CS 304 Midterm Review

Xiwei Wang, Ph.D.

Assistant Professor

Department of Computer Science

Northeastern Illinois University

Chicago, Illinois, 60625

6 October 2016

Exam format

- 16 multiple choices (40pts)
- 3 short answers (19pts)
- 2 coding questions (26pts)

Topics

- Basic concepts of object-oriented programming (Chapter 1)
- Abstract data types and linked lists (Chapter 2)
- The `Stack` ADT (Chapter 3)
- Recursion (Chapter 4)
- The `Queue` ADT (Chapter 5)

Chapter 1

- Classes and objects
  - What is an object? What does an object represent?
  - An object is an instantiation of a class and a class defines the structure of its objects.
  - Objects represent
    - Information: we say the objects have attributes.
    - Behavior: we say the objects have responsibilities.
  - Observers (getters, accessors) and transformers (setters, mutators)
- Class inheritance
  - Classes are organized in an "is-a" hierarchy.

Chapter 1

- Reference variables vs primitive variables
  - A primitive type refers to a primitive data type.
  - A reference type refers to a class or an array.
- Linear structures: arrays, lists, stacks, and queues
- Time complexity
  - How does the time taken depend on the input size?
  - Order of complexity: big O notation.
    - Calculate the number of operations performed.
    - Take only the fastest-growing term (highest exponent).
    - Remove any constant factors.
    - $3n^2 + 6n + 100$ becomes O(n^2).
    - O(1) < O(log$n$) < O(n) < O($n$log$n$) < O(n^2) < O(n^3) < O(2^n)

Chapter 2

- Abstract data type (ADT)
  - A data type whose properties (domain and operations) are specified independently of any particular implementation.
- Three levels in ADTs
  - Logical (or abstract) level, implementation (or concrete) level, and application (or user, client) level.
- Java interfaces
  - Benefits of using interfaces.
  - An interface can be implemented by multiple classes.
- The **`StringLog`** ADT
  - **`StringLog`** methods - constructors, **`insert`**, **`clear`**, **`isFull`**, **`size`**, **`toString`**, **`contains`**

Chapter 2

- The `StringLog` ADT
  - Array-based vs linked-based implementations
  - Bounded and unbounded
- Linked lists
  - Creating a linked list
  - Creating a node
    - In a singly linked list, there are two fields: `data` and `link` (a reference to the next node).
  - Inserting and deleting a node
  - Traversing a linked list
    - For singly linked lists, you can only traverse from the head to the tail (or from the tail to the head).
  - Time complexities

Chapter 3

- Stack
  - Definition, LIFO, can only access from the top of the stack.
  - Stack methods: **`push`**, **`pop`**, **`top`**, **`isEmpty`**, **`size`**, (**`isFull`**)
  - Time complexities
  - Array-based vs linked-based implementations
  - Bounded vs unbounded
- Method call stack
  - A typical application involves many levels of method calls, which is managed by a so-called method call stack.
  - So if method **`A`** calls method **`B`** which calls method **`C`**, their activation records are stored on a stack structure.

Chapter 3

- Java exception mechanism
  - Definition
  - Three major phases: defining the exception, generating the exception, handling the exception.
- Exceptional cases in the `Stack` ADT
  - `pop` and `top` an empty stack: throw a `StackUnderflowException`
  - `push` onto a full stack: throw a `StackOverflowException`

Chapter 4

- Recursion: a method that calls it self.
  - The idea: solve a hard problem by first solving an easier version of the same problem
    - which you solve by finding an even easier version
    - until the problem is so easy you can solve it right away.
- Two components of a recursive method:
  - There must be special cases to handle the simplest tasks directly so that the function will stop calling itself. (Base case)
  - Every recursive call must simplify the task in some way. (Recursive call)

Chapter 4

- Example recursive definitions:
  - Factorial: Base case: $0! = 1$;

    General case: $n! = (n - 1)! * n$
  - Fibonacci: Base cases: $F(0) = F(1) = 1$;

    General case: $F(n) = F(n - 1) + F(n - 2)$
- Avoiding infinite recursion:
  - Must check the base case first.
  - No recursion in the base case!
  - And the recursive call(s) must bring you closer to the base case.
- Mutual recursion: `A` calls `B`, `B` calls `C`, `C` calls `A`.
- Iterative solutions vs recursive solutions (efficiency, memory)

Chapter 5

- Queue - A structure in which elements are added to the rear and removed from the front; a "first in, first out" (FIFO) structure.
  - Queue methods: **`enqueue`**, **`dequeue`**, **`isEmpty`**, **`size`**, (**`isFull`**, **`enlarge`**)
  - Time complexities
  - Array-based vs linked-based implementations
  - Fixed front design vs floating front design
  - Bounded vs unbounded
    - Using array to implement the unbounded **`Queue`** ADT.

Chapter 5

- Using `Queue` and `Stack` to check for palindromes.
- Circular linked queues
- Exceptional cases in the `Queue` ADT
  - `dequeue` an empty queue: throw a `QueueUnderflowException`
  - `enqueue` onto a full queue: throw a `QueueOverflowException`