Chapter 1: Introduction
A. Computing as a fact of life
B. What is computing anyway?
   i) Computing as information processing
   ii) Computing as a revolution
C. The past: A (very) brief history of computing
   i) Living in scaffolding - a cautionary tale
   ii) Why are we still using this prototype?
   iii) Evolution of computing
D. Juggling (!)
   i) Learning to program
   ii) The Approach used here: Less is more; more is less
E. Conclusion

Chapter 2: Programming: A quick dip in the pool
A. Introduction
   i) Learning to program
   ii) Problem Solving
   iii) The two types of Java programs: Applications and Applets
      i) A high level typology
      ii) A template for every Java main method
B. Examples:
   i) The first example of a class - A Robot Greeter
   ii) The second example of a class - a personalized RobotGreeter
   iii) Third example: A minimal Applet
C. Conclusion

Chapter 3: Class design and implementation
A. Introduction
   i) A description of the task
   ii) Before beginning to program: Design!
B. Building and testing the prototype GUI
   i) Getting Started
   ii) Using the Button to alter the TextField
   iii) Simulating one bank account by hand (without writing the Account class)
C. A generic problem solving technique
D. Account class: Design, implementation and testing
   i) Account class design
   ii) Converting the design to Java code
   iii) Objects and classes
E. Creating and testing the finished GUI
   i) GUI design
   ii) GUI implementation
F. The Bank class: Design, implementation and testing
   i) Bank class design
   ii) Converting the design to Java code
G. Putting it all together - finally!
H. Conclusion

**Chapter 4: Graphics and Inheritance**

A. Introduction
   i) A description of the task
   ii) Creating a prototype
   iii) Object Oriented Design -- choosing classes to implement

B. The Graphics class
   i) The Graphics context
   ii) Inheritance, Components and public void paint(java.awt.Graphics)
   iii) Basics of graphics in Java

C. The Circle class -- design and implementation
   i) Circle class design
   ii) Converting the design to Java code
   iii) Testing your code

D. Displaying a Circle graphically
   i) public void paint(java.awt.Graphics)
   ii) Testing the paint method
   iii) More than one Circle

E. The Color class
   i) Setting the color of the Graphics context
   ii) Built in Colors
   iii) Creating your own Colors

F. The Eye class: design and implementation
   i) Designing an Eye class
   ii) class FilledCircle extends Circle
   iii) Testing FilledCircle
   iv) The Eye class
      a) variables
      b) methods
      c) testing
      d) debugging

G. Assembling a working Eyes program

H. Conclusion

**Chapter 5: Towards consistent classes**

A. Introduction

B. Details I - Statements in Java: syntax and semantics
   i) Syntax and Semantics
   ii) BNF notation
   iii) BNF, Java and adaptive systems
   iv) The assignment statement
   v) The message statement
   vi) How to generate a Null Pointer Exception
   vii) The return statement

C. The basics of classes
   i) Variables I (state)
   ii) Methods (control)

D. The ClassMaker tool
   i) Motivation
   ii) ClassMaker input and output

E. Constructors
Chapter 6: Software reuse
A. Introduction
B. Inheritance
   i) The power of inheritance
   ii) The Object class
   iii) The mechanics of message sending
   iv) This is super!
C. Composition
D. Composition Programming Example: Snowpeople
   i) A description of the task
   ii) Overall Design
   iii) SnowPerson Design
   iv) Implementation
E. Conclusion

Chapter 7: Conditional statements
A. Introduction
   i) Procedural programming and control structure
   ii) Object programming allows you to substitute class structure
B. Different actions depending on conditions - Conditional execution
   i) The if statement -- do something or don’t
   ii) if-else -- do one thing or another
   iii) cascaded if-elses -- do one of a number of things
   iv) The switch statement
C. Programming example: using the SingleScoreConverter class in a tennis score keeping program
   i) A description of the task
   ii) Design
   iii) Making it smaller; let’s just play a single game
   iv) Testing
D. Conclusion

Chapter 8: Iterative statements and Strings
A. Introduction
   i) Repetition
   ii) On being conscious
   iii) Imitation and culture, or monkey see, monkey do?
   iv) This chapter
B. Iteration: Repeated action
C. The while loop
   i) Syntax and semantics
   ii) Examples

D. The for loop
   i) Example
   ii) The empty statement
   iii) An infinite for loop

E. Strings: a very brief introduction
   i) A few String methods
   ii) Breaking lines using Strings

F. Conclusion

Chapter 9: Simulation and animation
A. Introduction
B. An introduction to Threads
   i) Simplest threaded animation
C. The programming task
   i) Design
   ii) Implementation
D. Recapitulation
E. Conclusion

Chapter 10: Reading and writing files
A. Introduction
   i) File I/O
   ii) The Model-View-Controller pattern
   iii) On ignorance, stupidity, and utilities
B. The programming task
C. java.util.StringTokenizer
   i) Isolating the I/O
   ii) Simplest example
   iii) Sample exam question
   iv) The other StringTokenizer constructors
D. The MyReader class
   i) Why MyReader?
   ii) Echoing a user specified file
   iii) MyReader internals
   iv) Emitting the tokens one per line
E. Writing to a file
F. Putting it all together
G. Conclusion

Chapter 11: Data structures
A. Introduction
B. Arrays
   i) Simplest examples
   ii) Printing a String backwards
   iii) An array of Accounts
C. ArrayList<type>
i) add(<type>)
ii) for each
iii) Simplest test program

D. A simple bank database
   i) The database
   ii) Inputting the database: load
   iii) Outputting the database: save
   iv) Enhancing the DBMS

E. Molecules in box
   i) The programming task
   ii) The Molecule class
   iii) Changes to the Controller
   iv) Experimenting with the program

F. Conclusion

Chapter 12: Sorting lists

A. Introduction

B. Intuition for three sorts
   i) Insertion sort
   ii) Selection sort
   iii) Bubble sort

C. Algorithm/Pseudocode
   i) Insertion sort
   ii) Selection sort
   iii) Bubble sort

D. Implementation
   i) Bubble sort implementation
   ii) Insertion sort implementation
CS141 Fall 2010: Learning Objectives

To pass this course students are expected to master the following.

**Basics of programming**

Elementary algorithm design, including:
1. Searching
2. Sorting

Elementary class design, including:
1. State (variables)
2. Control (methods)
3. Accessors
4. Testing
5. Debugging

Control structure, including:
1. Conditional statements
2. Iterative statements
3. Recursion

Java language elements, including:
1. Classes declaration
2. AWT class structure
3. Statements
   - assignment
   - message
   - return
   - if-else
   - while
   - for

**Information**

**Types**
All information in a Java program has a type. There are primitive types, built-in classes and user-defined classes. It is possible to change types by casting.

**Values**
Expressions have values. To compute the value of an expression it is evaluated. Every type has a range of legal values.

**Variables**
Variables hold information. Every variable has a name, a type and a value. A variable only holds one value at a time. There are five different kinds of variables: instance, parameter, method, loop, and class.
Expressions
Variables and constants may be combined in arbitrarily complex fashion to form an expression. Syntactically, expressions appear to the right of assignment operators and as actual parameters (Thus far. Later on you will see them other places.).

Language Elements
Classes
A class is a template for objects of that type. It includes both variables and methods. Every object of that type has all the instance variables and can use all the methods declared in it.

If a method is declared static, it is not an instance method, but a class method. If a method does not use any state information from an instance (i.e. any instance variables) it can be made into a class method.

Variables
Every instance of a class has its own copy of each instance variable. All instances share access to class variables which are stored in the class itself (interestingly, classes are also objects, they are instances of the class class).

Methods
Methods have a heading and a body. The heading specifies the type, name and parameters of the method. The method body is a single block statement, which is a pair of {}s around a series of statements. To execute the method, Java executes each of those statements in order.

Constructors
Constructors are typeless methods with the same name as the class that are executed when a new object of that class is constructed.

Objects
Objects (also called instances) of a particular class are created by saying

    new ClassName();

This is referred to as instantiation. When an object is instantiated, the constructor corresponding to the signature of the new message is executed.

Statements
Statements are what accomplish most of the action when a program executes. So far these statements have been presented: assignment statement, return statement, block statement, and message statement.

Identifiers
Identifiers are Java names. They start with a letter, and are composed only of letters, digits and underscores.
Methods/Messages
When you send a message to an object, it invokes the method in that class with the same signature. First it performs the parameter linkage, then executes the method body.

Signatures
The signature of a method is it’s access type, return type, name and parameter types.

Parameters (formal/actual)
The parameter in the method declaration is the formal parameter, the one in the message is the actual parameter.

Parameter Linkages
Each actual parameter is evaluated and its value copied to the corresponding formal parameter.

Syntax
The syntax of Java is defined by a set of BNF productions. Any source code not matching this grammar symbol for symbol is deemed to have compiler errors.

Semantics
The semantics of a statement is the action it performs when it is executed. An experienced programmer has internalized the semantics of enough of the constructs of the language that solving routine problems is easy.

Process
Problem Solving Techniques
1. Stepwise refinement
2. Count things
3. Reconceptualize
4. Solve a similar, simpler or simpler problem
5. Add/remove a constraint
6. Draw a picture
7. What are the things? What are their relationships?
8. Delegate to the “dim-witted assistant”

Editing
Editing is when the programmer is inputting or changing source code (classes).

Compilation
Compilation is when the compiler is checking the syntax of a class. Errors at this stage are compile-time errors and are either lexical or grammatical. Lexical errors happen when the compiler does not know what an identifier means; the most common causes are forgetting to declare variables, or typos. Grammatical errors occur when the syntax of the source code does not match the BNF description of the language precisely.
After verifying the syntax of a class, the compiler converts the source code to byte code. Assuming the source code is in a file called Foo.java, the byte code will be put in a file called Foo.class in that same directory.

**Execution**
To execute a program, the byte code is interpreted by the Java Virtual Machine. This is when the work of the programmer comes to fruition. The semantics of the various methods are carried out to achieve some desired result. Errors here are run-time errors, and appear as Exceptions.

**Debugging**
Debugging is the process of removing errors from a program. It is the most time consuming and frustrating aspect of programming. Any nontrivial program has multiple errors. Only novice programmer imagine that one can program without bugs. Like dropping the balls when juggling; it happens. One important skill in programming is learning to write code that is easy to debug.

**Prototyping**
Building simple prototypes and adding functionality as the previous prototype works is perhaps the most important way to make debugging simple; there are simply less places to look for the bugs.

**Assessment Measures**
Exams, quizzes and labs.

**Benchmarks**
Apologies, I still can’t understand what “benchmarks” means; help!