Odds and Ends

- Think about doing research
- Programming grading
  - Correctness, Documentation, Style, Testing
  - Working in pairs/groups; acknowledge accordingly
- Assignment 3 due Tuesday
Questions on HW

• ....?
Review

• What are the three most important aspects of type?
• How is an interface different from a class?
• What is difference between object identity and equality?
• How is ADT different from interface?
Java Moment: Arrays

- An array is a contiguous block of memory (think Scheme vector)
- Declarations
  - int[] foo; char[] gak; Object[] baz;
- Space allocation
  - foo = new int[10];
  - gak = new char[199];
  - baz = new Object[5];
- Literal initialization
  - int[] jub = {1, 2, 3, 4, 5, 6}
Java Moment: Arrays

• Accessing
  - Integer index starting at 0 for the first element
  - int[] jub = {1, 2, 3, 4, 5, 6}
  - System.out.println(jub[4] + " is 5");

• Mutation
  - System.out.println(jub[4] + " now 6");
  - jub[4]--;  
  - System.out.println(jub[4] + " is 5 again");
Chapter 3

Algorithms
Outline

- Introduction
- Characteristics of algorithms
- Recipes as algorithms
- Analyzing computer algorithms
- Recursive algorithms
Introduction

- Levels of abstraction for the Java program
  - Community of objects providing and consuming services
  - Services of individual members of community
    - Ignore who requests and how service is provided
    - Example: Java interface
  - Concrete implementation of abstract behavior
    - Example: Array implementing bag
  - Sequence of actions to accomplish single task
Characteristics of Algorithms

• What an algorithm is: precise process description
• Requirements
  • Accurate specification of input
  • Precise specification of each instruction
  • Correctness (contrast with heuristic)
  • Termination, time to execute
  • Description of the result or effect
Recipes as Algorithms

- Consider the characteristics of recipes compared to those required for algorithms
  - Accurate specification of input: ok
  - Precise specification of each instruction: no
  - Correctness: ok
  - Termination, time to execute: ok
  - Description of the result or effect: yes
Analyzing Computer Algorithms

- Specification of the input:
  - Type checking
  - Preconditions
  - Conditional branches and exceptions
  - Comment assumptions
Type Checking

```java
public int min(int a, int b) {
    // return smaller of two integer arguments
    if (a < b)
        return a;
    else
        return b;
}
```
public char digitChar(int val) {
    // pre: val must be between zero and nine
    // post: return the character for val
    switch (val) {
        case 0: return '0';
        case 1: return '1';
        ....
        case 9: return '9';
    }
}
public char digitChar(int val) {
    // pre: val must be between zero and nine
    // post: return the character for val
    if ((val < 0) || (val > 9))
        throw new Exception("arg out of range");
    int d = '0' + val;
    return (char) d;
}
public double power(double base, int n) {
    // pre: n is larger than or equal to zero
    // post: return the result of raising the
double-precision base to the integer exponent
    // assumes floating-point overflow does not occur
double result = 1.0; // based raised to zero
    // raise base to new powers
    for (int i = 1; i <= n; i++) {
        result *= base;
    }
    return result;
}
Analyzing Computer Algorithms

- Description of the result
  - Name of the function
  - Comment describing result
  - Documenting side effects
public int min(int a, int b){
    // return the smaller of two arguments
    ...
}

public void printInt(OutputStream out, int val){
    // print the character representation of arg val
    if (val < 10)
        out.write(digitChar(val));
    else {
        printInt(out, val/10);
        out.write(digitChar(val%10));
    }
}
Analyzing Computer Algorithms

- Instruction precision
  - Computers perform simple instructions rapidly
  - But only
  - Cannot cope with ambiguity found in process descriptions expressed in natural language
Analyzing Computer Algorithms

- Time to execute
  - Will the algorithm terminate at all?
  - If so, how much time will it take?

- Termination requires some property with three qualities
  - Placement in one-to-one correspondence with integers
  - Non-negative
  - Steadily or monotonically decreasing
public double power(double base, int n) {
    // pre: n is larger than or equal to zero
    // post: return the result of raising the
double-precision base to the integer exponent
    // assumes floating-point overflow does not occur

double result = 1.0; // base raised to zero
    // raise base to new powers
for(int i=1; i <= n; i++){
    result *= base;
}
return result;
}
Public int gcd(int n, int m) {
    // pre: assume args are greater than zero
    // post: compute greatest common divisor of args
    while (m != n) {
        if (n > m) {
            if (n > m)
                n = n - m;
            else
                m = m - n;
        }
        return n;
    }
    // what property to use?
Termination of Recursive Function

• How does our property and its characteristics apply to recursive functions? (e.g., printInt())
• Same idea
  - Establish mapping to integers
  - Value must never be negative
  - Value must be monotonically decreasing
Zeno's Paradox

- Seems to raise problems for our termination conditions
- Achilles and the Tortoise
  - If the Tortoise is given a head start
  - Achilles cannot overtake it

- Where is the breakdown of the conflict?...
  - ...

Analyzing Computer Algorithms

- Space utilization
  - Some algorithms require more space than other (accomplishing the same task)
  - For example, sorting algorithms
    - Copy and sort
    - Sort in place
  - Utilizing more memory sometimes causes alg to run faster
    - *time/space tradeoff*
Recursive Algorithms

• A powerful and elegant style of designing algorithms
• Always consists of
  – A non-recursive base case
  – One or more recursive cases where the algorithm is called again with different arguments
• Each call gets new memory for arguments and local variables (recursive or not)
Towers of Hanoi

• Classic puzzle with elegant recursive solution
• The puzzle:
  - Disks of increasing diameter starting on one of three pegs
  - Move the top disk from one peg to another, **only if** the target peg does not contain a smaller disk
  - Object is to move stack from first peg to second peg
• How to formulate this algorithm?
  - ....
public void solveHanoi(int n, char a, char b, char c) {
    // move n disks from peg a to peg b using c
    // first move all but last to peg c
    solveHanoi(n-1, a, c, b);
    // then move one disk from a to b
    System.out.println("move disk from " +a+ " to " +b+);
    // finally move the stack from c to b
    solveHanoi(n-1, c, b, a);
}

// what is the problem here???
public void solveHanoi(int n, char a, char b, char c){
    // move n disks from peg a to peg b using c
    if (n == 1) { // can move smallest disk directly
        System.out.println("move disk from "+ a +" to "+ b);
    } else {
        // first move all but last to peg c
        solveHanoi(n-1, a, c, b);
        // then move one disk from a to b
        System.out.println("move disk from "+ a +" to "+ b);
        // finally move the stack from c to b
        solveHanoi(n-1, c, b, a);
    }
}
Analysis of Algorithms

• This is part of an area of concentration within Computer Science, *Theory*
  - Less software development
  - More math and formal proofs focus
• Dominant areas of Computer Science
  - Software Engineering
  - Architecture
  - *Theory*
  - Artificial Intelligence
  - Programming Languages
  - Social Impacts of Computing