5/5/14

1. Review Session on Wednesday
2. Same length as the midterm: 40 multiple choice, 10 definitions
   1. Focuses on the 2nd half of the course
   2. Tougher than the questions in the midterm
3. Topics in CS (Chapters 1-5, 7 and 12 in Schneider and Gersting)
   1. What is cs?
   2. What are algorithms?
   3. Representing algorithms in pseudocode
   4. Basic algorithm analysis
   5. Representing information using bits
   6. Circuit construction
   7. Intro computer organization
   8. **Networks**
   9. **Computability**
   10. You should know insertion, linear, bubble, euclid and binary sort
4. Java programming skills (Big Java: Chapter 1-11 and some of 12)
   1. No graphics stuff
   2. Variable assignment
   3. Primitive data types in Java
   4. Control structures
      1. Conditionals
      2. Loops
   5. Writing classes
      1. Instance variables
      2. Methods
   6. Static methods
   7. Static variables
   8. Parameter passing
      1. Passing primitives
      2. Passing object references
   9. Designing better classes
   10. Arrays and ArrayLists
   11. File I/O
   12. Interfaces
   13. Turing machines
   14. Exceptions
       1. Pg. 535
   15. Inheritance
       1. 4-5 questions
       2. Polymorphism
       3. Ex. In Foo

Public int Example(){

Return 3;

}

Is it going to return 3, is it a legal call?

* + 1. Ex. Foo- with subclass bar

Public class Foo{

Private int a

Protected int b

Public String c

Which variables are accessible to subclass bar? (Answer: b and c)

1. Final Design
   1. Question 1
      1. 10 terms to define
   2. Question 2
      1. 40 multiple choice
2. 80% is on the stuff after Spring break
3. 20% is on the stuff after Spring break
   1. Not really on Java stuff

5/7/14

Final Review

* Computer systems
* The Von Neumann architecture
  + Three characteristics
    - Four major subsystems: primary memory, I/O, ALU, and control unit
    - The stored program concept
    - The sequential execution of instructions
* Memory and cache
  + Memory: stores and retrieves instructions and data
  + Cache memory: smaller, faster memory in the processor
  + Average memory access time = (cache hit rate) x (cache access time) + (cache miss rate) x ((cache access time) + (RAM access time))
* ALU
  + Consists of circuitry for performing operations and registers for storing operands and results
* Control unit
  + Where machine language instructions are executed
  + Machine language instructions: OP CODE ADDRESS 1 ADDRESS 2…
    - Example: assume 8-bit op code
    - Example: if the op code is 6 bits long, what is the max number of possible instructions? (2^6 = 64 bits)
* Debugging
  + Two main types of bugs:
    - Compile-time errors
      * Any error that won’t let you compile
      * Primarily syntax errors
      * Missing semicolons, missing quotes around a String, improper method calls
    - Run-time errors
      * Will get past the compiler, but can cause the program to crash or simply be logic errors
    - Style and design issues are not bugs!
  + Example phonenumber
    - At least 6 errors
    - Hints of the lines of errors
* OO Design
  + Code reuse
  + Encapsulation: hiding details from the user, in the Java API
  + Polymorphism (inheritance and interfaces)
    - Idea of being able to do the same things to different kinds of objects
  + EVERYTHING IS AN OBJECT- except primitive data types
* Binary/hexadecimal number conversions
* Algorithms and analysis
  + Big-O notation
  + Takes into account worst-case performance
  + Measures how much of a resource is required with an increasing number of inputs
  + The final answer takes the fastest-growing term as the complexity
  + Selection sort
    - Iterate through list, and find minimum
    - Place minimum in “sorted section” of list
    - Repeat until list is sorted from least to greatest
  + Time complexity
    - For one iteration: as many items are in the list
    - For each subsequent iteration, there is one less element in the list
    - Time taken: n(n+1)/2
    - Big-O complexity: O(n^2)
  + Bubble sort
    - Each iteration, the largest element “bubbles” to the top of the list
    - Mechanism: switching two adjacent elements
    - Each element moves forward in the list until it encounters something bigger, in which case that element starts moving forward
  + Insertion sort
    - Maintain a “sorted section” of the list
    - Each iteration, add the first element of the unsorted list to the sorted list, by iterating through and seeing where the correct position is
    - Ex. Cards in a hand
  + Linear Search
  + Binary Search
    - O(log base 2 n)
* Networks
  + Network: a set of independent computers (nodes) connected by links to transfer information
  + LAN: “Local Area Network”
    - Connects hardware in close proximity
    - Various network configurations: bus, star, ring, etc
  + WAN: “Wide Area Network”
    - Nodes not in close proximity
    - Many dedicated point-to-point links
    - Store-and-forward, packet-switched technology
      * Packets must hop between different nodes to get to a destination
      * Advantages:
        + Packets allow large messages to be broken up and sent through the network quickly
        + Failure of one node or line does not necessarily bring down entire network
  + The Internet protocol stack: 5 layers
    - 5 Application layer (protocols: HTTP, FTP, etc)
    - 4 Transport layer (protocols: TCP, UDP, etc)
    - 3 Network layer
    - 2 Data Link layer
    - 1 Physical layer
  + Collisions (Data Link Layer)
    - Two or more messages try to get delivered on the same line
    - Ethernet protocol dictates that both nodes wait a random amount of time and then retransmit, earlier node will get the line
  + ARQ (automatic repeat request) algorithm
    - Ensures that messages arrive at their destination
      * A sends packet to B
      * If B successfully receives the packet, sends ACK to A
      * If B does not receive the packet (or A does not receive the ACK), A will wait some arbitrary amount of time and resend
      * B will automatically discard any mistakenly resent packets
  + Inheritance
    - Allows you to model parent-child or category-subtype relationships
    - Motivation: code ruse. If one class “extends” another class, it inherits methods and instance variables from that class so you don’t have to write them again!
    - What’s inherited?
      * Only public and protected methods and instance variables
      * That means that the child class can’t make any references to private methods or instance variables of its parents
    - What else?
      * Child classes can define their own additional methods – these cannot be accessed by the parent classes
      * Child classes can also override their parents’ methods
        + Overriding: is used when the method with the same name is in two different classes

Everything is the same “static/public/etc” including the parameters

* + - * + Overloading: two methods with the same name, but with different parameters
        + Cat can call any public methods even though those methods might contain private variables
      * toString methods override objects
    - Reference types vs. Object types
      * Cat myCat = new Cat(“Snowball”);
      * Reference type variable name = new Object type;
      * Animal myAnimal = new Cat(“Grumpy”);
      * The reference type determines what methods can be called
      * The object type determines which version of the method is called
      * The object type must be the same as the reference type or be a child of it
      * So…
        + myCat.scratch() compiles but myAnimal.scratch() won’t

Can’t call scratch method on myAnimal because its reference type is Animal, and the Animal class does not contain a scratch() method.

* + - * + myCat.makeSound() prints “meow” and myAnimal.makeSound() also prints “meow”

myAnimal can call the makeSound() method because the Animal class does contain a makeSound() method

* + - * Examples on PPT
* Interfaces
  + Allow you to define strict templates for classes
  + Classes can only “extend” one other class, but they can implement multiple interfaces: interfaces allow multiple inheritance
  + If a class implements an interface is must implement all its methods
  + Interfaces can be references types too
  + If the interface is a reference type, the only methods that object can call are the ones in the interface
  + Interfaces aren’t classes, so you can’t create objects of interfaces
  + You can’t instantiate an interface
  + Interfaces can only have constants, not instance variables
* Exceptions
  + Throws
    - Coming up in your method declaration
  + Throw command
    - Actually throws the exception
  + Try/Catch

Public void meow() **throws** MeowException{

//stuff

}

Public void meow(String name){

If(name.equals(“sireesh”)){

**Throw** new NeowException(“MEOW”);

}

}

Main() {

C

Try{

c.meow();

} catch(MeowException e){

Print e.getMessage();

} catch(FileNotFoundException e){

Print “no file, bro”;

} finally{

// acts like the else in the if/else loop

}

}

* Strings
  + There will usually be a question about Strings
  + charAt, dot equals method?
  + compareTo, comparable interface
  + The shortest String is length zero, ie. Empty quotes “ “
  + Concatenation
    - Strings can be combined with other Strings as well as some other data types
    - If a String is involved in an expression, the whole expression evaluates to a String
    - Concatenation follows order of operations

Public void meow(String n){ // String n is a formal parameter

Print n

}

Main(){

myCat.meow(“hi there”); // “hi there” is an actual parameter=argument, and an explicit param.

}

* Static variable
  + Where you have one value for the entirety for the class
  + One variable that shares the value from the same class

Public class BankAccount{

Private String name;

Private static int number = 0;

Public BankAccount(String name){

This.name = name;

Number++

}

}

In main:

BankAccount a = new BankAccount(“jim”); // number is 1

BankAccount b = new BankAccount(“bob”); // number is 2 for EVERY account now (ie. “SYNCING”)

* Static method
  + You don’t need an instance of a class to run it
  + Math.random();
  + Math.max(2,5); //returns an int
  + Math.sqrt(4); //don’t need to create a “Math” object

Tutoring