David Keil
Fall 2015

Computer Science II with Java
Framingham State University

Problems for CS II - Java

Multiple-choice study questions on background

0.0a Recall Java or programming concepts*

1. Java compilation and syntax

1. A compiler outputs (a) high-level code; (b) assembler code; (c) binary code; (d) HTML; (e) byte code
2. A compiler inputs (a) high-level code; (b) assembler code; (c) binary code; (d) HTML; (e) byte code
3. The JDK compiler translates from (a) byte code to HTML; (b) HTML to Java; (c) byte code to Java; (d) Java to byte code; (e) Java to machine language
4. Java byte code (a) is compiled to machine language; (b) runs on the Java virtual machine; (c) the microprocessor; (d) is written by programmers in text form; (e) none of these
5. Byte code is like (a) Java; (b) ASCII code; (c) machine code; (d) HTML; (e) none of these
6. The Java virtual machine operates (a) on a server; (b) within the microprocessor; (c) on the hard disk; (d) at the command line and on all browsers; (e) on some browsers only
7. The instructions of the Java virtual machine are (a) within HTML tags; (b) CGI scripts; (c) coded in JavaScript; (d) low-level; (e) high-level
8. Java is (a) easier to understand than machine code; (b) the language of the executable program; (c) what the compiler generates; (d) a formatting language; (e) none of these
9. Microprocessors have on them: (a) a disk; (b) a screen; (c) registers; (d) high-level code; (e) documentation
10. What is fetched in the fetch-execute cycle? (a) instruction; (b) operand address; (c) record; (d) byte; (e) file
11. Machine language is (a) easier to understand than Java; (b) the language of the executable program; (c) the language of the source code; (d) what the compiler starts with; (e) none of these
12. Of the following, which is a high-level programming language? (a) assembler; (b) HTML; (c) machine; (d) Java; (e) none of these
13. A compiler produces (a) high-level code; (b) machine code or byte code; (c) documentation; (d) keyboard input; (e) Java code
14. Compared to assembler language, Java is (a) simpler; (b) about the same level; (c) lower-level; (d) more expensive; (e) higher-level
15. The Java Development Kit includes (a) a compiler; (b) an editor; (c) a spreadsheet; (d) an HTML tool; (e) assembler language.
16. A *project* is (a) a single source file; (b) a set of separately compiled source files that work together; (c) a set of classes; (d) a set of methods
17. Programming guidelines presented in this course include (a) state purpose of every program and component; (b) use minimal variable names; (c) every line of a program should execute; (d) write the most compact code possible; (e) use literals in expressions as much as possible
18. Structured design breaks problem solutions down through (a) iteration; (b) modular decomposition; (c) object-oriented analysis; (d) bottom-up design; (e) none of these
19. Which is *not* a valid Java identifier: (a) *in1*; (b) *my_method*; (c) *3p2*; (d) *t4*
20. When a problem is complex, the complexity can often be conquered in the design stage by (a) brute force; (b) documentation; (c) modular decomposition; (d) input/output; (e) logic gates
21. Modular decomposition of processes is most closely associated with which kind of design? (a) web-site formatting; (b) spreadsheet; (c) database; (d) algorithm; (e) none of these
22. Programming guidelines include (a) minimize white space; (b) format code for clarity; (c) every line of a program should execute; (d) write the most compact code possible; (e) use comments of only up to two words
23. Which determines whether a program compiles: (a) code is easy to understand; (b) code has no syntax errors; (c) code runs fast; (d) code generates no runtime errors; (e) none of these
24. Syntax is (a) documentation; (b) meaning; (c) grammar rules; (d) good-programming guidelines; (e) recursion
25. Java syntax requires that every program have: (a) a *main* method; (b) a variable; (c) input; (d) output; (e) comments

2. Standard data types

1. A data type is (a) a memory location; (b) a value; (c) a specification of the meaning and size of a category of data items; (d) a process; (e) a programming convention
2. Variable identifiers stand for (a) memory locations; (b) data types; (c) operations; (d) specific values; (e) persons.
3. An instance of *Scanner* is (a) a class; (b) a reserved word; (c) a data item; (d) a method; (e) an action
4. *print* is a (a) method; (b) class; (c) package; (d) data item; (e) class
5. Of the following, which is the largest capacity data type? (a) short; (b) *int*; (c) *double*; (d) *double*; (e) *byte*
6. A mantissa, or fraction, component appears in type (a) *double*; (b) *int*; (c) *char*; (d) *byte*; (e) *String*
7. Type casting in Java (a) is automatic; (b) is a way to produce a fractional value when dividing two integers; (c) uses the reserved word *throw*; (d) is discouraged; (e) is a syntax error
8. What is the data type of 3.14? (a) *char*; (b) *int*; (c) *double*; (d) *String*; (e) *void*
9. An item of Java type *char* occupies how many bits? (a) 0; (b) 1; (c) 4; (d) 6; (e) 16
10. Which is a *primitive* data type? (a) *String*; (b) any class; (c) array of *int*; (d) *char*; (e) none of these
11. Which is a *compound* data type? (a) *String*; (b) *byte*; (c) *int*; (d) *char*; (e) none of these
12. What is the data type of '\\'? (a) *char*; (b) *int*; (c) *double*; (d) *String*; (e) *void*
13. *char* is (a) a reserved word; (b) a user-defined identifier; (c) an expression; (d) a method; (e) an operator
14. && is (a) a logical operator; (b) a relational operator; (c) an arithmetic operator; (d) a Boolean expression; (e) a delimiter

0.0b Recall bitwise-operator concepts*

1. Bitwise operations (a) work chiefly with Boolean values; (b) turn 1's and 0's inside data items on and off; (c) are frequently used by application programmers

2. A bit is appropriate to represent (a) a character; (b) a status flag; (c) two check boxes; (d) an integer; (e) a string
3. Bit shifting is used in (a) some arithmetic operations; (b) working with Boolean values; (c) determining the size of sets; (d) string operations; (e) disk storage
4. & is (a) bitwise AND; (b) bitwise OR; (c) bitwise complement; (d) left shift; (e) right shift
5. "|" is (a) bitwise AND; (b) bitwise OR; (c) bitwise complement; (d) left shift; (e) right shift
6. ~ is (a) bitwise AND; (b) bitwise OR; (c) bitwise complement; (d) left shift; (e) right shift
7. << is (a) bitwise AND; (b) bitwise OR; (c) bitwise complement; (d) left shift; (e) right shift
8. >> is (a) bitwise AND; (b) bitwise OR; (c) bitwise complement; (d) left shift; (e) right shift

0.0c Recall loop concepts*

3. Loops and debugging

1. Algorithms (a) are by definition efficient; (b) take finite time; (c) are languages; (d) are a kind of program; (e) none of these
2. The loop is a (a) language; (b) control structure; (c) data structure; (d) program; (e) none of these
3. A trace of an algorithm provides (a) input; (b) a list of errors; (c) snapshots of the values of variables over time; (d) a specification of output; (e) the output
4. Tracing is a tool for (a) coding; (b) debugging; (c) syntax checking; (d) array boundary checking; (e) exception handling
5. A counted loop uses (a) jumps; (b) a sentinel; (c) multi-way branching; (d) floating-point data; (e) an index
6. A computer program or subprogram may compute a mathematical (a) expression; (b) function; (c) proof; (d) theorem; (e) none of these
7. An operator often corresponds to a(n) (a) interface; (b) function; (c) user; (d) program; (e) none of these
8. A mathematical function is a(n) (a) subprogram; (b) algorithm; (c) mapping between sets; (d) two-way relationship between ideas; (e) none of these
9. Counters are used (a) in all loop statements; (b) in all well-written loop statements; (c) normally in *for* statements; (d) only in *for* statements; (e) only in *while* statements
10. An infinite loop (a) is the goal of every programmer; (b) is generally a logic error; (c) is very rare; (d) can be fixed by inserting a semicolon before the loop body; (e) halts
11. The expression in parentheses after *while* is the (a) assignment; (b) sum; (c) exit test; (d) input validator; (e) iterator
12. The *for* statement implements (a) a counted loop; (b) a top-tested loop; (c) a multi-way branch; (d) a recursive method
13. The *while* statement is always (a) counter driven; (b) bottom tested; (c) recursive; (d) top tested; (e) to be avoided
14. A Java statement that starts with *while* is (a) an assignment; (b) a compound statement; (c) a method call; (d) a loop; (e) a branch
15. Which infinite loops are hard to find? (a) those lacking exit conditions; (b) those that always hang; (c) those that sometimes hang; (d) those that always terminate; (e) those that loop for a long time
16. For a loop to exit, the body must ______ a value that is tested by the exit condition (a) initialize; (b) declare; (c) make invariant; (d) change; (e) none of these
17. Debugging may be aided by (a) faster I/O; (b) specification review; (c) tracing loops; (d) user training; (e) none of these
18. A common pitfall with loops is (a) using a Boolean variable; (b) using too many variables; (c) using too few variables; (d) possible exit conditions; (e) impossible exit conditions
19. The main guideline for combatting errors is (a) work late; (b) be very smart; (c) design and code with maintenance in mind; (d) never write new code, always use pre-written code; (e) none of these
20. Testing *cannot* (a) help improve software reliability; (b) prove that a program performs according to specifications; (c) prove that a program crashes on some inputs; (d) locate subtle programming errors
21. What activities in programming loops are especially vulnerable to logic errors? (a) using invalid keywords; (b) forgetting the parentheses around the exit condition; (c) handling the I/O in the loop body; (d) handling boundary conditions; (e) declaring all variables used
22. A hanging computer is evidence of (a) a syntax error; (b) an infinite loop; (c) a bad pointer dereference; (d) an array out-of-bounds error; (e) none of these

3.0a Recall basic class concepts*

4. Classes and objects

1. A class is a (a) data type; (b) data item; (c) expression; (d) process; (e) method
2. Reasons to define methods include (a) maximizing the number of identifiers and reducing code repetition; (b) modularity and maximizing the number of identifiers; (c) modularity and reducing code repetition; (d) reducing code repetition and increasing the size of programs; (e) modularity and increasing program size
3. A data variable may have several data attributes and a set of characteristic behaviors is (a) a type; (b) an integer; (c) an object; (d) a class; (e) a control structure
4. The *state* of an object is (a) the process that created it; (b) the set of values of its data attributes; (c) the eventual destiny of the object; (d) the operations associated with it; (e) none of these
5. What are defined by common structure and behavior? (a) functions; (b) processes; (c) classes; (d) member items; (e) methods
6. Data abstraction is (a) the declaration of variables; (b) the creation of subprograms; (c) the definition of new data types; (d) the processing of input; (e) definition of methods
7. Classes model (a) state only; (b) behavior only; (c) transactions only; (d) state and behavior; (e) all three
8. An object implements the concept of a (a) string; (b) integer; (c) database record; (d) database table; (e) database query
9. Objects are declared using (a) *int*; (b) *while*; (c) *if*; (d) *new*; (e) *switch*
10. A class is an abstract specification for (a) a program; (b) an algorithm; (c) arrays; (d) objects; (e) none of these
11. Members of classes include (a) comments and specifications; (b) algorithms and control structures; (c) operations and properties; (d) numbers and specifications; (e) none of these

Problems for background on Java programming

0.1a Explain the Java virtual machine*

1. Describe the Java virtual machine.
2. Distinguish a *virtual machine* from a processor.
3. Describe *byte code* and its role in Java software development.
4. Compare *Java code, machine code,* and *byte code.*
5. Describe a program that will run on a Java virtual machine, and how it is generated.
6. How does *byte code* contribute to Internet computing?
7. Compare the *compilation* of Java code to a familiar Java-based process of *interpretation*.
8. What is a *virtual machine,* and why is it used to run Java programs?

0.1b Explain the fetch-execute cycle*

1. Name and describe the loop that a microprocessor is running at all times.
2. Describe the *fetch/execute cycle*, giving specific locations of all relevant data, with reference to the model computer discussed.
3. What happens *at the machine level* when a program executes?
4. What is the role of *random-access memory* in the execution of a machine-language program?
5. What is the role of the *program counter* in the execution of a machine-language program?
6. Describe the role of the *instruction register* in the execution of a machine-language program.
7. Describe the steps of a *load – add – store* computation.
8. Describe all items that an assembler-language statement may fetch.

0.1c Describe Java syntax*

1. What syntax elements are in a Java *compound statement*?
2. Give an example of a *recursive* Java grammar rule.
3. Describe four kinds of Java statements, including two that may *contain* statements.
4. Describe the syntax of Java branch and loop statements.
5. Give an instance of a statement within a statement.
6. What is the syntax of a Java assignment statement?
7. What is the syntax of the Java *if-else* statement?
8. How does a Java *variable* differ from a *literal*? Give examples.
9. Describe grammar rules for Java *Boolean* expressions.
10. Describe grammar rules for using arithmetic operators in Java.
11. Give an instance of a Java *arithmetic* expression within a *relational* expression.
12. Explain how three kinds of *operators* are used in Java to form expressions.
13. Give an instance of a Java *relational* expression within a *logical* expression.
14. Distinguish *Boolean* from *relational* operators in Java.
15. Distinguish *arithmetic* from *relational* operators in Java.

0.1d Identify the steps in system development*

1. Describe steps that follow *analysis* in the systems development life cycle.
2. Distinguish problem specification, system design, and program coding.
3. Describe the systems development life cycle.
4. Describe the steps that precede and follow *design*, in the system development life cycle.
5. In the software development process, describe steps that are recommended before coding a program; after coding.
6. What are some of the main aspects of a system that someone specifying a system must consider?
7. Distinguish program coding, system design, and problem specification, giving also their correct ordering.
8. Describe the *maintenance* phase in system development.
9. What happens after the design step in system development?
10. What is program maintenance?

0.1e Explain code documentation*

1. Name and describe two different forms of documentation that you can write in your source code to make the program easier to understand.
2. Give two reasons for using named constants in a program.
3. Name two steps the programmer may take in a Java program to make sure his or her intention is clear.
4. Explain what is recommended that every program have that the compiler will ignore.
5. How are the intentions of program code made explicit?
6. Why does code indentation matter?
7. What is documentation that every program is recommended to have at the top?
8. How can naming variables aid program documentation?
9. What is the syntax and purpose of comments?
10. When should a comment appear in program code?

0.2a Describe standard Java types and classes*

1. How are the newline, tab, and null-character constants expressed in Java?
2. Show how you can convert an integer to a character value and a character to an integer. Name the operation.
3. What are an advantage and a disadvantage of Java's automatic type conversion?
4. Write expressions that convert:
   a. "9" to *int*
   b. 30 to *double*
   c. "2.8" to *double*
   d. 0.9 to *String*
   e. 83 to *String*
5. What is the data type and memory allocation, in bits and bytes, of a single component of a *String* object?
6. Explain the output of this code:
```
int n = 25.9;
out.println(n + 1);
```
7. What is the relationship between *data types* and *variables*? What do variables have that data types don't?
8. Write an expression that type casts the *double* variable *amt* to type *int*.
9. Give an example of *overflow*. How is it avoided?
10. Name and explain the situation when a value assigned to a numeric variable is too large.

0.2b Correct a type error*

Debug this code, explaining the error or risk:

1.
```
int quantity = in.nextInt();
double price = in.nextInt();
out.println("Amount due: "  +
  quantity * price);
```
2.
```
string x = “Hello”;
```
3.
```
int quantity, price = 5;
out.println(quantity * price);
```
4.
```
String quantity = 4, price;
price = in.Next();
out.println("Amount due: " +
 quantity * price);
```
5.
```
int quantity, price = 5.95;
out.println(quantity * price);
```
6.
```
double n = in.nextDouble();
int q = 2 * n;
```

0.2c Use logical operators*

Write Java code that

1. accepts a distance in miles and displays “OK” if it is not higher than 100 or larger than 300.
2. displays “in order” if the values, *a*, *b*, and *c* are in ascending order.
3. has the value *true* if *height* is in the range of 60 to 72 inclusive, otherwise *false*.
4. tests whether the following is true: either *height* is greater than 72 inches or *age* is not less than 30.
5. displays an error message if input *x* is outside the boundaries (0.5,1.5).
6. classifies a person on the basis of input height and weight. Use the following classification scheme:

Height	*Weight*	*Classification*
> 72 in.	> 190 lb.	Tall and heavy
> 72 in.	≤ 190 lb.	Tall and light
≤ 72 in.	> 170 lb.	Short, heavy
≤ 72 in.	≤ 170 lb.	Short, light

7. tells whether two numeric inputs, *a* and *b*, are at least five apart but not more than 10 apart.

0.2d Use bitwise operators

Showing your work, evaluate these Java expressions:

1. (a) 12 >> 2 (b) 12 << 2
2. (a) 8 << 1 (b) 3 & 2
3. (a) 24 >>1 (b) 5 | 4
4. (a) 7 & 3 (b) 4 ^ 6
5. (a) 5 | 3 (b) a << b
6. (a) 6 & 2 (b) c >> d
7. (a) 6 | 3 (b) In a 4-bit register, ~ 3
8. 5 >> 1 (b) 5 ^ 7
9. Write a Java statement that uses a mask to (a) set the rightmost bit of the integer variable *n* to 1
10. Write a Java statement that uses a mask to clear the second-rightmost bit of the integer variable *n* to 0.

0.3b Trace a branch or loop**

(1-4) Trace the following pseudocode for inputs below.

```
input a, b, c
 $y \leftarrow a$
 if $b < y$
       $y \leftarrow b$
 otherwise
       if $c < y$
             $y \leftarrow c$
 display $y$
```

1. a = 1, b = 2, c = 3
2. a = 1, b = 2, c = 1
3. a = 3, b = 2, c = 1
4. a = 2, b = 3, c = 1

(5-10) Trace the pseudocode below,

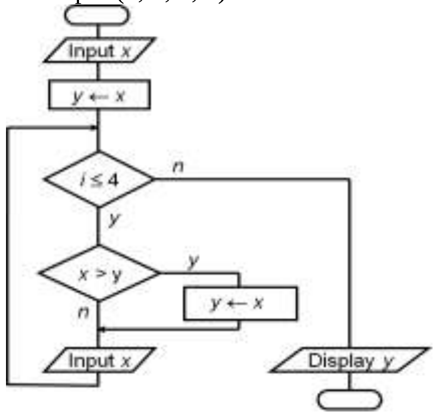
```
 if $a$ then
       $y \leftarrow 1$
else if $b$ then
       $y \leftarrow 2$
if $c$ then
       $y \leftarrow 3$
```

for (a, b, c) =

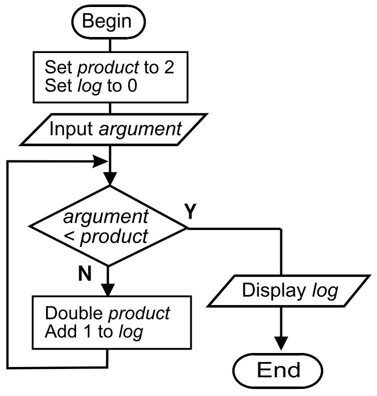
5. (true, false, true)
6. (false, false, true)
7. (true, true, false)
8. (false, true, false)
9. (true, true, true)
10. (false, false, false)

Choosing small sample input values, write a trace table for each of the following instances of pseudocode. A trace table has one column for each variable, plus a column for output, and one row for each iteration of a loop.

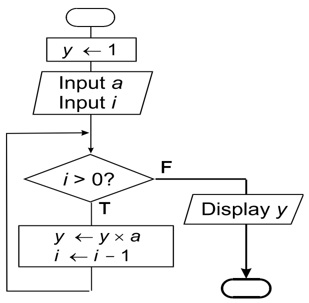
1.
```
$count \leftarrow 1$
$sum \leftarrow 0$
while $count < 5$
     $sum \leftarrow sum + count$
     $count \leftarrow count + 1$
display $sum$
```
2.
```
input $a$, $b$
$y \leftarrow 0$
while $i > a$
     $a \leftarrow a - b$
     $y \leftarrow y + 1$
     $i \leftarrow i + 1$
Display $y$
```
3.
```
input $x$, $m$, $k$
$y \leftarrow k$
while $m \leq 0$
     $y \leftarrow y + x$
     $m \leftarrow m - 1$
output $y$
```
4.
```
input $x$
$i \leftarrow 1$
$y \leftarrow x$
while $i \leq 4$
   if $x > y$
        $y \leftarrow x$
   Input $x$
   $i \leftarrow i + 1$
Display $y$
```
5.
```
input $v$
$i \leftarrow 1$
$y \leftarrow true$
while $i \leq 4$
   input $x$
   if $x < v$
        $y \leftarrow false$
   $i \leftarrow i + 1$
Display $y$
```
6. Show a trace table of this flowchart, on input (1, 2, 5, 3).



7. Show a trace table of this flowchart, for input 15.



8. Write a trace table for this flowchart, for inputs (2, 3).



0.3b Solve a numeric loop problem**

Write Java code, flowchart, or pseudocode to solve the following problems.

1. Input integers until the current input is less than the previous input. Display the largest input value.
2. Input values *first* and *n*, and display each of the *n* integers starting at *first*.
3. Accept *x*, and display the product of all the whole numbers from 1 to *x*.
4. Loop to accept input of exactly three pairs of integers (*A*,*B*) and compute and display the value of *A* – *B* for each input pair.
5. Compute and display the product of two non-negative input integers, performing the multiplication as repeated addition.
6. Input fifty scores and display the highest.
7. Prompt for an input value and outputs the integer part of its base-2 logarithm.
8. Accept a number *x* and displays the sum of all the integers from 1 to *x*
9. Input dollar amount *x* and output the value of an investment of *x* at 4% per year for three years.
10. Write a program that prompts for a loan amount for a loan at 8.5%, and displays the amount to be repaid after 30 years, including principal and interest. Show results accurate to the cent. Use named constants when appropriate. On input of a zero or negative amount, display an error message.
11. Repeatedly prompt for integers until the user enters 0; display the largest.
12. Input a depreciation rate, *d*, and output the value of a $15,000 car, after three years, depreciating at rate *d*.
13. Display the numbers from 100 to 200 that are divisible by 3.
14. Prompt for an integer value and, using integer division and modulo operations, output its individual digits, on separate lines. (*Note:* this is needed for the first assignment of CSCI 465.)
15. Input fifty scores and display the average.
16. Write a program that displays all the whole numbers from 1 to 1000 that are divisible by 2.
17. Prompt for *a* and *b*; loop to output a^b.
18. Input amount *x* and output the value of an investment of *x* at 10% per year for three years.

0.3c Solve a loop problem with strings**†

Input a line of text as a string, using 'nextLine()', and using loops, output the following values. (For examples of Java string manipulation, see 'srchspc.java' in handout, "String manipulation and search.")

1. the location of the last period in the string
2. the second word only
3. a count of the number of vowels found
4. "has double" if it contains any pairs of identical characters side by side.
5. the same string, with the character '*' in brackets within the string, wherever '*' is found
6. the same string backwards
7. the same string, assuming it is a name, with last name first, then comma, first, and middle
8. the same string, with any occurrence of the word "not" capitalized.
9. the number of occurrences of '1' or '2'.

0.3d Debug a defective loop**†

The code below contains errors. Test the code, fix the errors, explain the errors and fixes, and print test results.

1. Explain and correct the error in the following code, which is intended to compute 2^{10}. Use a loop, not *Math.pow*.
```
int power;
while (power < 10)
{
        power = power * 2;
}
out.println(power);
```
2. Debug this code, which compiles to an error message, "Multiple declarations of *m*":
```
for (int m = 0, m < 10, ++m)
    System.out.println(m);
```
3. Debug this code, which is intended to display the numbers from 1 to 5:
```
int num = 5, i;
for (i=0, i < num, num++)
    System.out.println(i);
```
4. The code below is intended to display the numbers from 1 to 10. (a) Will it compile? (b) Does it accomplish its objectives? If not, correct it.
```
int a = 1;
while (a < 10);
    a = a + 1;
out.println( a );
```
5. //intended to calculate (*x*!)
```
int y = 0;
for (int i = x; i >= 0; i-0-)
    y = y * i;
```
6. // intended to find the sum of all positive integers less than *n*
```
out.println( "Number? ");
int n = in.nextInt();
if (n > 0)
{
     int sum = 0;
     int i = n – 1;
     while (i > 0)
     {
          sum = sum + i;
          out.println( "sum, up to
           " + n + " is " + sum);
     }
}
```
7. // *total* should be sum of inputs
```
int count, x, total;
x = in.nextInt;
while (x > 0)
{
    out.print("x=" + x +
         " total=" + total);
    x = in.nextInt();
    total += x;
}
```
8. // should display *x* factorial:
```
Scanner in =
    new Scanner(System.in);
int y = 1, x = in.nextInt();
while (y < x)
    y = y * x;
System.out.println("y = " + a);
```
9. // should print the value 11:
```
int a = 1;
while (a < 10)
    a = a + 1;
System.out.println("a= " + a);
```
10.
```
// should count values in file:
int n = 0;
while (fin.hasNext())
   n++;
System.out.println("n= "+ n);
```

0.4a Write a UML class diagram*

Write a UML class diagram, with name, attributes, and operations, for a class design or specification to support the digital representation of

1. university course sections
2. students
3. college courses
4. video DVDs
5. songs
6. items of clothing for sale
7. job positions
8. corporations
9. stock offerings
10. exercise problems in this course
11. professors

0.4b Describe memory allocation for objects**

1. Explain memory allocation for *primitive type* instances and for instances of *classes*.Write Java statements to allocate instances of the type *int* and the class *String*, stating where each is allocated.
2. Contrast the ways in which a *double* variable and a *Scanner* object are allocated.
3. Explain where in memory *primitive type* instances reside, and instances of *classes*.
4. Contrast what happens at run time, in memory, when a variable is declared as in (a) *int a*; (b) *Point p = new Point;*

0.5 Explain precalculus concepts*

1. What is a mathematical *function*?
2. What is the equation form of a *linear* function?
3. What is the equation form of a *quadratic* function?
4. Give examples of a *quadratic* function, a *linear* function, and an *exponential* function.
5. What is the equation form of a *polynomial* function?
6. What is a *sequence*, as discussed in precalculus?
7. What is the equation form of an *exponential* function?
8. Explain the *factorial* function.
9. Distinguish *arithmetic* sequences from *geometric* sequences.
10. Compare the graphs of *logarithmic* functions, *linear* function, and *exponential* functions.

Multiple-choice questions on Topic 1: Methods

1.0a Recall basic Java method concepts*

1. Procedural abstraction and Java methods

1. Creating named subprograms is called (a) iteration; (b) data abstraction; (c) encapsulation; (d) procedural abstraction; (e) verification
2. Reasons to define methods include (a) maximizing the number of identifiers and reducing code repetition; (b) modularity and maximizing the number of identifiers; (c) modularity and reducing code repetition; (d) reducing code repetition and increasing the size of programs; (e) modularity and increasing program size
3. A way to make a program more modular is to (a) document variables with comments; (b) print clear output; (c) use file input; (d) write method definitions; (e) make *main* longer
4. The top-down approach (a) breaks down a problem and solves it step by step; (b) begins with ready-made components and puts them together; (c) focuses on minimizing the number of methods; (d) originated with object-oriented programming
5. A method definition (a) is a method call; (b) has a header and a body; (c) terminates with a semicolon; (d) may be used as a program statement; (e) none of these
6. A Java expression that ends with () is always (a) a loop (b) an arithmetic expression; (c) a method call; (d) a declaration; (e) an assignment
7. Every Java method is (a) a statement; (b) an expression ; (c) a member of a class; (d) a class; (e) a type
8. *main* is (a) a variable; (b) a class; (c) a public method; (d) a private method; (e) an expression
9. procedural abstraction enables (a) rapid execution; (b) proof of correctness; (c) modularity and reuse of code; (d) code complexity; (e) correct syntax
10. All Java methods are (a) fields; (b) members of classes; (c) data items; (d) global; (e) public
11. A *static* method (a) doesn't move; (b) doesn't change; (c) is always called by an object; (d) can't be called by an object; (e) has a constant value

2. Variables, parameters, and return values

1. Variables are located (a) within the method's machine code; (b) in secondary storage; (c) on the stack; (d) in the microprocessor; (e) all of these
2. In a method call we might find (a) a method definition; (b) a method header; (c) a formal parameter; (d) an actual parameter; (e) none of these
3. A value may be passed out of a method to the calling statement with a (a) value parameter; (b) return value; (c) variable parameter; (d) *goto* statement; (e) *new* operator
4. With an *int* parameter, what is passed to the method? (a) the address; (b) the value of an expression; (c) the full text of the expression; (d) nothing; (e) a request for information
5. A method whose name is used as an expression in a program should have (a) a value parameter; (b) a reference parameter; (c) a return value; (d) no parameters; (e) a variable declaration
6. Parameter values are stored (a) with the calling method's machine code; (b) with the called method's machine code; (c) in the calling method's activation record on the stack; (d) in the called method's activation record on the stack; (e) in a program's variable memory
7. A method that returns a value (a) must do so with a parameter; (b) should specify the data type of that value in the method header; (c) does so with an assignment statement; (d) does so automatically; (e) generates an exception
8. Activation records are created (a) at compile time; (b) when a program starts; (c) when a method executes; (d) when a program terminates; (e) when source code goes to QA
9. Parameters act line (a) classes; (b) methods; (c) local variables; (d) sensors; (e) actuators
10. Cohesion is associated with (a) syntax errors; (b) the principle that a method should have a single clear responsibility; (c) runtime errors; (d) loops; (e) branches
11. Weak coupling means (a) shaky compilation; (b) termination; (c) methods' independence of each other; (d) poor documentation; (e) other than binary relations

3. Documenting and testing methods

1. A *stub* is a (a) variable; (b) expression; (c) method; (d) class; (e) comment
2. A stub (a) performs a complex operation; (b) calls methods; (c) has a loop; (d) reports that it has been called; (e) contains processed input data

4. Recursive methods

1. Recursion is (a) a way to implement a loop; (b) a kind of file input; (c) a nested loop statement; (d) used in all loops; (e) none of these
2. If a method calls itself, it (a) is correct; (b) is incorrect; (c) crashes; (d) loops forever; (e) is recursive
3. A recurrence defines (a) a set of natural numbers; (b) a logical formula; (c) a computable function; (d) an undecidable problem; (e) none of these
4. A recursive method definition (a) uses a *while* loop; (b) lists all possibilities; (c) contains a call to the method itself; (d) is impossible; (e) is inefficient
5. Recursion uses the (a) cache; (b) hard disk; (c) accumulator; (d) stack; (e) heap
6. A recursive method call creates (a) a new method; (b) a new activation record; (c) a crash; (d) an exception; (e) a class
7. Part of a recursive definition of the factorial function returns (a) 0; (b) 2^n; (c) n^2; (d) *factorial*(1); (e) *factorial*(n – 1)
8. Recursion stores local variables (a) on the stack; (b) on disk; (c) as parameters; (d) on the heap; (e) globally

5. Java file input/output

1. Attempting to open a nonexistent file to read always (a) crashes a program; (b) generates an exception; (c) produces a syntax error; (d) produces false output; (e) produces correct output
2. References store (a) addresses; (b) numeric data; (c) methods; (d) classes; (e) string data

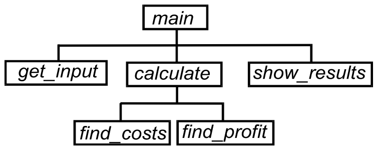
3. *FileReader* is a(n) (a) standard method; (b) scanner class; (c) user-defined object; (d) package; (e) program
4. Files are _____ objects (a) *String*; (b) stream; (c) *int*; (d) *char*; (e) none of these
5. The most appropriate statement for reading data from a file is (a) *if*; (b) *while*; (c) *do*; (d) *switch*; (e) assignment
6. Streams are of length (a) 256; (b) fixed by user; (c) indeterminate; (d) finite; (e) none of these
7. If a Java program attempts to read a file that contains no more data, it (a) crashes; (b) generates an exception; (c) produces a syntax error; (d) produces false output; (e) produces correct output
8. If a program attempts to open a file to write, and the file already exists, the program (a) crashes; (b) generates an exception; (c) produces a syntax error; (d) produces false output; (e) overwrites the existing file
9. *nextInt()* is used (a) with keyboard input only; (b) with screen output only; (c) with keyboard or file input; (d) with screen or file output; (e) with file input only
10. Any sequence of characters moving to or from a device is a (a) string; (b) loop; (c) type; (d) class; (e) stream
11. A stream is (a) a sequence of characters moving to or from a device; (b) a method; (c) a class; (d) a string; (e) a device
12. *PrintWriter* is a(n) (a) method; (b) file stream class; (c) device; (d) program; (e) control structure
13. The data type of *hasNextInt()* is (a) *int*; (b) *double*; (c) *char*; (d) *boolean*; (e) *file*

Problems for topic 1: Methods

Topic objective:
Define and test Java methods and classes with object-oriented features

1.1a Explain procedural abstraction**

1. Describe the modular approach to program development and give a reason for using it.
2. Name the type of diagram shown below and describe its role in program design. [*Show module hierarchy diagram.*]
3. In the module hierarchy diagram below, describe which methods call other ones, specifying the others as well.



4. Draw a module-hierarchy chart for the following skeleton program.
```
public static void main()
{ fa(); fd(); }
public static void fa() { fb(); fc(); }
public static void fb() { }
public static void fc() { }
public static void fd() { }
```
5. Write a module hierarchy chart for a program that emulates a hand calculator.
6. What is *procedural abstraction,* and how is it implemented in Java?
7. What are the name and purpose of the kind of abstraction used when defining methods?

1.1b Define and test a Java method**†

1. Write a method that takes a numeric parameter, x, and returns the value $(2x^2 - x + 1)$.
2. Write a Java program that defines and calls a method whose name is your name, and whose code displays your name.

(3-10) Convert the method *main* of the program that you wrote for objective 0.3b into an appropriately named static method that is called from *main*. Verify that test results are the same.

1.2a Explain method signatures and scope*

1. Describe a *method signature*.
2. How does the Java language support communication of data between methods?
3. Write the *signature* of a method that takes an integer as a parameter and returns a floating-point value.
4. Explain *method overloading*.
5. Describe the meaning of the *scope* of a variable, according to the rules of Java.
6. How are the *return value* and *parameters* expressed in a Java method definition?
7. In what parts of a program is it possible to use a certain variable that has been declared?
8. Do the statements in a method always execute at least once when the program containing the method definition executes? Justify your answer.
9. Write an appropriate *signature* (header) for a method that takes two integers as parameters, returns no value, and draws a rectangle. You need not spell out how the method does this.
10. What would be a good *signature* for a method that takes parameters and that displays the interest amount on a loan, given principal, interest, and term of loan?
11. Write the *signature* for a method that accepts two *double* parameters and returns their *integer* sum.

1.2b Write a method with parameters and return values**

Write a Java definition for a method that does the following, without calling any other method:

1. returns the absolute value of the difference between two integer parameters.
2. takes two floating-point parameters, a and b, and returns the *integer* ratio of their sum to their difference.
3. takes a parameter that is a Celsius (Centigrade) temperature, that returns the corresponding Fahrenheit temperature. Fahrenheit is 32 degrees above 9/5 of Celsius.
4. draws a *rectangle,* composed of X's like that below, accepting parameters for both the height and the width of the figure. *Sample I/O:*
```
Size of square: 3
XXX
XXX
XXX
```
5. accepts the (x,y) positions of two opposite corners of a rectangle as parameters and returns the *area* of the rectangle. (This is the product of the difference between the horizontal positions and the difference between the vertical positions.)
6. accepts an integer parameter, *length*, and displays a line of that many hyphens.
7. accepts three integer values as parameters and returns the smallest one.
8. takes two integer parameters, a and b, and returns the sum of all integers from a to b, if $a \le b$; otherwise 0.
9. takes a parameter that is a measurement in feet and returns the equivalent measurement in meters. A meter is 39.37 inches.

1.2c Debug a method†

Debug the following method, describing the error:

1.
```
public static void main()
{
  int area = 3;
  System.out.println("A="+twice(area));
}
void twice (int n)
{
  n = n * 2;
}
```
2.
```
int what()
{
  int m, n = 4;
  return 2 * m;
}
```

3.
```
void draw_line(int start, length)
{
  for (int i=0; i < start; i++)
    print(' ');
  for (int i=0; i < length; i++)
    print('-');
}
```
4.
```
public static rate(int score)
{
  if (score >= 1 && score < 2)
    return "weak";
  else if (score >= 2 && score < 3)
    return "OK";
  else if (score >= 3 && score < 4)
    return "good";
  else if (score >= 4)
    return "excellent";
}
```
5.
```
int doubtful(int m)
{
  int m = 0;
  return 2 * m;
}
```
6.
```
void peculiar(int m)
{
  return 2 * m;
}
```
7.
```
void get_amt()
{
  out.println( "Sales amt: ");
  double amt;
  amt = in.nextInt();
  get_amt();
}
```

1.3 Explain method documentation and testing*

1. Explain *Javadocs.*
2. Explain *stubs*, with an example.
3. Explain *drivers*, with an example.
4. Explain documentation of methods.
5. Explain testing of methods.
6. Explain *@param* when found in a Java comment.
7. Explain *@return* when found in a Java comment.

1.4 Derive a recursive method from a loop

Write a recursive method that, for parameter n, returns:

1. the sum of the natural numbers from 1 to *n*
2. *n* factorial
3. 2^n
4. the product of all the odd numbers from 1 to *n*
5. the sum of all the even numbers from 0 to *n*

Write a recursive method that, for parameters a and b, returns:

6. a^b
7. the product of all the natural numbers from *a* to *b*
8. the sum of all the natural numbers from *a* to *b*

1.5a Explain streams and sequential file I/O*

1. What is a *stream* and how are streams used in Java for disk access?
2. Describe how to read data from a file, using Java.
3. What are the data types returned by the *next()* and *nextLine()* methods? What differentiates these methods?
4. What is a *file*?
5. Explain how Java supports *file* data storage and retrieval?
6. What does "opening a file" mean?
7. Describe two common risks in Java file I/O.
8. What are some sources and destinations for streams?
9. What is *Scanner*, and how is it used in file I/O?
10. How are files opened and closed in Java?
11. What is a risk in opening a file for output?

1.5b Read a file using a loop**†

1. Write a program that reads a text file composed of lines and displays it on the screen.

(2-12) Use a loop to read a series of numeric inputs from a text file, until end of file, and outputs the following, with echo of file input and with summary of results. (For sample file-reading code, see handout and Java file read_file.java.)

2. "ascending" if they are in ascending order, otherwise "not ascending"
3. their average;
4. the largest
5. the smallest
6. "dups" if any consecutive duplicates exist, otherwise "no dups"
7. "all same" if all are the same, otherwise "not all same"
8. their maximum variance; i.e., the largest distance of any number from the average value (to do this you will have to read the file twice)
9. the second largest value
10. the winner in a two-way yes/no vote, with votes consisting of all the values in the file, assuming that a zero means "no" and any other value means "yes"
11. the most popular contestant, if all the inputs are votes for contestants 1, 2, or 3
12. "all pairs" if the first and second input, the third and fourth, and so forth are all pairwise matched; otherwise "not all pairs".

Multiple-choice questions on Topic 2: Arrays

2.0a Recall basic array concepts*

1. Defining and populating arrays in Java

1. An array is a(n) (a) compound data item; (b) simple data type; (c) package; (d) dynamic data structure; (e) control structure
2. The components of an array are called (a) members; (b) items; (c) elements; (d) enumerations; (e) return values
3. An indexed sequence of storage locations for data items all of the same type is a(n) (a) object; (b) class; (c) array; (d) list; (e) none of these
4. A subscript is (a) an address; (b) an index to the location of an array element; (c) the size of an array; (d) a decrementing operation; (e) a special graphics character
5. If an array is declared *int x = new int[8]*, then *x*[8] is (a) the seventh element; (b) the eighth element; (c) out of bounds; (d) the address of the array itself; (e) zero
6. Array elements are accessed by (a) key; (b) subscript; (c) size; (d) value; (e) none of these
7. To declare an array in Java requires use of the operator (a) +; (b) =; (c) <<; (d) *new*; (e) none of these
8. Curly braces may be used (a) to reference an array element; (b) to assign a value in an assignment statement; (c) to initialize an array; (d) to select an array element; (e) within expressions
9. The expression *A*[*i*] is analogous to (a) *A* + *i*; (b) *s.charAt(i)*; (c) *in.nextChar()*; (d) *for*(*int i* = 0; *i* < *A*; *i*++); (e) *sum*(*A* .. *i*)
10. *int [ ] A = new int[10];* (a) declares an integer; (b) allocates memory for an array; (c) declares a string; (d) contains a syntax error; (e) initializes an integer variable to 10
11. Which creates an array? (a) *int A = { }*; (b) *int[ ] A*; (c) *int[] A = new int*; (d) *int [ ] A = new int[10]*; (e) *String A*;
12. To initialize an array, use (a) a loop; (b) a list in curly braces; (c) *nextInt()*; (d) *init*; (e) *int [ ] A*
13. *int[]* is (a) an integer variable; (b) an array variable; (c) an array type; (d) a numeric type; (e) a syntax error

2. Array operations and boundary errors

1. *int[]a = new int[3]* creates (a) an integer; (b) a string; (c) an array with subscripts 1 to 3; (d) an array with subscripts 0 to 2; (e) an array of undefined size
2. An array boundary violation in Java triggers (a) program termination; (b) an exception; (c) a logic error risking false results; (d) a syntax error; (e) evaluation to zero
3. To copy the contents of an array *A* to another array *B*, write (a) *A* = *B*; (b) *B* = *A*; (c) *A = B.clone()*; (d) *A = A.clone()*; (e) *B = A.clone()*
4. When an array name is used as a parameter, the data copied is (a) all the values of its elements; (b) the memory address of the array; (c) the value of its first element; (d) all of the above; (e) random
5. *x[6]* may be a method (a) header; (b) call; (c) definition; (d) parameter; (e) name
6. *part[5][7]* is: (a) an element of a one-dimensional array; (b) an element of a two-dimensional array; (c) an array declaration; (d) a method call; (e) a violation of Java syntax
7. A two-dimensional array is a (a) vector; (b) class; (c) object; (d) matrix; (e) set
8. After the statement *int[] A = new int[5], A[–1]* triggers (a) a syntax error; (b) –1; (c) a boundary-error exception; (d) a crash; (e) an infinite loop
9. What aspect of programming arrays is especially vulnerable to logic errors? (a) declaring the array; (b) dimensioning the array with a valid number; (c) ensuring that all subscripts are within bounds; (d) remembering to use left and right brackets; (e) spelling the array name correctly
10. After the statement *int[] A = new int[5], A[5]* triggers (a) a syntax error; (b) –1; (c) a boundary-error exception; (d) a crash; (e) an infinite loop
11. To copy contents of an array *A* to array variable *B*, use (a) *int[ ] B = A*; (b) *int[ ] B = (int[])A.clone()*; (c) *int B = A*; (d) *int B[] = A[]*; (e) *int [ ] B = new int[A]*
12. An array variable is (a) a type; (b) a primitive data item; (c) a reference; (d) a method; (e) an oxymoron
13. How many array element accesses would it take to verify that an array of size *n* is sorted? (a) exactly one; (b) not more than log *n*; (c) exactly *n*; (d) at least *n*; (e) not more than *n*
14. How many array element accesses would it take to verify that an array of size *n* contains the value *k*? (a) exactly one; (b) not more than log *n*; (c) exactly *n*; (d) at least *n*; (e) not more than *n*

2.0b Recall advanced array concepts

3. Searching arrays

1. What is the fastest way to search for a value in an unsorted array of numbers? (a) calculate hash value; (b) scan from beginning to end until value is found; (c) sort array and perform binary search; (d) perform binary search; (e) no fastest way exists.
2. The speediest way to locate the number 243, for example, in an array of random integers is a (a) linear search; (b) binary search; (c) Bubble sort; (d) Quick sort; (e) none of these
3. A merge algorithm (a) takes longer than Bubble sort; (b) performs a search; (c) requires two or more sorted arrays; (d) all of these; (e) none of these
4. Each step of the binary-search algorithm (a) reduces the size of the sub-array to be searched by about half; (b) finds the search key; (c) reports failure; (d) moves one array element; (e) compares two array elements
5. An appropriate precondition for the binary search is that (a) the array contains a particular value; (b) the array is large; (c) the array is in ascending order; (d) the array's capacity is as declared; (e) the array is small
6. How many array element accesses would it take to verify that an array of size *n* does *not* contain the value *k*? (a) exactly one; (b) more than n^2; (c) exactly *k*; (d) more than *n*; (e) not more than *n*

4. Nested loops and sorting algorithms

1. Which algorithm finds adjacent elements out of order and swaps them, until none are found out of order? (a) insertion sort; (b) bubble sort; (c) selection sort; (d) Shell sort; (e) Quicksort
2. An appropriate postcondition for a sorting algorithm is that (a) the array contains a particular value; (b) the array is larger than it was before; (c) the array is in ascending order; (d) the array's capacity is as declared; (e) the array is smaller than before
3. All sorts we discussed contain (a) I/O; (b) nested loops; (c) numeric operations; (d) *switch* statements; (e) recursive calls
4. Finding duplicates in an array requires (a) searching; (b) traversing; (c) sorting; (d) merging; (e) nested loops
5. An example of a nested-loop problem is (a) a search; (b) finding duplicates; (c) a merge; (d) a traversal; (e) a branch
6. An example of a nested-loop problem is (a) a search; (b) a sort; (c) a merge; (d) a traversal; (e) a branch
7. Two sorts we discussed are (a) Bubble and insertion; (b) insertion and binary; (c) selection and linear; (d) linear and binary; (e) merge and linear

Problems for Topic 2: Arrays and loop design

Topic objective:
Define and safely manipulate arrays, designing nested loops and applying search and sorting algorithms

2.1 Describe Java arrays**

1. In one statement, create an array of integers and initialize it to the values 1, 2, and 3.
2. Declare appropriate Java data items to store in memory a set of up to 100 *double* values that your program reads from a disk file.
3. What is a Java *array*, and how is it created?
4. Describe what the following code does:
```
int A[] = new int[];
```
5. How can a program obtain the size of a Java array?
6. What are the possible subscripts of a Java array? Explain.
7. What does the following code do? What sorts of operations would be possible after it executes?
```
    int i = 0;
    while (in.hasNext())
       A[i++] = in.nextInt();
```

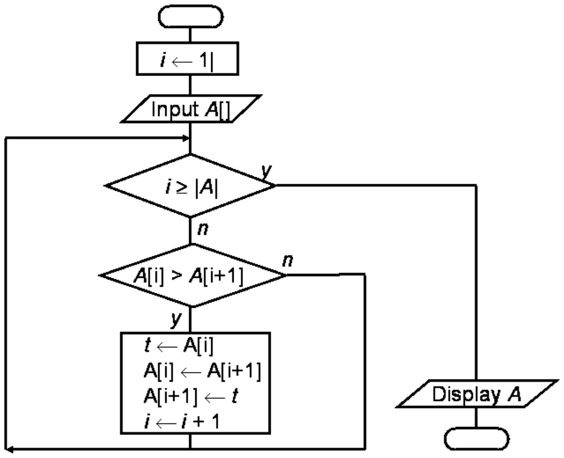
2.2a Traverse an array**

(1-16) Write a program that reads from a file into an array of integers, and displays the following, checking array boundaries as necessary:

1. the average of the maximum and minimum values
2. the average *variance* from the average value
3. the length of the longest run of consecutive elements with the same value
4. the second-highest value
5. "Ascending," if the elements are all in ascending order
6. the number of values that are greater than the average
7. the number of values that match the first element
8. the average of all the elements that are less than the last
9. the number of values that are less than the average
10. the greatest distance between members of any consecutive pair
11. the average *variance* (the distance between a single value and the average value)
12. the largest *two* elements
13. the sum of the elements of the array

(13-18) Write a Java method that accepts an array of integers as a parameter, and an integer x (in certain cases), and returns

14. the array with *x* inserted at the beginning of the array
15. the index of the rightmost occurrence of *x*; –1 if *x* not found.
16. the index of the first occurrence of *x*; –1 if *x* not found
17. the number of occurrences of *x*
18. *true* iff the array is in ascending order
19. *true* iff all elements of the array are the same
20. *true* iff any pair of consecutive elements match
21. *true* iff the sum of the first half of the array is greater than *x*.
22. The original array, with an integer, *x*, inserted at the *beginning* of the array.
23. Convert the flowchart below to Java, test it to tell what it does, and modify it to operate on a *minimal* rather than maximal value.



2.2b Argue for the correctness of a loop design

Use the given loop invariant (LI) to show that the following algorithms satisfy their specifications.

1. **Factorial (n)**
```
y ← 1
For i ← 2 to n
    > LI: y = (i – 1 )!
    y ← y × i
Return y
> Post: y = n !
```

2. **Product (a, b)**
```
>Performs multiplication
y ← 0
For i ← 1 to a
    > LI: y = (i– 1) × b
    y ← y + b
Return y
```

3. **Pow (a, b)**
```
> returns $a^b$
y ← 1
i ← 0
while i < b
    > LI: $y = a^{i-1}$
    y ← a × y
    i ← i + 1
return y
```

4. **Sum (A)**
```
> Computes sum of A
y ← 0
i ← 1
while i ≤ |A|
    > LI: y = sum(A[1 .. i– 1])
    y ← y + A[ i ]
    i ← i + 1
return y
```

5. **Max (*A*)**

```
> Returns largest element of A
y ← A[1]
i ← 1
while i < |A|
    > LI: y = max{A[1 .. i – 1]}
    if y < A[ i ]
        y ← A[ i ]
    i ← i + 1
return y
```

6. **Quotient(*a, b*)**

```
> Returns ⌊a / b⌋
y ← 0
i ← a
while i > 0
    > LI: y = ⌊(i – 1) / b⌋
    i ← i – b
    y ← y + 1
Return y
```

2.2c Write a simulation using a random number generator†

Using reference material and your knowledge of Java, translate one of the following files (available in a .zip archive) from C++ or C to Java:

1. Random.c 2. *Rolldie.cpp*

2.2c Define a two-dimensional array†

1. Given user inputs of a set of four quarterly sales figures for each of three regions, calculate totals by quarter and by region. *Example:*

	Q1	Q2	Q3	Q4
R1	236	311	398	892
R2	128	232	109	673
R3	776	897	349	1031

2. Declare one array of sales figures in five corporate divisions, one value of an appropriate numeric type for each division in each quarter of the year (Q1, Q2, Q3, Q4).
3. Write a program that declares and initializes a two-dimensional array that has two rows of seven integers each. Define and call a method that accepts such an array as a parameter and displays all the elements of the array in two rows and seven columns.
4. Define a data structure that will store a 3 × 3 tic-tac-toe board, consisting of *char* values ‘ ‘, ‘O’, or ‘X’. Write looping code to fill it with spaces.
5. Define an array that stores the number of computers sold for each 7-day week in a month of up to 5 weeks. Read data into it from a file and display the data in a grid.

2.3a Explain a search algorithm

1. Roughly how many steps may the linear search of an *n*-element array take? Explain.
2. Roughly how many steps does it take to merge two arrays of *n* elements each? Explain.
3. Which algorithm we discussed is roughly the one used to look up a number in a phone book? Explain.
4. Roughly how many steps does the binary search of an *n*-element array take? Explain.
5. How many values in a 10000-element sorted array would a linear search algorithm have to look at, on average, to find a search key? Explain.
6. Roughly how many comparisons would the binary search make in an array of 100 elements? Explain.
7. Explain why the binary search is faster than the linear search, describing each algorithm enough to show what you mean.

2.3b Search an array or merge arrays*†

1. Rewrite the linear-search algorithm to start at the *end* of an array, moving leftward.
2. Write a linear search that counts the number of occurrences of the search key.
3. Write an iterative version of the binary search.
4. Write a linear search of an array of objects.
5. Write a recursive version of the binary search.
6. Write pseudocode or Java code that outputs the subscript of the *last* element of an integer array that has the value 5, or outputs “not found” if 5 is not in the array.
7. Write a Java method to do one of the following with an array parameter or parameters:
(a) perform a binary search of a sorted array of *int*;
(b) insert a new value into a sorted array of *double*;
(c) sort an array of *double*;
(d) efficiently merge two sorted arrays of *int* into a third one, which should end up sorted
8. Write a method that accepts as parameters a sorted array of integers, its occupancy (current size), and a value to be inserted. The method should insert the value at its appropriate location. Write preconditions, postconditions, and loop invariants to argue that the method is correct.
9. Write a method that takes two C-style strings, *s1* and *s2,* as parameters and returns the location of the first match between *s2* and a substring of *s1;* that is, for *find(“concat”, “cat”)*, the return value would be 3. What is the complexity of the method?
10. Write a method that takes four sorted integer arrays as parameters and merges the first three into the fourth.
11. Merge two sorted subarrays of a single array into a single (second) sorted array, given array *A*; the length of A, *len*; and the subscript of the first element of the second subarray, *subarray2*. Write an appropriate precondition and postcondition for your method as comments. *Sample parameter data:*
A = {1,2,4,5,3,5,6,8,9}, *len* = 9, *subarray2* = 4
12. Write pseudocode or Java code for a method that will efficiently merge its two parameters, sorted arrays *A* and *B* of sizes *da* and *db*, into the third parameter array, *C*, leaving *C* sorted in ascending order. Discuss its complexity.

2.4a Give the output of a nested loop**

How many stars are printed? Explain.

```
1. for (i = n; i > 0; i--)
       for (j = i; j <= n; ++j)
           out.print( "*");
2. for (i = 20; i < n; i += 2)
      for (j = 0; j < 100; ++j)
          out.print("*");
3. for (i = 0; i < n; ++i)
       for (j = 0; j < n; ++j)
           out.print( "*");
```

4.
```
for (i = 0; i < n; ++i)
     for (j = i; j < n; ++j)
          out.print( "*");
```
5.
```
int count = 0;
for(int i=0; i < 50; ++i)
    for(int j=0; j < 50; ++j)
        ++count;
out.print(count);
```
6.
```
for (i = 0; i < 50; ++i)
    for (j = 0; j < n; ++j)
         out.print( "*");
```
7.
```
for (j = 10; j < 50; ++j)
    for (i = 0; i < n; ++i)
        out.print( "*");
```

2.4b Write a nested loop**

1. Correct the following code to display the number of duplicate values in the array A.
```
int dups;
for (int i = 0, i < A.length, i++)
    for (int j = 0, j < A.length, j++)
        if (A[i] = A[j])
            dups++;
System.out.println("dups = " + dups);;
```
2. Correct the following code to tell whether n is a prime number:
```
boolean isPrime = false;
for (int i = 0; i > n; i++)
    if (n % i != 0)
        isPrime = true;
```

Write a method to do the following:

3. Display "unique" if all elements of an *int* array are unique (no duplicates); otherwise, "duplicates."
4. determine the winner in a vote among candidates coded by integers; that is, find the integer that was most common in the array.
5. Find all the divisors of integer *n*.
6. Find the starting index of the longest ascending run in an array
7. Find the starting index of the longest run of the same value in an array
8. Find the largest and smallest divisor of integer *n*.
9. Display an addition table.
10. Display a multiplication table.
11. Accept an array of integers as a parameter and return the length of the longest series of consecutive array elements, starting at any position in the array, that have the value 2. For example, on input of (1, 2, 2, 3, 3, 3, 2, 2), the output should be 3, because there are 3 consecutive 3s.
12. Write a method that accepts an array of integers as a parameter and returns the length of the longest series of consecutive elements that have the same value. What is its complexity?
13. Write pseudocode or program code for an algorithm that takes as parameters an array of integers and its size, and that returns the *mode*, the element value that appears most often in the array. State its complexity in big-O notation and justify your analysis.
14. Given a set of tasks, each with start and finish times, find the smallest number of non-overlapping tasks that fill the day, from 1:00 to 6:00. *Example:* { (A,1,3), (B,2,4), (C,3,5), (D,4,6), (E, 5,6), (F,1,4), (G,1,2) } [MOVE TO CSCI 271]

2.4c Explain a sorting algorithm

1. Roughly how many steps can the *selection sort*, *insertion sort*, or *bubble sort* of an *n*-element array be expected to take? Explain.
2. Describe an algorithm to arrange an *n*-element array in ascending order, and explain roughly how many steps it will take.
3. Describe the loop or loops in a sorting algorithm.
4. Describe the *bubble sort* and its performance.
5. An algorithm that involves loops of roughly *n* iterations, nested to two levels, would take about how many steps? Explain.
6. An algorithm that involves loops of some constant number of iterations, nested to *n* levels, would take how many steps?
7. Explain roughly how many comparisons would be necessary, using the Bubble algorithm, to sort: (a) a 100-element array? (b) A 1000-element array.
8. State the running time of an algorithm that uses nested loops, each of which takes *S* steps, and justify your answer.
9. Could an array of size *n* be sorted in fewer than *n* comparisons? Why or why not?
10. Write a paragraph or two discussing the problem of searching and sorting arrays. Include considerations of time complexity and use big-O notation to compare algorithms.

2.4d Sort an array*†

1. Write a program that reads a series of floating-point values from a text file into an array and displays them, sorted ascending.
2. Modify Bubble Sort to sort *descending*.
3. Rewrite Insertion Sort so that it repeatedly finds the highest value in a sequence at the left of an array, and moves the value into a sequence at the right.
4. Write and test the Selection Sort algorithm.

Multiple-choice questions on Topic 3: Classes

3.0a Recall basic class concepts*

1. Data abstraction and Java classes

1. A class is a (a) data type; (b) data item; (c) expression; (d) process; (e) method
2. In a Java program, the expression *part.price()* is (a) a class; (b) a object; (c) a method header; (d) a method call; (e) a variable
3. Up to how many objects may be instances of the same class? (a) 0; (b) 1; (c) 2; (d) several; (e) there is no particular limit
4. A compound data type may be created with the keyword (a) *class*; (b) *enum*; (c) *int*; (d) *for;* (e) *compound*
5. It is often considered useful to (a) create a class especially for a certain method; (b) create a method especially for a certain class; (c) declare an object without a type name; (d) declare static public methods to manipulate private members; (e) call private methods from *main*
6. A data item may have several data attributes and a set of characteristic behaviors is (a) a type; (b) an integer; (c) an object; (d) a class; (e) a control structure
7. Creating new data types is (a) impossible; (b) data abstraction; (c) procedural abstraction; (d) to be discouraged; (e) a compiler error
8. The state of an object is (a) the process that created it; (b) the set of values of its data attributes; (c) the eventual destiny of the object; (d) the operations associated with it; (e) none of these
9. _____ exhibit common structure and behavior (a) methods; (b) processes; (c) instances of classes; (d) member items; (e) none of these
10. Data abstraction is (a) the declaration of variables; (b) the creation of subprograms; (c) the definition of new data types; (d) the processing of input; (e) the definition of methods
11. All classes always model
    i. state
    ii. behavior
    iii. user preferences

    (a) i only; (b) i and ii only; (c) i, ii, and iii; (d) i, iii, and iv; (e) iii and iv
12. An object implements the concept of a (a) string; (b) integer; (c) database record; (d) database table; (e) database query
13. Objects are created using (a) *int*; (b) *while*; (c) *if*; (d) *new*; (e) *switch*
14. A reference stores (a) an address; (b) a set; (c) input data; (d) a function; (e) nothing
15. To allocate memory for an object, we use (a) a loop; (b) a declaration; (c) *new*; (d) *delete*; (e) none of these
16. The operand to *new* is (a) a class name; (b) a reference; (c) the size of a dynamic variable being allocated; (d) the number of variables being allocated; (e) none of these
17. An object is allocated where? (a) disk; (b) stack; (c) global variable memory; (d) heap; (e) embedded in machine code

2. Encapsulation

1. The *interface* of a class is the set of its (a) data members; (b) ancestors; (c) descendants; (d) private member items; (e) public member items
2. The implementation of a class consists of (a) the names of its methods; (b) its ancestors; (c) its descendants; (d) private members; (e) public members
3. A mutator (a) calls a method; (b) changes the value of a class member; (c) is a global method; (d) returns a value; (e) has a reference parameter
4. An accessor (a) calls a method; (b) changes the value of a class member; (c) is a global method; (d) returns the value of a class member; (e) has a reference parameter
5. What reserved word is used to prevent a member item from being accessed from outside a class's methods? (a) *restricted*; (b) *local*; (c) *private*; (d) *public*; (e) *protected*
6. An object has its members initialized in the statement that declares it, by (a) the initialization operator once for each member; (b) a constructor; (c) an access method; (d) *Scanner*; (e) *System.out*
7. Encapsulation associates (a) attributes and behaviors with a data type; (b) implementations with interfaces; (c) input with output; (d) the user with private members; (e) none of these
8. To display a private member, you would have to (a) call a constructor; (b) initialize public members; (c) call an accessor; (d) call a destructor; (e) use the scope resolution operator
9. A(n) ___ initializes member data items (a) constructor; (b) destructor; (c) input/output method; (d) access method; (e) event handler
10. What take the name of their classes? (a) member data items; (b) member methods; (c) constructors; (d) access methods; (e) instances
11. What executes automatically when an object goes out of scope? (a) a constructor; (b) a destructor; (c) an inline member; (d) an access method; (e) none of these
12. The scope of access of a class is (a) its declaration; (b) its constructor; (c) its access methods; (d) its member methods; (e) its public member methods
13. (a) ; (b) ; (c) ; (d) ; (e)

3.0b Recall advanced class concepts^m

3. Code reuse and class debugging

1. Every class has all the methods of which class? (a) *Scanner*; (b) *System*; (c) *Item*; (d) *Object*; (e) *Applet*
2. Every class has a ___ method by default (a) keyboard input; (b) screen output; (c) file input; (d) file output; (e) *toString*
3. A *class invariant* is an assertion that (a) never holds for any instance; (b) always holds for some instance; (c) always holds for every instance; (d) may hold for some instance; (e) sometimes holds for every instance
4. An assertion that should always hold for every instance of a class is a (a) comment; (b) Boolean variable; (c) class invariant; (d) loop invariant; (e) precondition

5. To compare objects, what is used? (a) *if*; (b) =; (c) ==; (d) *equals()*; (e) !=
6. Well-defined classes use (a) public data members; (b) coupling; (c) data hiding; (d) data revelation; (e) data exposure
7. Nesting objects within other objects is called (a) encapsulation; (b) containment; (c) internalization; (d) protection; (e) privacy

4. Exception handling

1. Java's exception-handling feature is an alternative to _________ error handling (a) local; (b) global; (c) ad hoc; (d) run time; (e) a lack of
2. The keyword *throw* is used in (a) polymorphism; (b) loops; (c) virtual methods; (d) templates; (e) exception handling
3. An exception is a(n) (a) object; (b) method; (c) input item; (d) class; (e) comment
4. A block of code in which exceptions should be detected is given the header (a) *try*; (b) *throw*; (c) *catch*; (d) *exception*; (e) *class*
5. Exceptions are a way to handle ___ errors (a) syntax; (b) logic; (c) output; (d) runtime; (e) computational
6. Runtime errors may be handled by (a) syntax checking; (b) tracing; (c) code reviews; (d) exceptions; (e) they cannot be handled
7. Attempting to open a nonexistent file to read always (a) crashes a program; (b) generates an exception; (c) produces a syntax error; (d) produces false output; (e) produces correct output
8. If a program attempts to read a file that contains no more data, it (a) crashes; (b) generates an exception; (c) produces a syntax error; (d) produces false output; (e) produces correct output
9. An attempt to divide a number by zero results in (a) a syntax error; (b) incorrect output; (c) (–1); (d) an exception; (e) nothing
10. With exceptions, error information may be passed directly to (a) *main*; (b) a constructor; (c) error-handling code; (d) debugging code; (e) error-generating code
11. *Try* is used mainly with (a) loops; (b) branches; (c) exceptions; (d) files; (e) debugging
12. *Catch* is used mainly with (a) loops; (b) branches; (c) exceptions; (d) files; (e) debugging

Problems for topic 3: Class design

Topic objective:
Define and test Java classes, explaining object-oriented design concepts

3.1a Describe Java data abstraction**

1. What is one way to share data among methods without using parameters or return values? What are its disadvantages?
2. Explain why *classes* are used in programs.
3. Describe what is meant by *data abstraction* and how it is supported in Java.
4. Describe how Java supports the notion of *records* of the kind found in *database tables*.
5. Distinguish between an *object* and a *class*.
6. Name, and describe the purpose of, the kind of abstraction that occurs when defining Java classes.
7. What data is abstracted in data abstraction?

3.1b Define a Java class**

[Exercise: See 3.3b below, option (a).]

Define a class, with some basic members, to represent:

1. *machine parts*, with a name, an inventory quantity, and a price.
2. *models of refrigerator*, each with a price, an identification number, and a name.
3. *retail products*, with name, price, identification number, and quantity sold.
4. *cars*, naming some of its features and allowing values to be specified for them.
5. *times of the day* that could be expressed on a clock.
6. *durations* of time using the units of time found in a clock.
7. *dates on a calendar*.
8. *a bank customer's transaction* such as the kind that an ATM could carry out.
9. *rational numbers*, each with a numerator and a denominator, which are integers. Include methods *plus*, *minus*, *times*.
10. *sales records*, each having a date, item identification number, and quantity. You may assume that a class, *Date*, exists.
11. *student*, with names, addresses, and IDs.

3.2a Describe Java encapsulation*

1. Distinguish the *interface* of a class from the *implementation*.
2. Compare class *interface* with *implementation*, with respect to the notions of *private* and *public.*
3. Describe how notions of *cohesion* and *coupling* are used in class design.
4. What do class-design principles, presented in this course, say about making data members accessible to client code?
5. Distinguish *accessors* from *mutators*.
6. Is it recommended that data members of a class be public or private? Explain.
7. Is it recommended that accessors be public or private? Explain.
8. How are the data members of a class initialized?
9. How is a *constructor* used?
10. Could it be useful for a class to have more than one constructor? Explain.
11. Explain why *data hiding* is useful in object-oriented programming.
12. What is *encapsulation* and how is it implemented?

3.2b Write and document a class with encapsulation*

See your answer to a problem under objective 3.1c.
Include constructors, accessors, and mutators.
Explain its interface and its implementation parts.

3.3a Describe class debugging concepts

1. What happens when two objects created using *new* are compared to each other using the operator = =?
2. What happens when two objects created using *new* are compared to each other using the operator !=?
3. Suppose an object is created using *new*. What happens when its value is assigned to another variable of the same class, as in *Employee emp1 = new Employee(), emp2 = emp1;* ?
4. Describe a risk of creating an object created using *new*, if the class does not have a declared constructor.
5. What happens if an accessor is declared as *private*?
6. Comment on the idea of declaring a constructor *private*.

3.3b Test and debug a class**†

Options:

(a) Describe an error that you make in solving a problem under 3.1 or 3.2b above; describe how you corrected it, and show the erroneous and debugged versions of your Java code.

(b) Instantiate and debug the following classes, describing errors and how to fix them:

1.
```
public class Hmm
{
  public void Hmm() {};
  public int get_x() { return x; }
  public int get_y() { return y; }
  int x, y;
}
```

2.
```
class Employee
{
  public Employee(String nm,int hrs);
  public String get_name();
  public void set_name(String nm);
  public void set_hours(int hrs);
  String name;
  int hours;
}
```

3.3c Locate a fault in a multi-method class†

Report a fault and your correction of it, in version 4 of your semester project.

3.4a Explain exception handling*

1. What keyword is used to label a block of code in which exceptions will be handled? Explain.
2. Explain the Java feature that permits many kinds of errors to be detected and handled in one place.
3. What keyword is used as the header for a block of code in which exceptions should be detected? Explain.

4. What keyword transfers control to code that will handle an exception? Explain.
5. Describe *exception handling* in Java. Include some details.
6. What occurs when a Java program attempts to open a nonexistent file?

3.4b Use exceptions†

Write a program that does the following, using Java exception handling.

1. Call a method to open a file, detecting a file-not-found error in the method but handling it in *main*.
2. Attempts to read a file past end-of-file.
3. Divides by zero.
4. Handles a file-not-found error in your semester project.
5. Modifies *div2.java* to display trace output when the constructor and destructor of the *MathError* class execute.

Multiple-choice questions on Topic 4: Collections

4.0a Recall basic collection concepts*

1. Object-oriented design

1. Object-oriented design focuses problem solving on (a) categories of things; (b) processes; (c) methods; (d) integers
2. The *state* of an object is (a) the process that created it; (b) the set of values of its data attributes; (c) the eventual destiny of the object; (d) the operations associated with it; (e) none of these
3. The *has-a* relationship signifies (a) encapsulation; (b) containment; (c) linking; (d) inheritance; (e) polymorphism
4. Object-oriented design begins by looking for (a) categories of things; (b) processes; (c) relationships; (d) operations; (e) none of these
5. An object-oriented design may reflect ________ relationships between data types. (a) greater-than; (b) containment; (c) calling; (d) instantiation; (e) reference.
6. *Structured design* breaks problem solutions down through (a) iteration; (b) modular decomposition; (c) object-oriented analysis; (d) bottom-up design; (e) none of these
7. A ______ may inherit structure and behavior. (a) derived class; (b) template; (c) function; (d) client; (e) none of these
8. The *kind-of* relationship signifies (a) containment; (b) polymorphism; (c) inheritance; (d) encapsulation; (e) pointer types
9. Which of the following is *not* a criterion for including a candidate object in an object-oriented model? (a) need for persistent data items; (b) common attributes; (c) common operations; (d) an item has only one attribute; (e) state
10. UML is (a) a language used for design of interactive systems HTML; (b) machine language; (c) database query language; (d) UML; (e) none of these
11. UML is designed to support (a) Windows; (b) object-oriented design; (c) users; (d) coders; (e) marketers
12. A class with one or more fields that are objects is a (a) container; (b) iterator; (c) base class; (d) derived class; (e) friend class
13. A container class is one that has a(n) _________ member. (a) inline; (b) object; (c) pointer; (d) constructor; (e) access function

2. Collections as arrays of objects

1. A collection is (a) atomic; (b) a set of objects of the same class; (c) a compiler operation; (d) a predefined class; (e) an object with no fields
2. What is an object-oriented way to store and maintain multiple objects of the same type? (a) each as a member of one object; (b) as local variables; (c) as constants; (d) as parameters; (e) as a collection
3. The members of a collection may be stored in a(n) (a) integer; (b) string; (c) array; (d) object; (e) none of these
4. A collection is implemented as a(n) (a) integer; (b) string; (c) class; (d) method; (e) none of these
5. What is an object-oriented way to store and maintain multiple objects of the same type? (a) each as a member of one object; (b) as local variables; (c) as constants; (d) as parameters; (e) as a collection
6. What kind of data structure would you use to store information about a set of customers and to look up a customer's address knowing only the last name? (a) object; (b) array of objects; (c) two-dimensional array; (d) string; (e) object containing a string for each customer
7. A collection typically consists of (a) many items of different types; (b) just one item; (c) many objects of the same type; (d) an array of characters
8. If a sales transaction has an item ID, a customer ID, and a date, then what kind of data structure would you use to store information about a set of sales items? (a) object; (b) array of objects; (c) two-dimensional array; (d) string; (e) object containing a string for each custom
9. If a registration transaction has three fields, then what kind of data type would you use to store a set of registration items? (a) object; (b) array of objects; (c) two-dimensional array; (d) string; (e) object containing a string for each customer
10. Any database is (a) an atomic data item; (b) a method; (c) a collection; (d) a derived class; (e) an interface
11. An array would be an appropriate way to store (a) a set of database tables; (b) one database table; (c) one database record; (d) one database field; (e) one database field schema
12. A database normally consists of (a) pixels; (b) tables; (c) keys; (d) protocols; (e) none of these
13. A selection query corresponds to (a) a table; (b) a view; (c) a logical assertion; (d) a set of records; (e) all of these
14. In databases, an object or instance corresponds to a (a) record; (b) table; (c) bit; (d) relation; (e) all of these
15. In databases, an entity or class of objects is implemented by a (a) record; (b) table; (c) bit; (d) relation; (e) all of these
16. To display information from a database, we use a (a) format command; (b) named style; (c) master page; (d) query; (e) all of these
17. Non-duplication of data in tables is enforced by use of (a) formulas; (b) primary keys; (c) formats; (d) protocols; (e) all of these
18. A database table’s columns correspond to (a) records; (b) tables; (c) instances; (d) attributes; (e) all of these

4.0b Recall advanced collection concepts

3. Java ArrayLists and generic collections

1. *ArrayList* is a(n) (a) predefined object; (b) wrapper class; (c) generic class; (d) method; (e) none of these
2. One wrapper class is (a) *Integer*; (b) *int*; (c) *String*; (d) *char*; (e) none of these
3. A single *ArrayList* object (a) may contain elements of multiple types; (b) may be expanded in size; (c) is always declared with initialization; (d) is always composed of strings; (e) is always composed of integers
4. Wrapper classes (a) are user defined; (b) convert items of primitive types into objects; (c) implement arrays; (d) implement compound types; (e) teach hip-hop

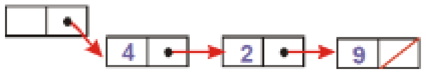
5. An iterator object is associated with a (a) branch; (b) global function; (c) global variable; (d) collection; (e) none of these
6. The strategy behind using iterator classes separate from their collections is to separate the abstraction of containment from a form of __________ abstraction. (a) data; (b) procedural; (c) control; (d) design; (e) object
7. Classes in the Java collection framework all (a) are arrays; (b) implement the *Collection* interface; (c) are implemented by the *Collection* interface; (d) lack methods; (e) are collections of strings

4. File-maintenance applications

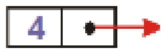
1. Which is *not* supported by typical file-maintenance applications? (a) create new document; (b) upload web site; (c) enable document update; (d) export to new file format; (e) none of these
2. File I/O in Java requires (a) administrator permission; (b) Internet access; (c) creation of *FileReader* or *PrintWriter* objects; (d) a special kind of compiler; (e) none of these
3. If an attempt is made to open a file that does not exist, (a) the program crashes; (b) an alert dialog is displayed; (c) a file exception is thrown; (d) a file is opened at random; (e) none of these
4. Regression testing finds errors introduced by the ____ process (a) specification; (b) design; (c) coding; (d) maintenance; (e) none of these
5. Integration testing determines whether (a) one subprogram works; (b) separately developed modules work together; (c) a distributed application works; (d) a networked system works; (e) none of these
6. The updating time of a random-access file's is analogous to (a) a sequential-access file update; (b) an array element update; (c) a linear search; (d) a linked-list traversal; (e) an array sort

5. Linked lists

1. The data type of a node of a linked list is (a) array; (b) simple; (c) class; (d) collection; (e) pointer
2. A linked list is one way to implement a(n) (a) function; (b) application class; (c) loop; (d) collection; (e) tree
3. To reach a given list node from the beginning (a) takes one step; (b) takes two steps; (c) requires that all its predecessors be visited; (d) requires that all its successors be visited
4. How many reference members, minimum, must a linked-list node have? (a) 0; (b) 1; (c) 2; (d) 3; (e) 4
5. A linked-list node object must have how many members? (a) 0; (b) 1; (c) 2; (d) 3; (e) more than 3
6. One member of a linked-list node must be a(n) (a) character; (b) integer; (c) node; (d) reference; (e) none of these
7. To implement node deletion with linked lists in an object-oriented approach, we would use (a) a loop in *main*; (b) a global function; (c) a global variable; (d) a method of the *nodes* class; (e) a method of the list class
8. Which node of a singly linked list must be known to delete a node? (a) none; (b) the first; (c) the predecessor of the node to delete; (d) the node to delete; (e) the successor of the node to delete
9. A linked list is traversed in time proportional to (a) $\log_n$; (b) 1; (c) n; (d) $n\log n$; (e) n^2
10. To prepend an item to a linked list of n nodes takes ____ steps (a) 0; (b) 2; (c) 50; (d) n; (e) an unpredictable number of
11. To append an item to a linked list of n nodes takes ____ steps (a) 0; (b) 2; (c) 50; (d) n; (e) an unpredictable number of
12. Deleting a node from a linked list normally involves (a) changing the data value in the node; (b) changing the value of the node's *next* reference; (c) changing the link in another node so as not to point to the deleted node; (d) changing the link in another node to point to the deleted node
13. Searching a linked list of n nodes may require visiting up to how many nodes? (a) 1; (b) 2; (c) n; (d) n^2; (e) none of these
14. The diagram below is of a(n) (a) array; (b) linked list; (c) list node; (d) tree; (e) atomic data item



15. The diagram below is of a(n) (a) array; (b) linked list; (c) list node; (d) tree; (e) atomic data item



Problems for Topic 4: Collections

Topic objective:
Design, implement, and explain a multi-class application that manages a disk-based collection

4.1 Explain what a collection is**

1. Explain how we discussed implementing a database table in Java.
2. What is a collection class?
3. Describe Java *ArrayLists*, distinguishing them from ordinary arrays.
4. What is a Java *wrapper class*?
5. How may a collection be *traversed* in Java?
6. How would you code an algorithm to find the *Employee* object with the highest salary in an array-based collection of *Employee* objects?
7. Explain what an *iterator* class is, and how it is used.

4.2a Define a collection class*

Define a class as described below.

1. An inventory class -- a collection of instances of class *parts*.
2. A *Roster* as a collection class implemented with an array of *Employee* items.
3. A collection of bank transactions. A transaction has an account ID, a date, and an amount. Write a method to compare two transactions by amount.
4. A collection of students.
5. A collection of courses.

4.2b Test a collection*†

Use Java to define a class as in objective 4.2b. Write methods to insert into, delete from, and search the collection. Compile and test.

4.3a Describe generic collections

1. Describe what a Java generic collection is.
2. How are iterators used with collections?
3. Describe the *ArrayList* class.
4. Describe predefined Java operations on collections.
5. Explain how it is possible to search or sort a Java collection without writing the search or sort algorithm from scratch.
6. Explain the following Java code:
```
ArrayList <Integer> A = new ArrayList<Integer>;
A.add(2);
```

4.3b Define and test a generic collection†

Use an ArrayList or other standard Java collection interface or class to implement a collection of the same type of objects as you did for objective 4.2b.

4.3c Define and use an iterator†

Define an iterator class in your project (part 1) and use it to access elements of your collection.

4.4a Write and test a file-maintenance application*†

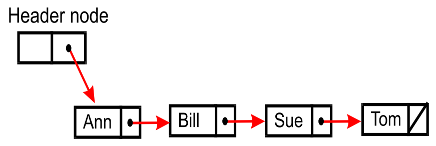
1. Define a collection of library-book data records. Each book has a title, author, and catalog number. In Java, define appropriate data types and define methods to search the collection for a particular catalog number. Discuss two ways to implement the collection other than the one you chose; what are their advantages and disadvantages?
2. Design or code an implementation of a linked list of *float*s. Include necessary class declarations and method definitions to implement insertion of one value, in ascending numeric order, and display of all values.
3. Write declarations for classes needed to implement a collection of butterflies. A butterfly has a wing span, a color, and a name. Include some appropriate method declarations for operations on the butterflies and the collection.

4.4b Describe random-access files

1. Distinguish a *random-access* file from a sequential-access one.
2. Describe an alternative to sequential file access.
3. Explain advantages of random access over sequential access and give an example application.
4. Describe Java's support for random-access files.
5. Describe a way to create a large file that may be updated *without* reading the entire file into memory, modifying its contents, and writing it back to disk.

4.5a Explain linked lists

1. Describe the procedures of inserting an item into a linked list and deleting an item from a linked list.
2. Describe how, knowing the header of the linked list below, to find the node position where a new node, with data value "Jim", should be inserted, in alphabetical order.
3. Describe the components of a linked list and how they are linked together.
4. How might you insert one linked list (*L2*) into another (*L1*) at some desired location whose predecessor node is stored in reference *pn*?
5. Describe the steps necessary to *append* one linked list, *L2*, to another, *L1*.
6. Once you know where to insert a node with the value "Jim," in the diagram below, what steps should be taken to link it into the list?



4.5b Define a linked-list class†

Declare a data type for a linked-list node as specified below.

1. Design or code an implementation of a linked list of *float*s. Include necessary class declarations and method definitions to implement insertion of one value and display of all values.
2. Write an algorithm that tells whether a linked list of integers contains *two or more* nodes that store a given value. Argue for its correctness and discuss its complexity
3. Write Java code to insert a new value into a linked list of integers.
4. Write a program to search of a linked list for a given value.
5. Write a program to determine whether a singly-linked list of integers stores any duplicate values. What is the complexity of your algorithm?
6. Find the maximum element of a linked list of integers or strings, or objects with some field that is comparable.
7. Add all the elements of a linked list of integers or *doubles*.
8. Tell if all elements of a linked list are in ascending order.

Multiple-choice questions on Topic 5: Inheritance

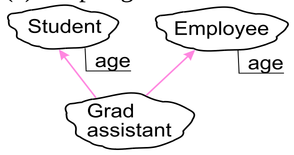
5.0a Recall basic inheritance concepts*

1. Inheritance and object-oriented design

1. The object-oriented feature that allows one class to acquire all the attributes and behaviors of another one is (a) inheritance; (b) containment; (c) composition; (d) data abstraction; (e) pass by reference
2. A *taxonomy* uses (a) encapsulation; (b) containment; (c) collection; (d) inheritance; (e) cohesion
3. An ISA or kind-of relationship between two classes is a case of (a) encapsulation; (b) inheritance; (c) message passing; (d) container classes; (e) none of the above
4. Inheritance is a relationship between (a) classes; (b) objects; (c) simple variables; (d) pointers; (e) methods
5. A subclass is (a) a class from which another class inherits; (b) a derived class; (c) a based class; (d) a class declared inside another class
6. A ______ may inherit structure and behavior. (a) derived class; (b) template; (c) method; (d) client; (e) none of these
7. What feature allows us to define a class that automatically has all the members of another class? (a) encapsulation; (b) top-down design; (c) constructors; (d) inheritance; (e) all of these
8. The *isa* relationship signifies (a) containment; (b) polymorphism; (c) inheritance; (d) encapsulation; (e) pointer types
9. A class hierarchy reflects ________ relationships between data types. (a) membership; (b) kind-of; (c) calling; (d) instantiation; (e) reference.
10. The *has-a* relationship signifies (a) encapsulation; (b) containment; (c) linking; (d) inheritance; (e) polymorphism
11. In object-oriented design, the concept of subcategories could be implemented by (a) encapsulation; (b) reference; (c) analogy; (d) inheritance; (e) polymorphism
12. Which is *not* a relationship between classes? (a) containment; (b) inheritance; (c) friendship; (d) reference; (e) negation
13. In object-oriented design, the concept of subcategories could be implemented by (a) encapsulation; (b) reference; (c) analogy; (d) inheritance; (e) polymorphism
14. The three main features that characterize object-oriented design are encapsulation, inheritance, and (a) data hiding; (b) functional decomposition; (c) polymorphism; (d) persistence; (e) planning
15. A *Menu* class could reasonably inherit from a _______ class. (a) screen location; (b) color; (c) window; (d) text; (e) application

2. Inheritance in Java

1. In declaring a subclass by inheritance in Java, we follow the subclass's name with ______ and the name of the base class. (a) period; (b) *extends*; (c) semicolon; (d) colon; (e) slash
2. Inheritance is said to replace certain uses of which keyword? (a) *switch*; (b) *if*; (c) *while*; (d) *void*; (e) *class*
3. Protected members are accessible to (a) a class's ancestors; (b) a class's descendants; (c) any function; (d) *main*; (e) library methods
4. After the declaration *class Supervisor extends Employee { }* (a) a variable named *Supervisor* exists; (b) any *Supervisor* object will have all the members that an *Employee* object does; (c) any *Supervisor* object has an *Employee* member; (d) any *Employee* object has a *Supervisor* member; (e) none of these is true
5. In declaring a subclass by inheritance in Java, we follow the subclass's name with and the name of the base class. (a) a period; (b) a colon; (c) *extends*; (d) *subclass*; (e) slash
6. The Java keyword *extends* implements (a) encapsulation; (b) containment; (c) collection; (d) inheritance; (e) cohesion
7. The diagram below illustrates (a) encapsulation; (b) containment; (c) multiple inheritance; (d) cohesion; (e) coupling



5.0b Recall advanced inheritance concepts

3. Interface types

1. Interface types support (a) verification; (b) performance; (c) reusability; (d) user friendliness; (e) GUI design
2. To implement an interface class, declare a class that implements it and define (a) its abstract methods; (b) instance variables; (c) I/O methods; (d) a GUI; (e) a base class
3. Interface types support (a) containment relationships; (b) *is-a* relationships; (c) primitive types; (d) time efficiency; (e) user documentation
4. An alternative to inheritance in Java uses what keyword? (a) *new*; (b) *uses*; (c) *extends*; (d) *interface*; (e) *while*
5. An alternative to inheritance in Java uses what keyword? (a) *new*; (b) *uses*; (c) *extends*; (d) *implements*; (e) *while*
6. The code *public Batter implements Comparable* uses a(n) (a) class; (b) control structure; (c) interface type; (d) method; (e) string

4. Polymorphism

1. With polymorphism, the behavior of an object depends on whether (a) it is an instance of a base or derived class; (b) it is statically allocated; (c) it is dynamically allocated; (d) it has a pointer member; (e) it has a private method
2. Determination of the call address of a method call at run time rather than compile time is (a) inheritance; (b) encapsulation; (c) early binding; (d) late binding; (e) forward reference
3. An array of _______ makes possible a collection of shapes of mixed classes with polymorphic behavior. (a) integers; (b) function pointers; (c) character pointers; (d) base-class references; (e) none of these
4. Polymorphism is an alternative to the use of (a) *if…else*; (b) *switch*; (c) *while*; (d) *goto*; (e) method calls

5. An abstract base class (a) has no virtual methods; (b) cannot be instantiated; (c) has a definition that may not be overridden; (d) has a null constructor; (e) inherits from a concrete base class
6. A generic array would be composed of (a) strings; (b) *void* references; (c) derived-class objects; (d) base-class objects; (e) *void* methods
7. In late binding, the address of a call to a virtual method is resolved at (a) compile time; (b) the time a program is loaded; (c) the time the method actually executes; (d) the time the method terminates; (e) the time the program terminates
8. What cannot be instantiated? (a) static methods; (b) derived classes; (c) abstract base classes; (d) collections; (e) overloading

Problems for Topic 5: Inheritance

Topic objective:
Explain and implement the notions of an inheritance hierarchy and of polymorphic behavior

5.1a Explain inheritance*

1. Name the two main kinds of relationship between two classes.
2. Why is it not appropriate to derive an *Employee* class from an *Address* class? After all, en employee has an address.
3. Distinguish *containment* from *inheritance*.
4. Describe *inheritance* in Java.
5. What are *base* and *derived classes*?
6. Describe a relationship between a *car* class and a *vehicle* class.
7. Explain use of the Java *extends* keyword.
8. When declaring a derived class, what keyword should you use before the base-class name to give the derived class but not its descendants access to private members?
9. What members of a base class are never accessible to instances of derived classes?

Write a diagram that shows classes and containment relationships among classes in the following scenarios:

10. A business-management application. The business sells different kinds of vehicles (cars, trucks, SUVs) to customers, and purchases them by submitting orders to the manufacturer. The company has sales employees and service employees.
11. A company's internal structure, consisting of sales and manufacturing departments; supervisors; and associates.
12. A college's information processing system. A college has students, courses, instructors, buildings, class schedules, staff members, and departments.
13. A retail firm, which has outlets, departments, customers, and wholesale vendors.
14. A student's academic environment. A student is in a major, in a department, and takes courses, with professors.

5.1b Design an inheritance hierarchy*

1. Draw an inheritance hierarchy that depicts the relationships among these categories: *people, employees, managers*, clerical staff, production staff, forepersons, vice presidents.
2. Write a declaration for classes *Employee* and *Person*. An employee is a kind of person but, unlike a person, should have a department number and title.
3. Name some of the attributes and operations related to *colleges, departments, institutions of learning, supervisors,* and *employees*. Describe the *kind-of* class relationships and draw a hierarchy diagram.
4. Write a diagram that describes the relationships among classes in a situation where a business sells different kinds of vehicles (cars, trucks, SUVs) and has sales employees and service employees.
5. Give the *kind-of* class relationships among the following. *companies, consulting firms, supervisors,* and *employees*.
6. Give the *kind-of* class relationships for a college's information processing system. A college has work-study students, professors, staff members, administrators, adjuncts, temporary instructors, and tenure-track faculty.
7. Write the design for a compiler environment. Include display entities (windows, menus, etc.) and tools used, such as the lexical analyzer, parser, code generator, editor, and debugger.
8. Express the relationship between *rental properties* and *apartments* in an inheritance example.

5.1c Explain issues raised by inheritance

1. What is the chief dilemma related to multiple inheritance and how is it resolved?
2. Consider the relationship among three classes, *instructor, teaching_assistant,* and *student*. A TA is a kind of student and also a kind of instructor. How would you design inheritance relationships among these classes? What problems arise, and how would Java resolve them?
3. Declare a class for lab assistants and related classes. A lab assistant is a student and also an employee. What type of class relationships exists and what problem arises? Suggest a way to solve it.
4. What is the *diamond inheritance problem* and how does Java address it?
5. What problem of inheritance arises where a person taking classes is both a *Student* and also a *State-resident*? Describe a Java solution.
6. What is *overloading*? Give an example.
7. What is *overriding*? Give an example.
8. What is *shadowing*? Give an example.
9. Distinguish *overloading* from *overriding.*
10. Distinguish *overloading* from *shadowing.*
11. Explain how a derived class may access a base-class method that the derived class has overridden.

5.2a Define and use a derived class*†

Implement in Java your design in your solution to a problem in objective 5.1b, Design an inheritance hierarchy.

5.2b Predict behavior of derived-class objects

Describe what happens when this program runs, including output.

```
class Drawable
{  public String toString() { return "shape"; }}

class Triangle extends Drawable
{  public String toString()
   { return "3-corner " + super.toString(); } }

class Square extends Drawable
{  public String toString() { return "4-corner"; } }

class Draw
{  public static void main(String[] args)
  { System.out.println(new Triangle()+"\n"+
        new Square()); } }
```

5.3 Test an interface type†

[To be defined.]

5.4a Explain polymorphism

1. What is late about *late binding*?
2. What is an *abstract base class*?
3. Name a kind of class that can have no instances and give an example.
4. Describe the kind of situation in which the address of a method is determined at run time.
5. Describe *polymorphism*.

5.4b Implement polymorphism†

[To be defined.]

Multiple-choice questions on Topic 6: Events, GUIs, graphics

6.0a Recall basic GUI concepts*

1. Developing event-driven software

1. Events are represented by (a) integers; (b) objects; (c) strings; (d) arrays; (e) none of these
2. JavaScript encodes (a) web-page formatting; (b) responses to input events; (c) database design; (d) statistical analysis; (e) none of these
3. JavaScript code is likely to appear in (a) spreadsheet formulas; (b) processor registers; (c) database queries; (d) HTML files; (e) none of these
4. To use a JavaScript variable, syntax requires it to be (a) requested; (b) initialized; (c) promulgated; (d) declared; (e) none of these
5. JavaScript encodes (a) web-page formatting; (b) responses to input events; (c) database design; (d) statistical analysis; (e) none of these
6. JavaScript is a _____ language (a) markup; (b) procedural; (c) functional; (d) machine; (e) none of these
7. *alert* is a(n) (a) JavaScript method; (b) HTML event; (c) HTML tag; (d) user warning; (e) none of these
8. A mouse click is (a) a data item; (b) an algorithm; (c) an event; (d) a prompt; (e) none of these
9. Command-line environments have (a) buttons; (b) windows; (c) icons; (d) dialogs; (e) prompts
10. The application programmer may use a pre-written user-interface library and write _______ handler methods to respond to different kinds of user input. (a) view; (b) string; (c) loop; (d) event; (e) mouse
11. An *event handler* (a) selects relevant events from all events; (b) responds to all events; (c) responds to a specific event; (d) is a loop; (e) is an object
12. An *event listener* (a) selects relevant events from all events; (b) responds to all events; (c) responds to a specific event; (d) is a loop; (e) is an object

2. Graphical user interface construction

1. Three important aspects of the design of an application are (a) method, view, and control; (b) data structure, model, and view; (c) control, design, and method; (d) control, model, and view; (e) analysis, data structure, and method
2. An instance of a *view* class may be (a) an array; (b) a character; (c) a window; (d) a screen driver; (e) an *int*
3. A GUI or application framework (a) is a data structure; (b) is a single class; (c) is a set of classes that defines a user interface; (d) is a form of documentation
4. In a GUI or application framework, a *window* is (a) a class; (b) an instance of a model class; (c) an instance of a view class; (d) an instance of a controller class; (e) an event
5. In a model/view/controller architecture, program execution is driven by (a) events; (b) views; (c) models; (d) commands; (e) interrupts
6. An application programmer writes an application class (a) as a base class for the library's derived class; (b) as a class derived from the library's base class; (c) as a class that is not part of an inheritance relationship; (d) to implement a model; (e) to implement a view
7. A *Menu* class could reasonably inherit from a _______ class. (a) screen location; (b) colors; (c) window; (d) text; (e) application
8. In a Java GUI, button presses are detected by a (a) virtual machine; (b) compiler; (c) HTML file; (d) listener object; (e) none of these
9. A virtual event handler will be called by an object of a ________ application class. (a) base; (b) derived; (c) model; (d) container; (e) view
10. An instance of a view class may be (a) an array; (b) a character; (c) a window; (d) a screen driver; (e) a menu option
11. A listener object may respond to a (a) text message; (b) voicemail; (c) client; (d) button press; (e) display of text

6.0b Recall basic Java graphics concepts

3. Java graphical tools

1. A Java program usually opens a window by (a) declaring a *JFrame* object; (b) calling the Windows API; (c) drawing a rectangle; (d) outputting a packet; (e) none of these
2. Java graphics uses (a) RGB color; (b) bitmap storage of lines; (c) *JFrame*; (d) exceptions; (e) none of these\
3. Drawing an image pixel by pixel (a) takes several seconds; (b) is bitmap rendering; (c) is vector rendering; (d) is impossible; (e) is always preferred
4. Drawing an image stored as instructions (a) takes several seconds; (b) is bitmap rendering; (c) is vector rendering; (d) is impossible; (e) is always preferred
5. Two advantages of vector over bitmap storage are (a) precision and compression; (b) precision and ease of editing; (c) compression and ease of editing; (d) compression and esthetics; (e) none of these
6. A *JFrame* object (a) may be a line; (b) creates a graphics window; (c) is an application; (d) is an event handler; (e) is an event
7. *setColor* affects a (a) shape; (b) window; (c) pixel; (d) line; (e) file
8. Pixels have what attribute? (a) *boolean*; (b) *String*; (c) *int*; (d) *double*; (e) *Color*
9. In *drawstring(String s, int a, int b)*, the parameter *a* is a(n) (a) character; (b) color; (c) horizontal location; (d) vertical location; (e) size
10. The virtual method *draw* of a *Shape* class works differently for different kinds of shapes; hence *draw* is (a) recursive; (b) virtual; (c) abstract; (d) null; (e) empty
11. The virtual method *draw* of a *Shape* class works differently for different kinds of shapes; hence *Shape* is (a) recursive; (b) virtual; (c) abstract; (d) null; (e) empty

Problems for Topic 6: GUIs and graphics

Topic objective:
Explain event-driven GUI development and use Java graphics libraries.

6.1a Explain event-driven programming*

1. Describe the tools used in *event-driven programming*.
2. What is an *event* in a graphical user interface; what is its role?
3. Describe what an *event handler* does.
4. Describe a loop present in all GUIs that terminates only when the application terminates.
5. How is input generated and responded to in a GUI?
6. Explain a kind of programming that centers around a loop that responds to input and other occurrences that need a response.
7. Describe *event-driven programming* and relate it to graphical interfaces. What is an *event*?

6.1b Write event-driven code†

[Problems to be defined.]

6.2a Describe elements of a graphical user interface*

1. What is the *model-view-controller* architecture?
2. Describe elements of a graphical user interface.
3. Describe the three kinds of classes that are said to work together in an event-driven, graphical environment or architecture.
4. Describe a method that the application programmer must redefine, when using a predefined application class library, to make the application respond in a customized way to user input.
5. Describe a class library that supplies models, views, and controllers for a user interface.
6. Describe a class that is inherited from very often to take advantage of a user-interface class library?
7. Describe a hierarchy of classes used to build a consistent user interface.
8. Describe an object-oriented way to implement a collection of diverse graphical objects to be drawn on the screen, and how you would use one loop to draw all the objects.
9. Describe how a skeleton Windows, X Windows, or Macintosh, application works behind the scenes. What classes are predefined and what tools are provided to the application programmer to ease the process of customizing a user interface?

6.2b Implement a Java-based GUI†

[To be defined.]

6.3a Describe Java graphics tools

1. How is a graphics window created and used in Java?
2. How may a graphics object be displayed in Java?
3. How does Java support the drawing of lines and shapes?
4. Distinguish bitmap from vector graphics.
5. Describe *JFrame* and its use.
6. Describe the specification of color for a pixel.
7. Distinguish drawing from filling.

6.3b Write a graphics application†

1. Implement a collection of diverse graphical objects to be drawn on the screen using Java graphics libraries. Use one loop to draw all objects, including line segments, circles, and rectangles.

Multiple-choice questions on Topic 7: Concurrency

7.0 Recall concurrency concepts

1. Concurrency and Java threads

1. Multitasking is (a) concurrent activities by multiple computers; (b) a series of activities by one processor; (c) one processor running multiple programs concurrently; (d) the same as multithreading; (e) parallel computing
2. Multithreading is (a) concurrent activities by multiple computers; (b) a series of activities by one processor; (c) one program running multiple processes concurrently; (d) the same as multitasking; (e) parallel computing
3. A thread simulates ownership of (a) a network; (b) a user account; (c) a computer's resources; (d) a processor; (e) none of these
4. Garbage collection in the background is an example of (a) batch processing; (b) non-concurrency; (c) parallel computing; (d) multithreading; (e) distributed computing
5. The Java *new* operator invokes (a) dynamic allocation of memory; (b) memory allocation on the stack; (c) deallocation of memory; (d) allocation and deallocation of secondary storage; (e) none of these
6. Threads of equal priority get processor resources by (a) pre-emption; (b) round-robin time slicing; (c) submitting requests; (d) user intervention; (e) random selection
7. The Java class that enables concurrency within a program is (a) *System*; (b) *String*; (c) *Output*; (d) *Memory*; (e) *Thread*
8. The Java scheduler ensures that the ____ thread executes at all times (a) shortest; (b) next-in-line; (c) highest-priority; (d) lowest-priority; (e) none of these
9. Daemon threads (a) execute in the foreground; (b) execute in the background; (c) can execute while waiting for the threads they serve to start; (d) all of these; (e) operate over the Internet
10. Java garbage collection is a (a) program; (b) distributed service; (c) web service; (d) daemon thread; (e) none of these
11. Daemon threads are (a) programs; (b) distributed services; (c) utilities performing services for other threads; (d) error threads; (e) malicious
12. An object with a method declared *synchronized* may have the method execute (a) never; (b) only once; (c) only under one thread at a time; (d) only as it goes out of scope; (e) only with user permission
13. Producer and consumer threads must (a) never coexist; (b) always coexist; (c) be synchronized; (d) be paid; (e) obey each other

2. Software for mobile computing

1. A smart phone is likely to differ most from a laptop computer in (a) processor speed; (b) screen resolution; (c) number of I/O streams; (d) Internet access; (e) access to URLs
2. A robot is likely to differ most from a laptop computer in (a) processor speed; (b) possession of an IP address; (c) number of I/O streams; (d) Internet access; (e) that it lacks an operating system
3. One device that runs concurrent software is a (a) processor; (b) gate; (c) RAM; (d) mobile phone; (e) keyboard
4. One device that runs concurrent software is a (a) processor; (b) gate; (c) RAM; (d) robot; (e) keyboard
5. Desktop GUIs work mostly (a) autonomously; (b) retroactively; (c) randomly; (d) synchronously; (e) asynchronously
6. Mobile devices work mostly (a) autonomously; (b) retroactively; (c) randomly; (d) asynchronously; (e) synchronously

3. Web development and concurrency

1. Code that runs on a browser's Java virtual machine is (a) in HTML files; (b) in *.exe* files; (c) in applets; (d) in JavaScript; (e) a security violation
2. Java applets (a) are compiled to machine language; (b) run on the Java virtual machine; (c) run on the microprocessor; (d) run on servers; (e) are hardware specific
3. An example of a concurrency issue is (a) processor speed; (b) operating system I/O speed; (c) Internet speed; (d) integrity of bank transactions when multiple users have ATM access; (e) user authentication
4. The HTML tag <applet> indicates code that runs on (a) a desktop; (b) a laptop; (c) a smart phone; (d) a browser; (e) fruit juice
5. The HTML tag <applet> indicates code that runs on (a) a desktop; (b) a laptop; (c) a smart phone; (d) the Java virtual machine; (e) fruit juice
6. Interoperability is best supported by (a) the IOS; (b) Windows; (c) XML; (d) Intel machine language; (e) proprietary specifications
7. XML (a) is an alteration to HTML; (b) provides graphics support; (c) is a way to tag web data with semantic information; (d) is an extended machine language; (e) is machine language

4. Ethical and security issues

1. A firewall (a) all access; (b) spam; (c) blocks phishing; (d) filters outside access to computers via the Internet; (e) is hardware
2. Authentication, access privileges, and firewalls help provide (a) reliability; (b) speed; (c) connectivity; (d) security; (e) storage space
3. Encryption (a) enables authentication of user ID; (b) examines packets to block unauthorized access; (c) scrambles data to be unreadable without a special key; (d) requires special hardware; (e) none of these
4. Secure web sites are (a) firewall protected; (b) password protected or encrypted; (c) government protected; (d) accessible via special hardware; (e) unknown
5. Authentication assures that (a) a computer is safe to use; (b) a website is safe; (c) only users with permission gain access; (d) facts at a website are correct; (e) none of these
6. Examining packets to block unauthorized outside access to a computer is done by (a) the user; (b) the network administrator; (c) a firewall; (d) encryption; (e) hacking

7. Encryption (a) provides complete privacy;
   (b) enables government inspection of messages;
   (c) provides security of varying reliability;
   (d) makes messages unreadable by the recipient; (e) all of the above
8. Analyzing and searching databases to find patterns and to enable analysis is (a) hacking; (b) data mining; (c) a privacy violation; (d) a free-speech violation; (e) a free-speech issue
9. Privacy in the electronic era includes (a) accurate information; (b) control of dissemination of personal information; (c) sufficient time alone; (d) freedom from exposure to undesirable ideas; (e) none of these
10. Privacy issues are raised directly by IT due to the (a) existence of data storage media; (b) existence of digital processing; (c) ease of copying and communication; (d) existence of curiosity; (e) none of these
11. Ethical responsibilities of IT professionals include (a) privacy; (b) minimizing cost; (c) maximizing profit; (d) minimizing testing; (e) maximizing testing
12. Ethical responsibilities of IT professionals include (a) limiting risks; (b) minimizing cost; (c) maximizing profit; (d) minimizing testing; (e) maximizing testing

Terminology on Topic 7: Concurrency

concurrency	interrupt	pre-emption	round robin	*step*	*wait*
consumer thread	lock	priority	*Runnable*	suspend	
daemon thread	monitor	producer thread	scheduler	synchronization	
dynamic allocation	multithreading	*ready*	sleep	thread	
garbage collection	parallelism	*resume*	*start*	time slicing	

Problems for Topic 7

Topic objective:
Explain event-driven GUI development and use Java graphics libraries.

7.1a Describe multi-threading

1. What are some examples of *multi-threading* in computing systems today?
2. Describe the Java feature that enables the creation of more than one process occurring at the same time.
3. Name and describe the Java feature that supports background processes like garbage collection.

7.1b Use Java threads†

[Problems to be defined.]

7.2a Describe features of software for mobile computing

1. Describe a kind of input typical of mobile devices but not of desktop computers.
2. In what form does mobile computing encounter problems of *asynchrony*? In what sense are desktop GUIs *synchronous*, by contrast?

7.2b Design and test a multi-threaded robotic program†

[Problems to be defined.]

7.3a Describe issues in web software development

1. How does a web-based application handle the issue of disk storage of user data?
2. Describe how a web site may display the same animation on a Windows machine, a Macintosh, and a mobile phone.
3. Describe issues in web software development.
4. Describe *cloud computing*.

7.3b Write a Java applet†

[Problems to be defined.]

7.4a Describe a network security issue

1. What is *authentication* used for?
2. What specifically do *firewalls* do, and for what purpose?
3. How is unwanted outside access to networked computers blocked?
4. Name and describe a utility that filters incoming packets. For what purpose?
5. Describe some forms of malware.
6. Describe how a secure web site is protected.
7. Explain two technical means by which secure communication of data is assured.
8. What is *encryption* and what is its use?
9. What are some principles of IT security?

7.4b Describe ethical issues for computer professionals*

Discuss in one or two paragraphs one of the following issues, referring to technology related factors and referring to possible arguments on each side.

1. Describe some ethical concerns that apply to IT professionals.
2. To whom or to what do IT professionals have ethical responsibilities? Explain.
3. How are movies on the web different from journals on the web with respect to intellectual property?
4. What is *ethics* and how does it apply to professionals?
5. What is *copyright*, and how does its application differ between electronic and other media?
6. Does the opportunity to broadcast negative messages to a large Internet audience preclude strong legal protection from defamation?
7. What new *privacy* issues are raised by the Internet?

Study questions on multiple topics

1. What have been your experiences with the stages of the software development life cycle?
2. Describe your use of the JDK or of the IDE that you used.
3. Describe your work on the semester project.
4. Please describe your experience with the exercises, the semester project, the classroom discussion, and the group work in this class.