COM 1101 Algorithms and Data Structures 1
Syllabus and Other Information for Winter 2001

Viera Proulx	Richard Rasala	Jeff Raab
117 Cullinane Hall	109 Cullinane Hall	239 Cullinane Hall
617-373-2225	617-373-2206	617-373-5876
vkp@ccs.neu.edu	rasala@ccs.neu.edu	jmr@ccs.neu.edu

Course Web Site: `http://www.ccs.neu.edu/course/com1101/`

1. Purpose

The purpose of the course COM 1101 is to introduce fundamental *algorithms and data structures* in the framework of *objects and classes* using *Java* as the programming language. Questions of design will be important on four levels: the design of individual algorithms, data structures, and classes; the use of classes that interact with one another; the use of software toolkits, especially for the creation of graphical user interfaces; and the design of the program structure as a whole. Graphics and visualization will be incorporated whenever possible.

2. Prerequisites

The course assumes an introductory knowledge of Java such as is being taught in COM 1100. Students may also take this course with a similar background in C or C++. Topics required include: variables and primitive basic types (`byte`, `short`, `int`, `long`, `float`, `double`, `char`, `boolean`); loops (`for`, `while`, `do-while`); decisions (`if`, `switch`); functions, parameters, and return values; the character `String` class; and the basics of one-dimensional arrays. Since some students enter this course with a C or C++ background, we will not assume a knowledge of classes and objects in Java but will introduce the necessary information as we proceed.

The course will use the *Java Power Tools* developed by Jeff Raab, Richard Rasala, and Viera Proulx to enable the rapid development of graphical user interfaces in Java. These tools will *not* be a direct focus of the course but as you work with the interfaces created by the instructors you will gradually learn how to build graphical user interfaces on your own.

3. Text Books

Peter van der Linden, *just JAVA*, 4th Edition, Sun Microsystems Press (Prentice-Hall), 1999, ISBN: 0-13-010534-1; or 5th Edition, 2002, ISBN: 0-13-032072-2.

Michael T. Goodrich & Roberto Tamassia, *Data Structures and Algorithms in Java*, 2nd Edition, John Wiley & Sons, 2001, ISBN: 0-471-38367-8.

4. Syllabus

A: Java Foundations

A1. Primitives data types: `byte`, `short`, `int`, `long`, `float`, `double`, `char`, `boolean`.

A2. Classes.

Classes, objects, references, and dynamic object construction using `new`. Class structure:

Definitions for individual objects: member data and member functions (methods).

Definitions for the shared information of the class: static data and static functions.

The fundamental duality principle for objects: *An object may be viewed as a collection of data with associated functions that provide operations on that data or an object may be viewed as a collection of functions whose behavior is determined by the data within the object.*

Variations of classes: abstract classes and interfaces.

Building classes: composition, inheritance, nested definitions, and on-the-fly definitions.

Building functions within classes: statements, loop control (`for`, `while`, `do-while`), decision control (`if`, `switch`), function call, dynamic method dispatch, parameters, access to member data, and function return values.

The fundamental design principle for functions: *Build small functions that call other functions to do the components of a task rather than build large functions that try to do everything "in line".*

A3. Exceptions or *breaks in the normal flow of the program control*: why exceptions are necessary and how exceptions are implemented (`try`, `catch`, `throw`).

A4. Threads or *doing many things at once*: how to permit multiple activities to automatically share the computer's processor time and appear to be happening simultaneously.

A5. Graphics and graphical user interfaces: tools built into Java (AWT, Swing, 2D Graphics) and the *Java Power Tools* extensions for rapid GUI development.

B. Algorithmics

The most basic principle of algorithmics is *divide-and-conquer* in which a problem is divided into smaller sub-problems, these sub-problems are solved one-by-one, and then the solutions to the sub-problems are integrated into a solution of the original problem.

The divide-and-conquer principle has two major variations:

Loops: Here a problem is divided into one that is solved by executing the cycles of the loop.

Functions that call functions: Here a problem is divided into tasks that are solved by calling functions within the original function to do the separate tasks.

The technique of *functions that call functions* is perhaps the most fundamental technique in all of computer science. An important special case of this technique is *recursion* in which a function directly or indirectly *calls itself* to work on a subtask of a problem.

The course will illustrate the divide-and-conquer principle both by using loops and nested loops and by studying numerous situations in which functions call functions. In particular, the course will study loop patterns, cooperating classes, recursion, algorithmic complexity, and function objects. The technique known as *encapsulate action as object* will be used throughout the course.

C: Data Structures

Data structures maintain the information necessary for the performance of the algorithmics of a program. In Java, the class is the building block for data structures. Because the member data of a class can consist of both primitive types and objects of simpler classes, *it is possible to build structures of arbitrary levels of nesting and complexity using classes*. Indeed, one can think of recursion as a principle that applies to both algorithms and data structures.

In many structures, it is necessary to have several entities of the same type. To support this, Java has an *array* construct that permits you to define such a sequence. The size of an array may be specified at compile time or at runtime when the array is created. Once the array is created, its size may not however be changed.

A Java array is not a primitive type but neither is it fully an object. An array shares some of the properties of objects but there are subtle restrictions. For example, one cannot define a class that extends an array and thereby add special methods to the array. Nevertheless, arrays can be used very effectively both as tools within functions and as components within objects.

The dynamic data structures provided in the Java libraries often use arrays internally to maintain data. To allow these dynamic data structures to change their effective size, the methods that control the size will build a new array, copy the data from the original array to the new array, and then discard the original array. In this way, something that is fixed size (the array structure) can be used to implement something whose size may vary.

This course will focus on data structures that are *simple collections of components* and on data structures that are *linear*, that is, represent an array-like sequence of entities. You will learn how to build and work with such structures. For example, we will study how the Java `vector` class is designed. The course will also introduce and use hash tables that permit one to retrieve information by providing a key. In effect, a hash table behaves like a fancy array whose indices can be general keys rather than simply a sequence of integers.

This course will *not* discuss data structures such as linked lists, trees, and graphs whose implementation requires the concept of *linking nodes*. This is discussed in the next course COM 1201. In particular, since hash tables require such techniques in their implementation, we will use hash tables but not discuss how they are constructed. This will illustrate the very important fact that you can use a structure based on its *application programming interface* (API) without knowing how the structure is built. It is the ability to use something without knowing its details that makes the modern software industry possible. If everyone had to know everything about everything, it would be impossible to create significant software.

5. Readings in the Text Books

Reading technical material is not the same thing as reading a newspaper or reading a novel. The material is usually much more densely packed and many words have a specific technical meaning that is not the same as in ordinary English. You should therefore plan to read in at least three stages:

1. Read a chapter fairly rapidly to find out what topics are in the chapter, what words are introduced, and what are the main concepts and the main technical issues.

2. Read the chapter a second time more slowly to try to learn as much as possible of the details.

3. Put the chapter aside and then come back later to individual sections that should be read in depth as needed.

Also, be aware, that not all of the material in the books will be taught in the course so you will be able to skip some sections.

The readings in *just Java* may be divided into 6 major areas:

1. General introduction to Java and to classes and objects.
   Chapter 1: Java philosophy and hype
   Chapter 2: Introduction to Java and object-oriented programming
   Chapter 3: Examination of a sample Java program and reflections on general issues.
2. Elements of Java
   Chapter 4: Identifiers and types
   Chapter 5: Expressions, operators, mathematical functions
   Chapter 7: Java statements and control structures
3. Elements of object-oriented programming in Java
   Chapter 6: Inheritance and polymorphism
   Chapter 8: Interfaces
   Chapter 9: Packages, inner classes, and on-the-fly class definitions
4. Java and data structures
   Chapter 5: Arrays
   Chapter 15: Java data structures library (&, in 5th, Pattern Matching)
5. Java input-output, graphics, and graphical user interfaces
   Chapter 13 & 14 (5th): Input-Output
   Chapter 17 (4th) or 19 (5th): GUI Basics
   Chapter 18 (4th) or 21 (5th): Swing
   Chapter 19 (4th) or 22 (5th): Containers and layouts
   Chapter 20 (4th) or Appendix A(5th): Graphics, colors, fonts, text, images, sound
6. Other Java facilities [discussed only to a very small extent in the course]
   Chapters 10 & 11: Threads
   Chapter 21 (4th) or 14 (5th): File I/O
   Chapter 22 (4th) or 17 & 18 (5th): Networking

Note that we will use the Java Power Tools developed at CCS to make GUI building much easier than in pure Java.

The readings in *Goodrich & Tamassia* will touch on those aspects of data structures that will be covered in COM 1101. Note that approximately half of the topics in the book will be covered in the next course COM 1201. The chapters relevant to COM 1101 are:

Chapter 1: Quick overview of Java programming
Chapter 2: Object-oriented design
Chapter 3: Making estimates for space usage and execution time
Chapter 5: Linear structures: arrays, vectors, lists, sequences
Chapter 10: Sorting

Java and Object-Oriented Design Reference Books

1. Fundamentals of Java

Peter van der Linden, *just JAVA*, 4th Edition, Sun Microsystems Press (Prentice-Hall), 1999, ISBN: 0-13-010534-1; or 5th Edition, 2002, ISBN: 0-13-032072-2.

Patrick Niemeyer & Jonathan Knudsen, *Learning JAVA*, O'Reilly, 2000, ISBN: 1-56592-718-4.

Ken Arnold, James Gosling, & David Holmes, *The Java Programming Language*, 3rd Edition, Addison-Wesley, 2000, ISBN: 0-201-70433-1.

Cay S. Horstmann & Gary Cornell, *core JAVA, Volume 1 Fundamentals*, Sun Microsystems Press (Prentice-Hall), 2001, ISBN: 0-13-089468-0.

Cay S. Horstmann & Gary Cornell, *core JAVA, Volume 2 Advanced Features*, Sun Microsystems Press (Prentice-Hall), 2000, ISBN: 0-13-081934-4.

Mary Campione, Kathy Walrath, & Alison Huml, *The Java Tutorial: A Short Course on the Basics*, 3rd Edition, Addison-Wesley, 2001, ISBN: 0-201-70393-9.

Mary Campione, Kathy Walrath, & Alison Huml, *The Java Tutorial Continued: The Rest of the JDK*, Addison-Wesley, 1999, ISBN: 0-201-48558-3.

2. Java Graphics

Jonathan Knudsen, *JAVA 2D Graphics*, O'Reilly, 1999, ISBN: 1-56592-484-3.

Lawrence H. Rodrigues, *Building Imaging Applications with Java Technology*, Addison-Wesley, 2001, ISBN: 0-201-70074-3.

3. Java Graphical User Interfaces

Kim Topley, *CORE Java Foundation Classes*, Prentice-Hall, 1998, ISBN:0-13-080301-4.

Kim Topley, *CORE Swing advanced programming*, Prentice-Hall, 2000, ISBN:0-13-083292-8.

Robert Eckstein, Mark Loy, & Dave Wood, *JAVA Swing*, O"Reilly, 1998, ISBN: 1-56592-455-X.

4. Algorithms and Data Structures in Java

Michael T. Goodrich & Roberto Tamassia, *Data Structures and Algorithms in Java*, 2nd Edition, John Wiley & Sons, 2001, ISBN: 0-471-38367-8.

David A. Watt & Deryck F. Brown, *Java Collections: An Introduction to Abstract Data Types, Data Structures, and Algorithms*, John Wiley & Sons, 2001, ISBN: 0-471-89978-X.

5. Object-Oriented Design and UML

Barbara Liskov with John Guttag, *Program Development in Java: Abstraction, Specification, and Object-Oriented Design*, Addison-Wesley, 2001, ISBN: 0-201-65768-6.

Erich Gamma, Richard Helm, Ralph Johnson, & John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995, ISBN: 0-201-63361-2.

Kent Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 2000, ISBN: 201-61641-6.

Martin Fowler, *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 1999, ISBN: 0-201-48567-2.

Martin Fowler with Kendall Scott, *UML Distilled*, 2nd Edition, Addison-Wesley, ISBN: 0-201-65783-X.

6. Advanced Algorithmics

Thomas H. Corman, Charles E. Leiserson, Ronald L. Rivest, & Clifford Stein, *Introduction to Algorithms*, 2nd Edition, MIT Press & McGraw-Hill, 2001, ISBN: 0-262-03292-7.

Algorithms and Data Structures. The Basic Toolbox. Prof.Â complex transport mechanism for data between memory and the processing unit, but they will have a similar effect for all i, and hence the number of primitive operations is also representative of the running time of an actual implementation on an actual machine. The argument extends to multiplication, since multiplication of a number by a one-digit number is a process similar to addition and the second phase of the school method for multiplication amounts to a series of additions. Let us conrm the above argument by an experiment. Figure 1.1 shows execution times of a C++ implementation of the s