Recitation Guide for March 24th, 2008

I. Housing Keeping
   a. Homework 6 part 1 – solution available on T^2
   b. Homework 6 part 2 – due March 28th
   c. Homework 6 Demo Signups – due March 28th
      i. Signup page: http://coweb.cc.gatech.edu/cs1316/1050
      ii. Rules available on T^2
   d. Quiz 3 – in class Wednesday April 2nd
   e. Bonus Quiz Question – solution available on T^2

II. Abstract Data Types (ADTs), abstract classes and interfaces
   a. See http://coweb.cc.gatech.edu/cs1316/uploads/336/Spring_2008_Abstract_Data_Structures_%28ADTs%29_Stacks_and_Queues.pdf
   b. Interfaces
      i. Code template with just abstract methods, method headers with no method bodies
      ii. Uses keyword implements when it is used by another class
         1. i.g. public class ButtonListener implements ActionListener
      iii. Any implementing class is forced to "give the abstract methods a body."
      iv. A good way to think about interfaces is to think of it as a set of rules (method headers) for let us say Georgia Tech club sports. Assume that one of the rules is that all Georgia Tech club teams must wear yellow tops and black bottoms, but the style (method body) of the clothes may differ with the purpose and goal of each sport. The ultimate team may wear yellow t-shirts and black shorts, while the club basketball team might wear yellow basketball jerseys and black basketball shorts. Both satisfy the requirements, but in different ways.
   c. Abstract classes
      i. Class containing regular methods and abstract methods
      ii. Uses keyword extends when it is used by another class
         1. i.g. public class exampleClass extends exampleAbstractClass
      iii. Any extending class is forced to "give the abstract methods a body."
      iv. An abstract class means that some methods are already provided and it includes methods headers for more methods that you must implement yourself. In the same example from before, Georgia Tech decided that it will provide all club sports with water and sport drinks (predefined classes), but still requires all teams to follow the previous dress code (define the empty methods).
   d. Abstract Data Types
      i. Queue (pronounced Q)
         1. FIFO (first in, first out) or LILO (last in, last out) structure
         2. Example: People in a line

3. Methods
    a. dequeue (pronounced D-Q) – removal method
    b. enqueue (pronounced N-Q) – insertion method
    c. Insertion and removal occur at different ends. Normally insertion occurs at the tail and removal at the head. Can occur in the other sense depending on implementation.
4. Will be specially important for discrete simulations.

ii. Stack
1. LIFO(last in, first out) or FILO (first in, last out) structure
2. Example: A stack of papers, a stack of dishes
3. Methods
    a. pop – removal method
    b. push – insertion method
    c. Insertion and removal occur at the same end.

iii. For more information on ADTs see: http://en.wikipedia.org/wiki/Abstract_data_type

III. Introduction to Random class and Math.random()

a. Math.random()

i. Returns a random double from 0 to 1 excluding 1 itself, also abbreviated [0,1)

ii. Example: simulating the roll of 1 die
1. 1 die has numbers from 1 – 6.
2. First we need to make the number Math.random() returns an integer.
   int dieRoll = (int)(Math.random()); //returns 0
3. Great we have succeeded in creating a method that always return 0. Try to make it so that it will always return 0 – 6.
   int dieRoll = (int)(7 * Math.random()); //returns 0 - 6
4. How do we make it return 1 – 6 now? What if we add 1 to the result from the previous one?
   int dieRoll = (int)(7 * Math.random()) + 1; //1 - 7
5. Okay closer now. We need to change the inner part to return 0 – 5 and then add 1 to the result.
   int dieRoll = (int)(6 * Math.random()) + 1; //1 – 6
6. Final answer:
```
public int dieRoll(){
   int dieRoll = (int)(6 * Math.random()) + 1;
   return dieRoll;
}
```

iii. Example: simulating the roll of 2 dice
1. Answer 1, using the answer from before:
```
public int twoDiceRoll1(){
  int dieRoll1 = dieRoll();
```

```
  int dieRoll2 = dieRoll();
  int sum = dieRoll1 + dieRoll2;
  return sum;
}
```

2. Answer2 : two dices has possible numbers from 2 – 12

```
public int twoDiceRoll2(){
  int twoDiceRoll = (int)(11 * Math.random()) + 2;
  return twoDiceRoll;
}
```

iv. Probability

1. Sam has a 5% chance of winning the company raffle. How do we represent this probability?

```
public boolean winTheRaffle(){
  if (Math.random() < 0.05 )
    return true;
  else
    return false;
}
```

b. Random class

i. API: http://java.sun.com/javase/6/docs/api/java/util/Random.html

ii. Includes more functionality than Math.random() and can return types other than double, such as booleans, longs and floats.

iii. Another solution for the die roll:

```
//must import java.util.Random;
public int dieRoll2(){
  Random gen = new Random();
  int dieRoll = gen.nextInt(6) + 1;
  return dieRoll;
}
```

IV. Sorting Algorithms

a. Insertion Sort

i. How it works

1. The insertion sort algorithm sorts a list of values by repetitively inserting a particular value into a subset of the list that has already been sorted. One at a time, each unsorted element is inserted at the appropriate position in that sorted subset until the entire list is in order. Each time an insertion is made, the number of values in the sorted subset increases by one. The insertion process requires that the other values in the array shift to make room for the inserted element.

ii. Needed to for Discrete Simulations

iii. Big O (Efficiency notation)

1. Best case: n (for an already sorted list)

2. Worst case: n^2 (for a list sorted in reverse order)

iv. Example

[ ] 9 2 1 3 5 10 4 7 8 6, where [sorted list] unsorted list

1. [9] 2 1 3 5 10 4 7 8 6
2. [2 9] 1 3 5 10 4 7 8 6
3. [1 2 9] 3 5 10 4 7 8 6
4. [1 2 3 9] 5 10 4 7 8 6
5. [1 2 3 5 9] 10 4 7 8 6
6. [1 2 3 5 9 10] 4 7 8 6
7. [1 2 3 4 5 9 10] 7 8 6
8. [1 2 3 4 5 7 9 10] 8 6
9. [1 2 3 4 5 7 8 9 10] 6
10. [1 2 3 4 5 6 7 8 9 10]

b. Other sorts

i. Bubble sort, selection sort, radix sort, heap sort, quick sort, merge sort.

V. Introduction to Simulations

a. Simulation – a representation of a system of objects in a real or fantasy world. The purpose of creating a computer simulation is to provide a framework in which to understand the simulated situation. Such simulations make it possible to collect statistics about these situations and to test out new ideas about their organization.

i. Continuous simulations – each moment of time is simulated.
Ex: WolfDeerSimulation.

ii. Discrete simulations – simulation is advanced from event time to event time.
Ex: FactorySimulation, TrafficSimulation (HW7)