those days we wanted to print. Something like this:

```
System.out.println(DAYS\_OF\_WEEK[day]);
```

To achieve this in Java, we need arrays.

Definition 3.0.1. An *array* is an ordered and fixed-length list of values that are of the same type. We can access data in an array by *indexing*, which means referring to specific values in the array by number. If an array has `n` values, then we think of it as being numbered from `0` to `n-1`.

To *loop* or *iterate* over an array means that our program accesses every value in the array, typically in order. For example, if we looped over the array in the diagram, that would mean that we looked at the value at the 0th index, then the value at the 1st index, then the value at the 2nd index, and so on.

When we say that the array is "ordered" is that the relationship between an index and its stored value is unchanged (unless we explicitly modify it). If we loop over an unchanged array multiple times, we will always access the same values.

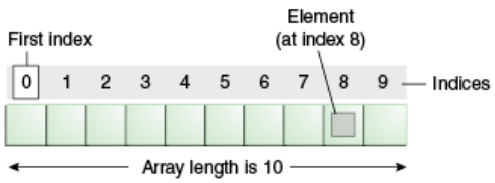

Figure 3.1: Diagram of an array (Credit: https://www.geeksforgeeks.org/arrays-in-java/)

Arrays are *fixed-length*, meaning that after we have created an array, we cannot change its length. We will see in the next chapter [TK: confirm] that `ArrayLists` are an array-like data structure that allows for changing lengths.

Finally, all the values in an array must be of the same type. For example, an array can hold all floating point numbers or all characters or all strings. But an array cannot hold values of different types.

3.1 Creating arrays

The syntax for creating an array in Java has three parts:

1. Array type
2. Array name
3. Either: array size or specific values

For example, this code creates an array of size `n = 10` and fills it with all `0.0`s

```
double[] arr;                       // Declare array
arr = new double[n];                // Initialize the array
for (int i = 0; i < n; i++) {       // Iterate over array
    arr[i] = 0.0;                   // Initialize elements to 0.0
}
```

The key steps are: we first declare and initialize the array. We then loop over the array to initialize specific values. We can also initialize the array at compile time, for example

```
String[] DAYS_OF_WEEK = {
//  Indices:
//  0      1      2      3      4      5      6
    "Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"
};
```

Notice the difference in syntax. When creating an empty array, we must specify a size. When initialize an array at compile time with specific values, the size is implicit in the number of values provided.

Finally, in Java, it is acceptable to move the brackets to directly after the type declaration to directly after the name declaration. For example, these two declarations are equivalent:

```
int arr[];
int[] arr;
```

3.2 Indexing

Consider the array `DAYS_OF_WEEK` from the previous section. We can *index* the array using the following syntax:

```
System.out.println(DAYS_OF_WEEK[3]);  // Prints "Thu"
```

In Java, array's are said to use *zero-based indexing* because the first element in the array is accessed with the number `0` rather than `1`.

Example 3.2.1: What does `System.out.println(DAYS_OF_WEEK[1]);` print?

Example 3.2.2: What does this code do? What number does it print?

```
double sum = 0.0;
double[] arr = { 1, 2, 2, 3, 4, 7, 9 }
for (int i = 0; i < arr.length; i++) {
    sum += arr[i];
}
System.out.println(sum / arr.length);
```

3.3 Array length

As mentioned previously, arrays are *fixed-length*. After you have created an array, it's length is unchangeable. You can access the length of an array `arr[]` with the code `arr.length`.

Example 3.3.1: What does `System.out.println(DAYS_OF_WEEK.length);` print?

Example 3.3.2: Write a `for` loop to print the days of the week in order (Monday through Sunday) using an array rather than seven `System.out.println` function calls.

3.4 Default initialization

In Java, the default initial values for numeric primitive types is `0` and `false` for the `boolean` type.

Example 3.4.1: Consider this code from earlier:

```
double[] arr;
arr = new double[n];
for (int i = 0; i < n; i++) {
    arr[i] = 0.0;
}
```

Rewrite this code to be a single line.

3.5 Bounds checking

Consider this snippet of code.

Example 3.5.1: Where is the bug?

```
int[] arr = new int[100];
for (int i = 0; i <= 100; ++i) {
    System.out.println(arr[i]);
```

```
}
```

The issue is that the program attempts to access the value `arr[100]`, while the last element in the array is `arr[99]`.

This kind of bug is called an "off-by-one error" and is so common it has a name. In general, an off-by-one-error is one in which a loop iterates one time too many or too few.

Example 3.5.2: Where is the off-by-one-error?

```
int[] arr = new int[100];
for (int i = 0; i < array.length; i++) {
    arr[i] = i;
}
for (int i = 100; i > 0; --i) {
    System.out.println(arr[i]);
}
```

Example 3.5.3: Fill in the missing code in this `for` loop to print the numbers in reverse order, i.e. `5, 4, 3, 2, 1`:

```
int[] arr = { 1, 2, 3, 4, 5 };
for (???) {
    System.out.println(arr[i]);
}
```

3.6 Empty arrays

This code prints five values, one per line, but we never specified which values. What do you think it prints?

```
int[] arr = new int[5];
for (int i = 0; i < arr.length; i++) {
    System.out.println(arr[i]);
}
```

In Java, an unitialized or empty array is given a default value:

- For `int`, `short`, `byte`, or `long`, the default value is `0`.
- For `float` or `double`, the default value is `0.0`
- For `boolean` values, the default value is `false`.
- For `char`, the default value is the null character `'\0000'`.

Note that an array can be partially initialized.

Example 3.6.1: What does this code print?

```
char[] alphabet = new char[26];
alphabet[0] = 'a';
alphabet[1] = 'b';
```

```
for (int i = 0; i < alphabet.length; i++) {
    System.out.println(alphabet[i]);
}
```

3.7 Enhanced for loop

So far, we have seen how to iterate over arrays by indexing each element with a number:

```
char[] vowels = {'a', 'e', 'i', 'o', 'u'};
for (int i = 0; i < vowels.length; ++ i) {
    System.out.println(vowels[i]);
}
```

We can perform the same iteration without using indices using an "enhanced `for` loop" or `for-each` loops:

```
char[] vowels = {'a', 'e', 'i', 'o', 'u'};
for (char item: vowels) {
    System.out.println(item);
}
```

3.8 Exchanging and shuffling

Two common tasks when manipulating arrays are *exchanging two values* and *shuffling* values. (*Sorting* is more complicated and will be address later.)

To exchange to values, consider the following code:

```
double[] arr = { 1.0, 2.0, 3.0, 4.0, 5.0, 6.0 };
int i = 1;
int j = 4;
double tmp = arr[i];
arr[i] = arr[j];
arr[j] = tmp;
```

Example 3.8.1: What are the six values in the array, in order?

To shuffle the array, consider the following code:

```
int n = arr.length;
for (int i = 0; i < n; i++) {
    int r = i + (int) (Math.random() * (n-i));
    String tmp = arr[r];
    arr[r] = arr[i];
    arr[i] = tmp;
}
```

Example 3.8.2: What does this code do:

```
for (int i = 0; i < n/2; i++) {
    double tmp = arr[i];
    arr[i] = arr[n-1-i];
```

```
4       arr[n-i-1] = tmp;
5   }
```

Exercises

3.1 Write a program that reverses the order of values in an array.

3.2 What is wrong with this code snippet?

```
1   int[] arr;
2   for (int i = 0; i < 10; i++) {
3       arr[i] = i;
4   }
```

3.3 Rewrite this snippet using an enhanced `for-each` loop (for now, it is okay to re-define the array):

```
1   char[] vowels = {'a', 'e', 'i', 'o', 'u'};
2   for (int i = array.length; i >= 0; i--) {
3       char letter = vowels[i];
4       System.out.println(letter);
5   }
```

3.4 Write a program that uses `for` loops to print the following pattern:

```
1   1********
2
3   12*******
4
5   123******
6
7   1234*****
8
9   12345****
10
11  123456***
12
13  1234567**
14
15  12345678*
16
17  123456789
```

3.5 Write a program `HowMany.java` that takes an arbitrary number of command line arguments and prints how many there are.

References

Computer Science: An Interdisciplinary Approach, Robert Sedgewick and Kevin Wayne.

4

ArrayLists

A *collection* is a group of objects. Today, we'll be looking at a very useful collection, the `ArrayList`. A *list* is an ordered collection, and an `ArrayList` is one type of list.

Create a class `NameTracker` and follow along in it.

Before we can use an ArrayList, we have to import it:

```
import java.util.ArrayList;
```

Next, we call the constructor; but we have to declare the type of object the :code:‘ArrayList‘ is going to hold. This is how you create a new :code:‘ArrayList‘ holding :code:‘String‘ objects.

```
ArrayList<String> names = new ArrayList<String>();
```

Notice the word “String” in angle brackets. This is the Java syntax for constructing an ArrayList of String objects.

We can add a new String to names using the :code:‘add()‘ method.

```
names.add("Ana");
```

Example 4.0.1: Exercise: Write a program that asks the user for some names and then stores them in an ArrayList. Here is an example program:

```
Please give me some names:
Sam
Alecia
Trey
Enrique
Dave
Your name(s) are saved!
```

We can see how many objects are in our ArrayList using the size() method.

```
System.out.println(names.size()); // 5
```

Example 4.0.2: Modify your program to notify the user how many words they have added.

```
Please give me some names:
Mary
Judah
Your 2 name(s) are saved!
```

Remember how the `String.charAt()` method returns the `char` at a particular index? We can do the same with names. Just call `get()`:

```
names.add("Noah");
names.add("Jeremiah");
names.add("Ezekiel");
System.out.println(names.get(2)); // ‘‘Ezekiel’’
```

Example 4.0.3: Update your program to repeat the names back to the user in reverse order. Your solution should use a for loop and the size() method. For example:

```
Please give me some names:
Ying
Jordan

Your 2 name(s) are saved! They are:
Jordan
Ying
```

Finally, we can ask our names `ArrayList` whether or not it has a particular string.

```
names.add(‘‘Veer’’);
System.out.println(names.contains(‘‘Veer’’)); // true
```

Example 4.0.4: Update your program to check if a name was input by the user. For example:

```
Please give me some names:
Ying
Jordan

Search for a name:
Ying
Yes!
```

An `ArrayList` can hold any type of object! For example, here is a constructor for an `ArrayList` holding an instance of a `Person` class:

```
ArrayList<Person> people = new ArrayList<Person>();
```

where `Person` is defined as

```java
public class Person {

    String name;
    int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return this.name;
    }

    public int getAge() {
        return this.age;
    }
}
```

Example 4.0.5: Modify our program to save the user's input names as Person instances. Rather than storing String objects in the ArrayList, store Person objects by constructing them with the input name. You'll need to use the Person constructor to get a Person instance!

Exercises

4.1

Write a class BlueBook that tells the user the price of their car, depending on the make, model, and year. You should use Car.java and the stencil file provided, BlueBook.java.

Your program depends on what cars your BlueBook supports, but here is an example program:

```
What is your car's make?
Toyota
What is your Toyota's model?
Corolla
What is your Toyota Corolla's year?
1999

Your 1999 Toyota Corolla is worth \$2000.
```

4.2 Notify the user if the car is not in your BlueBook.

4.3 Clean up main by putting your code for creating the ArrayList in a separate method. What type should the method return?

4.4 If the car is not in the BlueBook, ask the user to input the relevant data, construct a new Car instance, add it to your ArrayList.

References

1. https://github.com/accesscode-2-1/unit-0/blob/master/lessons/week-3/2015-03-24_arraylists.md