Welcome to CS206

Based on notes from Steve Kautz

Bryn Mawr College
CS206 Intro to Data Structures
Please turn off your cell phone!

Example: Trapping rain water

Given \( n \) non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.

For example, given \([0,1,0,2,1,0,1,3,2,1,2,1]\), return 6.
Example: Sudoku solver

- Write a program to solve a Sudoku puzzle by filling the empty cells

What are data structures & algorithms?

- A *data structure* is an arrangement of data in a computer’s memory (or sometimes on a disk).
- Data structures include arrays, linked lists, stacks, binary trees, hash tables, and others.
- *Algorithms* manipulate the data in these structures in various ways, such as searching for a particular data item and sorting the data.
Data Structures & Algorithms in Java

1. Programming in Java
2. Objects
3. Data structures & algorithms

What is programming?

- Well, it isn’t computer science
- Programming is like craftsmanship in building something
  - The end product is called software
  - Applications and systems
  - Informally: “code”
- In construction, architects design houses, craftsmen build them
  - Architects usually do not have the skills to build the things they design!
- In software, it never seems to work that way
  - There are software designers who don’t actually write code
  - ...but they all start out learning to write code
The way it is

- Virtually every university program in CS or SE or CPRE starts out with a couple of semesters of programming
- So, here we are 😊

You didn’t answer the question

- What is programming?
  - It just means writing out a sequence of instructions for a machine to carry out.
  - Instructions are usually very basic, e.g., “add these two numbers together”
- Any sequence of instructions can be called a “computer program”
Example: find the biggest number in a list

43 17 85 32 86 79 18

It’s easy to spot, right?
But what if you had a bigger list?


A strategy or “algorithm”

1. Look at the first number, and remember it
2. Read through the rows from left to right
3. If we’ve run out of numbers, then we’re done.
4. Otherwise, look at the next number and compare it to the maximum we remembered
5. If the new number is bigger, then remember that one instead
6. Go back to step 3
Programming

- We can turn these steps into a program by writing them down carefully in a *programming language*
- The statements in a programming language are translated, or *compiled*, into *machine instructions*
  - Numeric codes that control the millions of tiny electrical switches in the processor

---

Basic ingredients of computation

1. Store a value so we can remember it later
2. Do basic arithmetic
3. Check a condition and take some action, depending on whether the condition is true
4. Repeat some action, continuing as long as a condition is true
5. Get input or produce output
   
   *And that’s all any computer can do!*
Compiling and running

A simple Java program

- Write the program in a text editor
- Invoke the Java compiler to create the class file (machine instructions)
- Invoke the Java runtime to execute the compiled code

*We usually use an integrated development environment to perform these steps.*
A simple Java program

```java
public class HelloPrinter {
    /**
     * The 'main' method is always the entry point
     * for a Java application.
     */
    public static void main(String[] args) {
        // Display a greeting on a text console
        System.out.println("Hello, world!");
    }
}
```

But, what does it mean to “design” software?

Designing software

- Suppose you have a few hundred lines of instruction
  - Tic-tac-toe game, print loan table, sort list of names...
  - Well, this is probably just a “program”
- Applications like Word or Firefox may involve a million lines of code
  - Too complex for one person to understand...
  - ...unless very carefully designed!
Object-oriented design

• This is where the “OO” comes in
• Modern applications are too complex to be written as a simple sequence of instructions
• OO is a natural way of breaking down a complex system into components
  o Each component is simpler than the whole
  o You specify
    • What does each component do?
    • How do the components interact?

Analogy

A typical car consists of approximately 30,000 parts
Analogy

But it makes a lot more sense as a system of interacting components.

Objects

- In OO design and programming, the components are called “objects”
  - Within each component there is a sequence of instructions to execute...
  - But we understand an application as a system of interacting objects
Invoking a method

- The System.out object has a method called `println()`. We invoke the method using the “dot” operator
  - The expression we want to print goes in the parentheses, and is called the argument to the method
  - The semicolon at the end is just a required piece of syntax, forming a statement:
    ```java
    System.out.println("Hello, world!");
    ```

Classes

- The definition for a type of object is called a class
  ```java
  public class HelloPrinter {
    ...
    ...
  }
  ```
- “HelloPrinter” is an identifier (a name we chose for the class)
- `public` and `class` are keywords in Java
  – I.e., part of the syntax (grammar) for the language
A simple Java program

• Our class has one method, called main()
  ○ main() contains one *statement*
• The entry point for executing a program must always be a method called main with this form

```java
public class HelloPrinter {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

Language ingredients

• Keywords (reserved words such as public, class, void, ~50 others...)
• Identifiers (names such as HelloPrinter or System)
• Literal values (42, 3.14, “Hello, world!”)
• Operators (+, -, *, /, etc.)
• Syntax rules (“every statement ends with a semicolon”)
Compile errors

• If a syntax rule is violated, the compiler can’t translate the code into machine instructions
  o Compile errors are flagged in Eclipse with red squiggles
• Comments are ignored by the compiler
  o Comments are used to document code
  o See code example for two forms of comments
    • // - style for “internal” comments
    • /** - */ style for “Javadoc” comments

Every value has a type

• 8 primitive types
  – int for whole numbers
  – double for “floating-point” numbers
  – boolean for true/false values
  – (There are 5 others, byte, short, long, float, char, not used too much)
• Everything else is a kind of object
  – System.out has type PrintStream
  – “Hello” is a literal value of type String
Expression

- An *expression* is something that represents a value
  - As opposed to a *statement*, which means “do this”
- Every literal is an expression
- We can create new expressions by combining them with *operators*
  - (2 + 3) * 5
  - 3.14 / 2.0

Examples

- The `println()` method can be used to print many types of expressions, e.g.
  - `System.out.println((2 + 3) * 5);`
- (See sample code...)
Variables

• A variable can be used to store a value
• Must be declared first with a type
• Value is assigned using the symbol “=”
  – Called the assignment operator, does not mean “equals”!!
  – Works right-to-left only
• Examples (see sample code)

Restrictions on identifiers

• May contain only letters, numbers, and underscores
  o Compiler-generated identifiers may contain ‘$’
• Must start with letter or underscore
Conventions for identifiers

- Variable names start with lowercase letter
- Multiple words use camelCase
- Variable names should be meaningful
- Method names start with lowercase letter
- Class names start with upper case letter