"I really hate this darn machine; I wish that they would sell it.
It won't do what I want it to, but only what I tell it."
- *The Programmer's Lament*

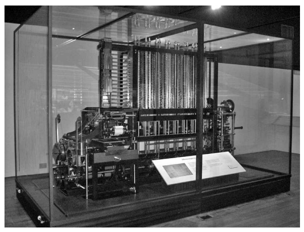
ITI 1120 Introduction to Computing I Class Notes Winter 2008

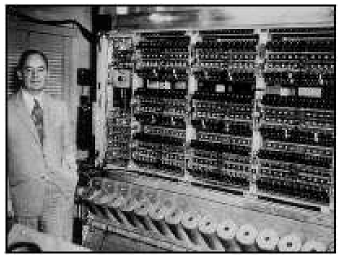
A. Felty

(contributors: D. Amyot, G. Arbez, S. Boyd,
R. Holte, D. Inkpen, W. Li, S. Somé, A. Williams)

Historical note …

- **Charles Babbage**, British mathematician and engineer, designed and built, in 1833, parts of a **machine** that contained modern components such as: central processing unit, memory, and a data input device with punch cards.





- **John von Neumann**, Hungarian mathematician, participated in the development of the **first computer**: ENIAC (1945)
- The principle of the Von Neumann architecture: the data and the programs are encoded in the memory

Software

- This course is about solving problems using computer software.
  - Real-life software can include tens of millions of lines of program code, or it can be just a few lines of code in a life-critical system.
  - Software design teams can range from a single person to over a thousand people.
  - Software can live for decades (example: the SABRE airline reservation system is over 50 years old) and must be maintained to be successful.
  - To produce successful software, a systematic and rigorous development process is needed.
- Software engineering:
  - The process of designing software that functions correctly, and producing the software on time and within a budget.

3

Software Life Cycle

1. Requirements analysis
   - What problem are you trying to solve?
   - What are the needs of the users?
   - What resources are available?
     - Equipment, time, cost, people
   - Develop a plan
2. Design
   - Proposal for the solution of the problem within the constraints of the requirements
   - Model the software system
     - Structure of the software ("architecture")
     - Organization of data
3. Algorithm development
   - Determine the steps required to solve particular problems or sub-problems.

4

Software Life Cycle

4. Implementation
   - Creation of program code:
     - Manually, or semi-automatically with tools.
5. Quality Assurance
   - Verification and Validation: The process of checking that a software system meets specifications and that it fulfills its intended purpose.
   - Testing: Running experiments to see if the software has the expected functionality and performance.
   - Debugging: The process of determining how to modify software to remove problems.

Software Life Cycle

6. Deployment
   - How does the software get to the customer?
   - How is the software installed and configured in the customer's environment?
7. Maintenance
   - As customers report problems, how are the fixes developed, tested, and deployed?
   - How are new features added?
   - How are obsolete features retired

- Documenting is an activity that occurs throughout the cycle.

Problem Analysis

- Our aim is to use a computer to solve problems.

> "Computers are good at following instructions, but not at reading your mind."
> *- D. Knuth*

- Problems are usually stated in a "natural language" (e.g. English), and it is up to you, as a software designer, to extract the exact problem specification from the informal problem statement.

- This involves understanding the problem, and clarifying:
  - What data is "given" or "input" (available for use when the problem-solving takes place)
  - What results are required
  - What assumptions are safe to make
  - What constraints must be obeyed.

7

Example 1: Average of three numbers

- Informal statement:
  - John wants to know the total and average cost of the three books he just bought.

- GIVENS: descriptions and names of the values that are known
  - Num1, Num2, Num3: numbers representing the cost of each of John's books

- RESULTS: descriptions and names of the values to be computed from the givens
  - Sum, the sum of Num1, Num2, Num3
  - Avg, the average of Num1, Num2, Num3

8

What is an Algorithm?

- An algorithm is a sequence of well-defined steps for solving a problem.
- Developing an algorithm for a problem is a creative process.
- Sometimes, part of this process involves problem decomposition, which involves deciding how to break the problem into smaller sub-problems.
- Keep in mind:
  - There can be many algorithms for the same problem, and you may have to choose the most appropriate solution.
  - There are problems for which there is no algorithm that solves the problem.

9

Algorithm models

- Before getting into the details of coding, you should build a model of the algorithm.
- The algorithm model we will use in this course has the following format:

GIVENS
  - A list of the names of given values

RESULTS
  - The name of the result (or a list of results)

HEADER <RESULTS> ← <algorithm name> (<GIVENS>)
  - It specifies the name of the algorithm, an ordered list of the givens, and an ordered list of the results.

BODY
  - A sequence of instructions which, when executed, computes the desired results from the givens.

10

Algorithm for Average

?

Informal statement:

– John wants to know the total and average cost of the three books he just bought.

GIVENS: Num1, Num2, Num3

– (three numbers representing the cost of each of John's books)

RESULTS:

– Sum (the sum of Num1, Num2, and Num3)

– Avg (the average of Num1, Num2, and Num3)

HEADER:

BODY:

11

Data in Algorithm Models

- Literals: constant data values
  - Two types: numeric data, and textual data

- Examples of numeric literals:

  2, 3.14159, –14, –14.0

- A string is a sequence of characters enclosed in double-quotes. The quotes are not part of the data.

- Examples of strings:
  - "Hello", " foo", "bar ", "2", " ", ""

12

Storing values in a computer

- The computer's memory consists of a large, but NOT unlimited, number of storage locations, each of which has an address to identify the location.
- Variables represent values stored in the computer.
- A variable has the following attributes:
  - Name: Used by the computer as the address in the memory.
  - Value: The contents of the storage location.
  - Type: Constraints on the values that can be stored in a variable, or on the operations that can be performed.
- Example: X ← 4, Y ← 3, X ← Y
  - What are the values of X and Y after the execution?

13

Assigning Values to Variables

- To give a value to a variable is called assignment.
  - Example: X ← 3
- In our model notation, we use

  AVariable ← AnExpression

  to denote assigning the value of the expression AnExpression to a variable named AVariable.
- On the right-hand side, AnExpression could be a literal value, a variable (its value is taken), or an expression involving both.
- The names (variables) of the GIVENS of an algorithm are called parameters (or, formal parameters)

14

Expressions

- In our algorithm models, it is permitted to use any mathematical expression to perform calculations on numeric data values and values of variables.
- Examples:

3	$\sqrt{49}$
3 + 5	$\lvert -2 \rvert$
3 + Y × 4	$\log_2(32)$
3 + (X ÷ 2) × Y	
9.3 – 6.2	

15

Operators

- Operators perform some sort of calculation on values.
- The number of values an operator uses may vary:
  - Binary: has two operands
  - Unary: has one operand
  - Examples:
    - Subtraction: a binary operator.
    - Negation: a unary operator.
- Operators must be evaluated in a specific order, and some operators may have precedence over others.
  - For our algorithm models, we will use the order and precedence rules from mathematics.

16

Division and remainders

- It is often useful to perform division in entirely within integers, as opposed to using real numbers.
- Two types of division:

Real:	11.0 / 4.0	result: 2.75
Integer:	11 ÷ 4	result: 2

    - The result of integer division is truncated; the fraction is cut off.
- When doing division in integers, it is also very useful to obtain the remainder from the division. This is done with the mod operator.

  11 mod 4: result: 3
- Finding the remainder has several important uses:
  - The value of X mod Y is always in the range from 0 to Y − 1.
  - The value of X mod 10 gives the last digit of X.

17

Models vs. Computation

- We are using mathematical operators and values as a model for computation.

- A model is only an abstraction that captures the essentials of the problem, but not all the details. When it comes to actual computation, additional constraints may be introduced:
  - Can we use numbers of unlimited range?
  - What about types of values?
  - How fast are the computations performed?
  - Do we have enough memory for all of the variables?
  - Does an operator in a programming language work exactly the same way as in mathematics?

18

Coding

- Coding = translating an algorithm into a particular programming language so it can be executed on a computer.
- Do not be too quick to code! It is much better to spend time on your algorithm, mentally checking it, thinking about it, and "tracing" it on test data to be sure that it is correct.

"The sooner you start to code,
the longer the program will take." - R. Carlson

- Coding is largely a mechanical process, not a creative one.
- Both algorithm development and coding require very careful attention to detail.

Programming Languages

- Inside the computer, the hardware understands only sequences of electrical signals, represented by digits 0 and 1 ("machine language").
  - This language is usually specific to a particular computer processor.
- Since long sequences of zeros are difficult for people to work with, programs are normally created in a programming language, and then translated to machine language.
- Programming languages have a very precise grammar ("syntax") and unambiguous meanings of statements ("semantics").

Types of Programming Languages

- Machine-level languages
  - Numbers. The only language computer understands directly
- Assembly languages
  - Instructions made up of symbolic instruction codes
  - Assembler converts the source code to the machine language
- High-level languages (third generation)
  - Use a series of English-like words to write instructions
- Forth-generation languages:
  - Syntax closer to human language (for databses, e.g., SQL)
  - Provide visual or graphical interface for creating source code (e.g., VisualBasic.Net)

21

Programming Paradigms

High-level languages:

- imperative / procedural programming
  (e.g., Basic, Pascal, Fortran, C)
- object-oriented programming
  (e.g., Java, C++, SmallTalk)
- functional programming (e.g., Lisp, ML)
- logic programming (e.g., Prolog)

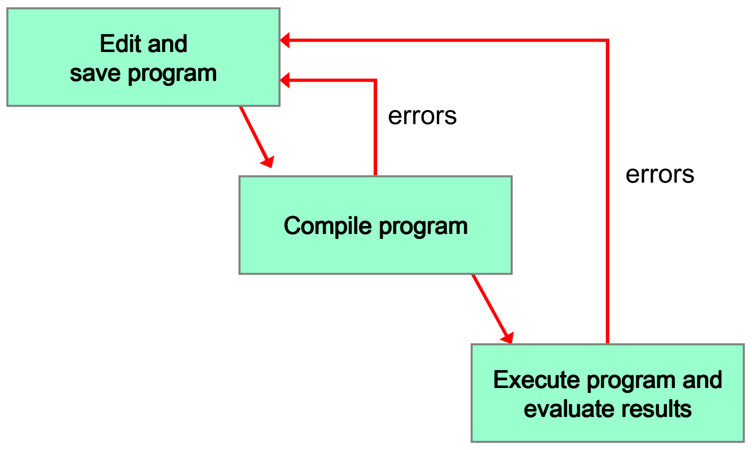
22

Compilers vs. Interpreters

- Compiler = a program that translates source code into machine code. At the end of the compilation the machine code can be executed.
- Interpreter = a program that translates each line of code into machine language and executes it before translating the next line.
- Differences:
  - Execution of compiled code is faster.
  - Compilers can do optimization.
  - Interpreters used for prototyping, multiplatform.

23

Basic Program Development



24

Testing, Debugging, and Maintenance

- Testing = looking for errors ("bugs") in an algorithm or program by executing it on test data (givens) and checking the correctness of the results. Big programs are usually impossible to test completely.
- Debugging = locating and correcting an error in an algorithm or program.
- Maintenance = changing an algorithm or program that is in use, updating, fixing errors.

25

Three Types of Errors

1. Syntax errors: These are illegal combinations of symbols that do not obey the rules of the programming language.
    - Symptom: the program will not compile.
2. Run-time errors: These are errors which result from the data values used.
    - Symptom: the program crashes while running.
3. Logic (semantic) errors: These are errors which result from incorrect reasoning in the program. They probably occur because the algorithm is wrong.
    - Symptom: the program runs, but the results are wrong.

26

Documentation

- Documentation is all of the materials that make software easy to understand for both software designers, and users of the software.

"If you can't explain it simply, you don't understand it well enough."
- A. Einstein

- Internal documentation (such as comments, descriptive variable names) occurs inside the program.
- External documentation (such as models, user manuals, etc.) is documentation which is outside of the program.

27

Another example

- Write an algorithm (ComputeP) that takes as input three values (A, B, C) and returns the proportion of each value out of their total.
- First solution:

```
GIVENS:      A, B, C          (three numbers)
RESULTS:     PA, PB, PC       (three proportions)
HEADER:      (PA,PB,PC)   ComputeP(A,B,C)
BODY:
             1. PA    A / (A+B+C)
             2. PB    B / (A+B+C)
             3. PC    C / (A+B+C)
```

NOTE: The computation of A+B+C is repeated three times.

28

Another example (cont.)

- Second solution:

```
GIVENS:          A, B, C          (three numbers)
INTERMEDIATES:   Sum              (their sum)
RESULTS:         PA, PB, PC       (three proportions)
HEADER:          (PA,PB,PC)   ComputeP(A,B,C)
BODY:
                 1. Sum    A+B+C
                 2. PA   A / Sum
                 3. PB   B / Sum
                 4. PC   C / Sum
```

There is no difference in logic and results, but the algorithm is more clear and there is less computation to execute.

29

Intermediate Variables

- Often it is useful within an algorithm to use a variable that is neither a given nor a result used to temporarily hold a value.
- These INTERMEDIATE variables should be listed and described along with the givens and results at the start of an algorithm definition.
- Their values are not returned to the calling statement, nor are they remembered from one call to the next.
- Intermediate variables are used mainly for improving readability of the algorithm or for the efficiency of the algorithm.

30

More on INTERMEDIATES

- INTERMEDIATES have values that can vary.
  - They may vary depending on the problem instance (in other words, depending on the values of the GIVENS).
  - They may change during computation.

- INTERMEDIATES that do not vary are called CONSTANTS.
  - Their values are fixed.
    - They are the same no matter what the values of the GIVENS are.
    - Their values will not change during the computation.
  - They can be given names that help document the algorithm and make it more readable.
  - Representing constants by names reduces maintenance effort.

31

Problem: Average out of 100

?

- Write an algorithm that takes three scores (each out of 25) and computes their average (out of 100).
  - (Idea: average the scores, then convert the result to be out of 100.)

32

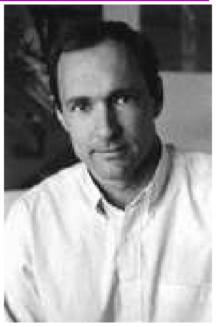
Summary of Variables

- GIVENS are the input data. They vary from one call to another. In other words, their values can be different for each problem instance.
- RESULTS are the answers that are generated by an algorithm.
- INTERMEDIATES are everything else.

33

Historical note …

- In 1990, **Tim Berners-Lee**, researcher at CERN in Geneva, developed the *World Wide Web* technology, in order to facilitate information access on the Internet.
- He initiated many standards, among which the most utilised are HTTP, URL, and the HTML language.
- Now he is working on the Semantic Web



In 1992, at Sun Microsystems, **James Gosling** **(born in Canada)** and his team invented, the programming language Oak, renamed Java in 1994.

34

Historical note …

- Ada Byron, countess of Lovelace, mathematician and collaborator of C. Babbage, defined the principle of successive iterations in the execution of an operation (1840).
- She created the first computer algorithm; she is considered the first programmer.
- There is a programming language named after her: Ada

35

Invoking an Algorithm

- To invoke an algorithm you use a "call" statement which is identical to the header except for the names of the givens and results.

  ```
  TheAvg  ← Average(10, 7, 4)
  ```

  invokes our algorithm with givens Num1=10, Num2=7, Num3=4 and returns the results in TheAvg (average).

- Information is passed between the call statement and the algorithm based on the ORDER of the givens and results, not their names.

36

Tracing an Algorithm

- To TRACE an algorithm is to execute it by hand, one statement at a time, keeping track of the value of each variable. The aim is either to see what results the algorithm produces or to locate "bugs" in the algorithm.

- Tracing always involves a single problem instance. You may need to do several traces (with different givens) of the same algorithm to find bugs.

37

Tracing Steps

1. Number every statement in the algorithm.

2. Make a table. The first column will say which statement is being executed. The other columns each correspond to a variable. There should be one column for each given, result, and intermediate.

3. Fill in the first row of the table with the variables' initial values.
   - The givens will have the values supplied by the calling statement; all other variables will have a "?".
   - Write "initial values" in column 1 of row 1.

38

Tracing Steps (continued)

4. The second row is for the first statement executed. Put its number in column 1, and for every variable whose value is changed by this statement put the new value in the variable's column. Leave the other columns blank.

5. Proceed to the next statement, make a row for it, and continue until you reach the end of the algorithm.

- The values across a row represent the "state" of the system at a particular point in time.

39

Tracing Example

?

AvgPct ← MarkResult(18, 23, 19)

40

Modified Parameters

- Some problems require the value of a given variable to be changed. The variable is called a MODIFIED parameter and is listed among the GIVENS and put in the header like a GIVEN, but it is also described in a list of MODIFIEDS separate from normal RESULTS.

- Example: Suppose we want to write an algorithm that swaps the values of two variables .
    - e.g. given X=7 and Y=3, the algorithm would exchange the values, resulting in X=3 and Y=7.

41

Problem: Swap two values

?

GIVENS:

RESULTS:

MODIFIEDS:

INTERMEDIATES:

HEADER:

BODY:

42

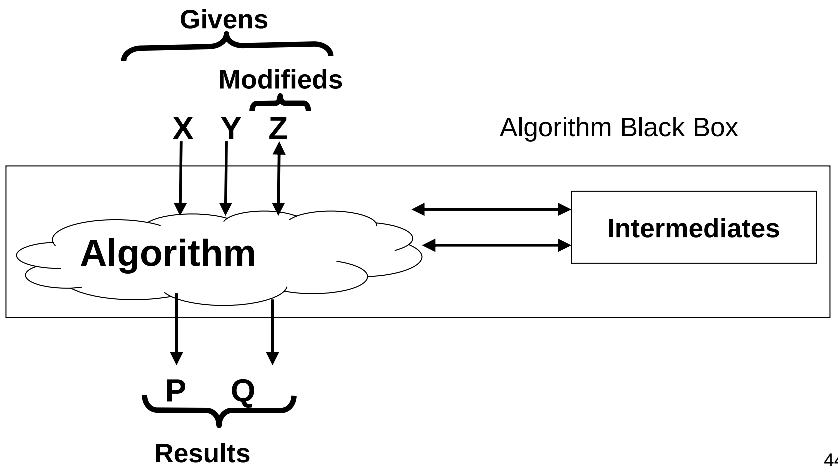
Augmented Algorithm Template

```
GIVENS:
RESULTS:
MODIFIEDS:
INTERMEDIATES:
HEADER
  (List of Results) ← Algorithm Name (List of Givens)
BODY
  statement
  statement
     :
  statement
```

43

Modified Parameters

(P,Q) ← algorithm(X,Y,Z)



44

Using Algorithms

- When developing an algorithm, it is a good idea to make as much use as possible of existing algorithms (ones you have written or that are available in a library).
- You can put a CALL statement to any existing algorithm wherever it is needed in the algorithm you are developing.
- Be sure you get the ORDER of the givens, modifieds, and results correct.
- To call an algorithm you need to know its header but not how it works – you must just trust that it works correctly.
- The values passed to an algorithm is called the arguments (or actual parameters). The arguments (actual parameters) match the parameters (formal parameters) according to their orders

45

Marks out of 100, again

?

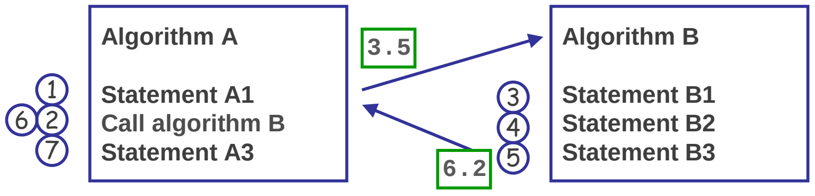
- Redo the "marks out of 100" problem, but this time use the algorithm developed for the "average" problem:

  Avg ← Average(Num1, Num2, Num3)

46

Algorithm calls

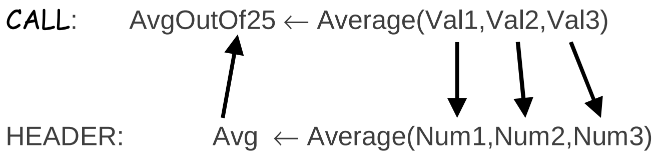
- When one algorithm calls another:
  - The algorithm that is currently executing stops and waits at the point of the call.
  - Values may be "passed" to the called algorithm.
  - The called algorithm executes.
  - When the called algorithm finishes, result values may be passed back to the calling algorithm.
  - The calling method restarts and continues



47

Information Passing

- When calling an algorithm the call statement and the header of the called algorithm must be identical except for the names of the givens, modifieds, and results. These are matched one-to-one in the order they appear


CALL: AvgOutOf25 ← Average(Val1,Val2,Val3)
HEADER: Avg ← Average(Num1,Num2,Num3)

- The arrows show how information is passed.

48

Tracing a Call

- When tracing an algorithm, every time it calls another algorithm whose body is known you must produce a trace of the called algorithm on the givens provided by the call. The trace for each call of each algorithm should be done on a separate page. Each trace should say where it was invoked (“called from page X”).
- When the statement being executed is a call you put more in column 1 of the trace table than just the statement number - you say where to find the trace of the called algorithm (“see page Y”), and you put in a complete picture of the information passing. Just as on the previous slide, write the call statement directly above the header of the algorithm being called and draw arrows showing how the information is passed.
- Values (results, modifieds) that are passed back when the called algorithm terminates should be written in the columns for the variables in the results part of the call.

49

Tracing Example (page 1)

?

- Trace: AvgPct ← MarkResult(23, 16, 21)

50

Tracing Example (page 2)

51

Problem: Reverse Digits

- Given a 2-digit positive number, N, reverse its digits to obtain a new number, ReverseN.
- Assume there is available an algorithm with the header

  (High, Low) $\leftarrow$ Digits(X)

  which returns the left (High) and right (Low) digits of a given 2-digit number X.

52

Trace

?

53

Problem: Join four numbers

?

- Write an algorithm that takes 4 positive integers and joins them into one,
  - e.g., given 11, 35, 200, and 7 it should produce 11352007.
- Trace your algorithm on these givens.
- You may assume there is available an algorithm:

C ← Join(A, B)

Givens:	A, B, two positive integers
Result:	C is the number having the digits in A followed by the digits in B.

- Example: Join(120, 43) produces 12043

54

Variable scope and duration

- Scope: Where you can use a variable's name.
  - General rule: Variables can be only be accessed inside their own algorithm.
  - If you use the name X in two algorithms, the names represent two different values in each algorithm.

- Duration: The "lifetime" of a variable's value.
  - When an algorithm finishes execution, the values of all variables are forgotten.
    - The values of the RESULTS will be passed back to the calling algorithm.
  - When you call an algorithm again, new values are used for all variables.

55

Algorithm Refinement

- If an algorithm with a complex block nested inside another block, make the inner block as a separate algorithm, called a helper algorithm.

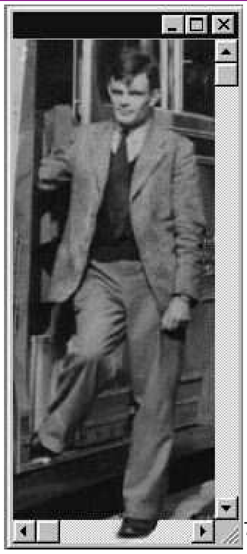
- Then the former algorithm "calls" the latter algorithm.

- In this way, we can keep algorithms simpler, shorter and clearer.

56

Historical note …

- Alan M.Turing, British mathematician and father of modern computer science, designed, in 1936, a logic machine able to solve **all** the problems that can be formulated in algorithms for the modern computers: the Turing machine.
- He also proposed the Turing test, for artificial intelligence.
- The highest distinction in computer science now, awarded by ACM, bears his name: the Turing Award.



57

Translating to Code

- Our approach to programming is to first develop and test algorithms using a model and then TRANSLATE them into code in a programming language.
- Translating algorithms into code is very much a mechanical process, involving only a few decisions.
- For each type of algorithm block we will see one way of translating it to code. Algorithms are then translated block by block.

58

Translating an Algorithm to a Program with a single class and a single method

- A program consists of a single Java class with a single method `main` that is the translation of the algorithm.
- Givens, Results, Modifieds, and Intermediates all get translated to VARIABLES.
- They all must be declared and given a type. Their descriptions are put in the program as comments (called the DATA DICTIONARY).
- Initial values for Givens and Modifieds will be read in from the keyboard.
- Final values for Modifieds and Results will be printed out.

Translation Example

- Algorithm to translate:

```
GIVENS:         Num1, Num2, Num3      (three numbers)
RESULT:         Avg                   (the average of Num1, Num2,
                                      Num3)
INTERMEDIATE:
                Sum                   (the sum of Num1, Num2,
                                       Num3)
HEADER:
                Avg ← Average(Num1, Num2, Num3)
BODY
                Sum ← Num1 + Num2 + Num3
                Avg ← Sum / 3
```

Translated to a Program (Scanner class)

```
// PROGRAM Average--reads 3 real numbers and computes their average.
import java.util.Scanner;
class Average
{
   public static void main (String[] args)
   {
      // SET UP KEYBOARD INPUT
      Scanner keyboard = new Scanner( System.in );
      // DECLARE VARIABLES/DATA DICTIONARY
      double num, num2, num3 ;  // Given numbers
      double sum ;     // Intermediate, sum of num1, num2, and num3
      double avg ;     // Result, average of num1, num2, and num3
      // READ IN GIVENS
      System.out.println ("Please enter 3 real values: ");
      num1 = keyboard.nextDouble( );
      num2 = keyboard.nextDouble( );
      num3 = keyboard.nextDouble( );
      // BODY OF ALGORITHM
      sum = num1 + num2 + num3;
      avg = sum / 3.0;
      // PRINT OUT RESULTS
      System.out.println("The average is " + avg);
  }
}
```

Translated to a Program (ITI1120 class)

```
// PROGRAM Average--reads 3 real numbers and computes their average.
// The file ITI1120.java has to be in the same directory

class Average
{
   public static void main (String[] args)
   {
      // DECLARE VARIABLES/DATA DICTIONARY
      double num1,num2, num3 ;  // Given numbers
      double sum ;     // Intermediate, sum of num1, num2, and num3
      double avg ;     // Result, average of num1, num2, and num3
      // READ IN GIVENS
      System.out.println ("Please enter 3 real values: ");
      num1 = ITI1120.readDouble( );
      num2 = ITI1120.readDouble( );
      num3 = ITI1120.readDouble( );
      // BODY OF ALGORITHM
      sum = num1 + num2 + num3;
      avg = sum / 3.0;
      // PRINT OUT RESULTS
      System.out.println("The average is " + avg);
  }
}
```

Translating Statements to Java

- Assignment statement
  - Model: X ← expression
  - Java: **`X = expression ;`**
- Call statement – we'll see later
- Translating algorithms to Java methods (other than the main method) – we'll see later

Tracing Java Programs

- When you are tracing an actual program, a useful tool is a "debugger". This tool can perform a trace on a running program.
  - Often this requires telling the compiler to add extra information to help the debugger.
- Two modes of operations:
  - Run up to a breakpoint: the program will run at full speed up to a pre-defined point in the source code, and then stop.
  - "Single step" mode: the program will execute one statement and stop.
- When the program stops, you can inspect the current values of variables
  - You can check if the variables' values correspond to the equivalent row of a manual trace.

Using a debugger

- Many debuggers come with four options for execution of a stopped program.
  - Step Into: If the next statement involves a call to another algorithm, execution will go to the first statement of the algorithm body and stop.
  - Step Over: If the next statement involves a call to another algorithm, the other algorithm will be completely executed, and debugger will stop at the next line in the current algorithm.
  - Step Out: Execute all statements up to the end of the current algorithm.
  - Resume: Begin normal execution from the next statement.

Historical note …

- 1945: An insect in the circuits blocked the computer Mark I. The computer scientist **Grace Murray Hopper** decided to call any program malfunction « bug »!
- 1951: Invented the first compiler (A0) for generating machine code from a program source code.
- She was one of the main creators of one of the first programming languages: COBOL.

Started Cosine Tape (Sine check)
Started Multy Adder Test.
Relay #70 Panel F (moth) in relay.
First actual case of bug being found.
closed down.

Branching Control Structure

- So far in the bodies of our algorithms, we have used:
  - a simple statement
  - a straight sequence of simple statements
- Sometimes, we need more than a straight sequence in our solutions, as we sometimes need to do different computations depending on certain conditions.

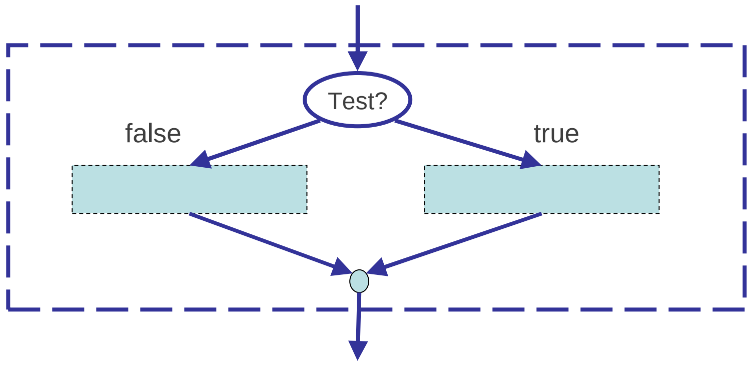
Problem: Larger of Two Numbers

- Write an algorithm to compute the larger of two given numbers.

GIVENS:	X, Y	(two numbers)
RESULT:	M	(the larger of X and Y)
HEADER:	M $\leftarrow$ Max2(X, Y)	

- We will represent this with a Branching Control Structure

Branching Control Structure

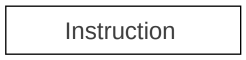


- The shaded boxes are BLOCKS.
  - There are several types of blocks, any type can be put in any box.

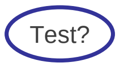
69

Algorithm Model Diagrams

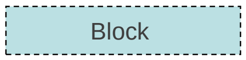
- Provides visual description of algorithms.
- Composed of nodes connected with arrows.
- Instruction block:
  - Represents a simple instruction or a sequence of simple instructions



- Test node:
  - Represents the testing of a condition (Boolean expression with question mark)



- Block node:
  - Indicates where another block node (of any block type) can be inserted



70

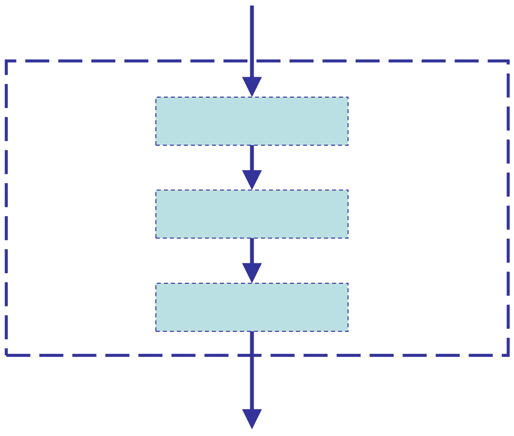
Types of Blocks

- Simple statement (call, assignment)
- Empty statement (Ø = "do nothing")
- Branching control structure
- Loop control structure (coming soon…)
- Sequential control structure (next…)

- Important: Each block has exactly one entrance (one arrow in) and one exit (one arrow out).

71

Diagram of a Sequential Block



72

Back to the Larger of Two Numbers ?

- How would the following algorithm for finding the larger (Max) of two values X and Y be written in a model diagram ?
  - Set M to X
  - If Y is bigger than M, set M to Y

73

Problem: Maximum of 3 numbers ?

- Given three numbers X, Y, and Z, find the maximum of the three values.
  - Version 1: nested tests
  - Version 2: sequence of tests

74

Tracing algorithms with branching

- When tracing an algorithm with tests (branches or loops) number the tests as well as the statements and include a row in the trace indicating which test is being done and whether it is true or false.

75

Trace of Maximum of 3 numbers

?

Trace: MAX3(5, 11, 8)

Do both versions.

76

Problem: Movie Tickets

?

- Calculate the amount to charge for a person's movie ticket given that the charge is \$7 for a person 16 or under, \$5 for a person 65 or older, and \$10 for anyone else.
  - Version 1: nested tests
  - Version 2: sequence of tests

77

Boolean Variables

- A Boolean variable is one which can have only 2 possible values: TRUE or FALSE. (These are not numbers.)

- An assignment statement is used to put a value into a Boolean variable, e.g.,

```
X ←TRUE
Y ← FALSE
```

- The outcome of a test (Boolean expression) can be assigned to a Boolean variable:

```
X ← (A < 0)
```

78

Problem: Positive Value ?

- Write an algorithm which checks if a given number X is positive.

79

Compound Boolean Expressions ?

- A compound Boolean expression consists of two or more Boolean expressions connected by operators AND and/or OR.

- Example: Write a compound Boolean expression that is true if a given age is between 16 and 65 (not including 16 or 65) and false otherwise.

80

Truth Tables

?

- A TRUTH TABLE for a compound Boolean expression shows the results for all possible combinations of the simple expressions:

X	Y	X AND Y	X OR Y
TRUE	TRUE		
TRUE	FALSE		
FALSE	TRUE		
FALSE	FALSE		

81

Operator NOT

X	NOT X
TRUE	FALSE
FALSE	TRUE

- NOT is an operator to negate the value of a simple or compound Boolean expression:
- Example. Suppose Age = 15. Then:
  - Expression Age > 16 has a value FALSE, and NOT (Age > 16) has a value TRUE.
  - Expression Age < 65 has a value TRUE, and NOT (Age < 65) has a value FALSE.

82

Examples of Compound Boolean Expressions

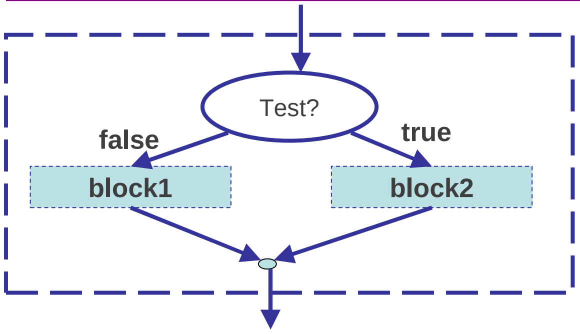
?

Suppose X = 5 and Y = 10.

Expression	Value
(X > 0) AND (NOT (Y = 0))	
(X > 0) AND ((X < Y) OR (Y = 0))	
(NOT (X > 0)) OR ((X < Y) AND (Y = 0))	
NOT ((X > 0) OR ((X < Y) AND (Y = 0)))	

83

Translating Branches to Java



Java :
```
if (Test)
{
    block2
}
else
{
    block1
}
```

- Empty statement
  Java: semicolon followed by comment

```
; // do nothing
```

84

Example

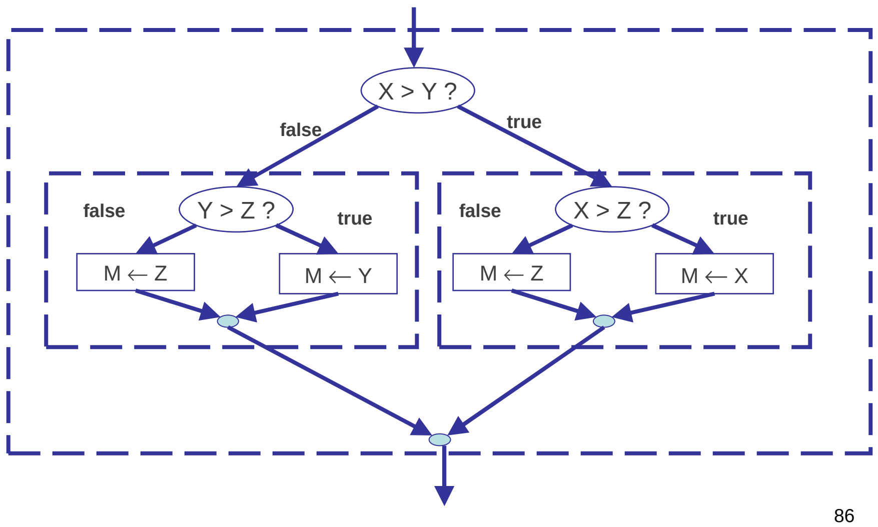
?

Givens:	X, Y, Z	(three numbers)
Result:	M	(the largest given value)
Header:	M←Max3(X, Y, Z)	

- Two solutions:
  - sequence of branch structures
  - nested branch structures
- Translate the latter into Java:

85

Nested Branches



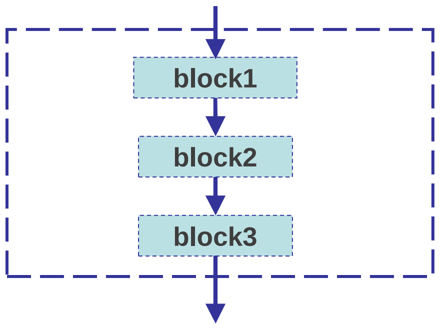
86

Translation

?

87

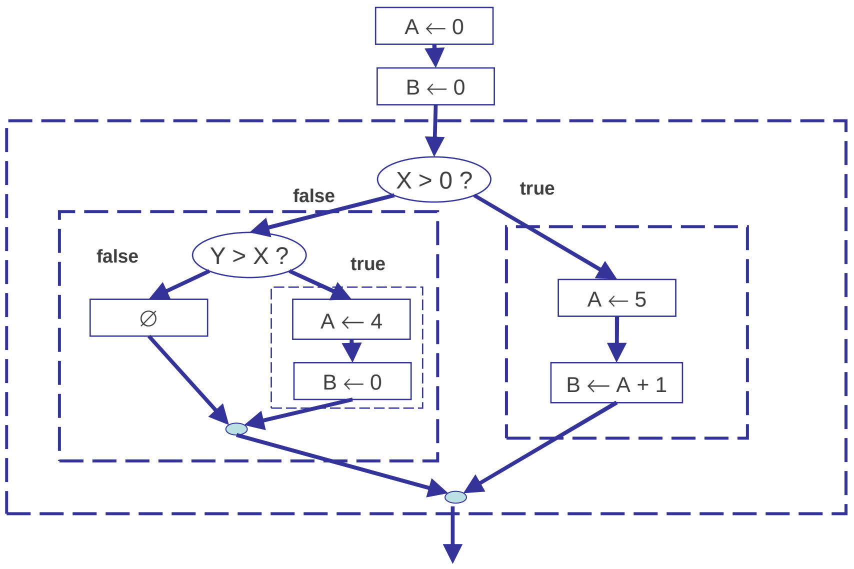
Translating Sequences



Java:
```
{
    block1
    block2
    block3
}
```

88

Example



89

Translation

?

90

Expressions in Tests

- The TEST in a Branch or Loop may be any Boolean expression:
  - Boolean variable
  - Negation of a Boolean expression
    - NOT (Java: **`!`**)
  - Comparison between two values
    - Java operators: **`==  !=  <  >  <=  >=`**
    - The data being compared may not necessarily be **`boolean`**, but the result of the comparison is **`boolean`**
  - Join two Boolean expressions
    - AND (Java: **`&&`**)
    - OR (Java: **`||`**)
- Watch out for
  - confusing = with ==
  - confusing AND with OR
- e.g. test if x is in the range 12..20:

```
(x >= 12) && (x <= 20)
```

91

Historical note …

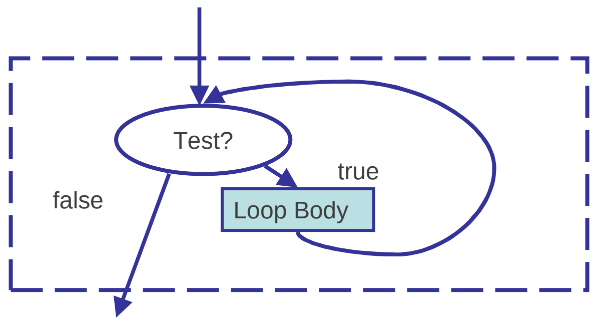
- **Donald Knuth**, American computer scientist, a pioneer of the domain of algorithm analysis.
- He is the author of the very respected book *The Art of Computer Programming* and of the scientific text editor TeX.

- **Edsger Dijkstra**, computer scientist from Netherlands, developed the algorithm of the shortest path (that bears his name) and the concept of sentinel for programming and for parallel processing.
- His 1968 article, "Go To Statement Considered Harmful" revolutionized the utilization of the instruction GOTO to the profit of the control structures such as the while loop.

92

The Loop Block

- Sometimes we need to repeat a set of statements. In our algorithm diagrams, we use a loop block:



- The body of the loop is itself a block (or set of blocks). It is repeated over and over until the test becomes false.

93

Designing a loop block

1. Initialization
   - Are there any variables to initialize?
   - These variables will be updated in the loop.

2. Test condition
   - A condition to determine whether or not to repeat the loop body

3. Loop body
   - What are the steps to repeat?

94

Examples of “counting” loops

- Write an algorithm to find the sum of the numbers 1…N (1+2+…+N).

- Write an algorithm to find the factorial of a number N, denoted as N!
  - Definition of factorial:

    $N! = 1 \times 2 \times \ldots \times N$

95

Arrays

- “Simple” variables hold one value.
- An array has many positions, each able to hold one value.
- If array A has 10 positions, we refer to them using the integers 0-9, called indices or subscripts.
  - e.g. A[2] is the THIRD position with index 2.
- Since we might only have put values into some of A’s positions, when we pass A to an algorithm we’ll normally also have to pass one or more variables saying which positions have values.
  - e.g. if the values are in the first 5 positions, we might pass 5 in a variable ALength (or 4, since the positions are indexed 0-4).

96

Array Indexing

- The index (subscript) of an array of length L may be any integer expression that returns a value in the range 0...(L-1).

"Should array indices start at 0 or 1? My compromise of 0.5 was rejected without, I thought, proper consideration."
-- Stan Kelly-Bootle

  - Suppose K =2, and A (length 4) is

2	-1	5	7

A[2] =
A[K] =
A[2*K-1] =
A[A[0]+1] =

- A[expression] is just like any ordinary variable and can be used anywhere an ordinary variable can be used.

Problem: Value in Middle of an Array

- Write an algorithm that returns the value in the middle of an array A containing N numbers, where N is odd.

Problem: Swap Values in an Array ?

- Write an algorithm that swaps the values in positions I and J of array A.

99

Loop and Array Examples

? 1. Find the sum of the numbers 1…N (1+2+…+N).

? 2. Find the sum of the values in an array containing N values.

3. Given a value T and an array X containing N values, check if the sum of X's values exceeds T.
   - ? Use algorithm from Example 2.
   - ? Efficient version which exits as soon as the sum exceeds T.

? 4. Count how many times K occurs in an array containing N values.

5. Given an array X of N values and a number K, see if K occurs in X or not.
   - ? Use algorithm from Example 4.
   - ? Efficient version which exits as soon as K is found.

100

More Loop and Array Examples

6. Given an array X of N values and a number K, find the position of the first occurrence of K. (If K does not occur, return –1 as the position.)

7. Find the maximum value in an array containing N values.

8. Find the position of the first occurrence of the maximum value in an array containing N values.
   - Use algorithm from Example 7.
   - Use algorithms from any examples.
   - Version using one loop and no other algorithms.

9. Check if an array of N values contains any duplicates.
   - Strategies?

Creating an Array

- Create an array containing the integers 1 to N in reverse order.
- Assume there is an algorithm

  A ← MakeNewArray(L)

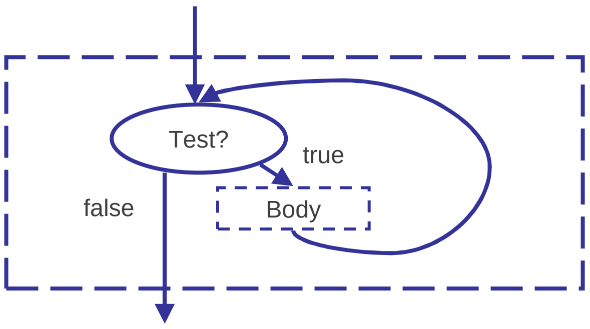
  that creates an array A having L positions with no values in them.

Trace for N=3

?

103

Translating Loops to Java



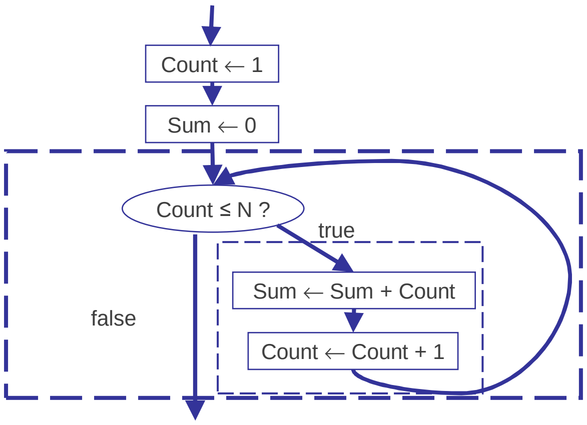
- Java

```
while (Test)
{
    Body
}
```

104

Algorithm: Sum from 1 to N

GIVEN:	N	(a positive integer)
INTERMEDIATE:	Count	(index going from 1 to N)
RESULT:		Sum (sum of integers 1 to N)
HEADER:		Sum ← Sum1ToN(N)
BODY:		



Translate to Program

?

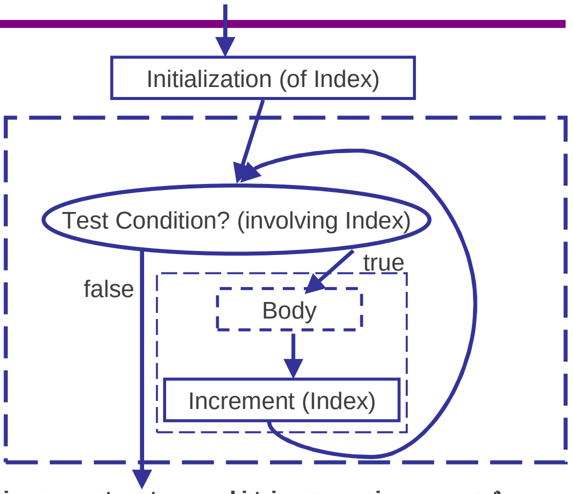
```
import java.io.*;

class
{
   public static void main (String args[ ])
   {

   }
}
```

The FOR loop diagram

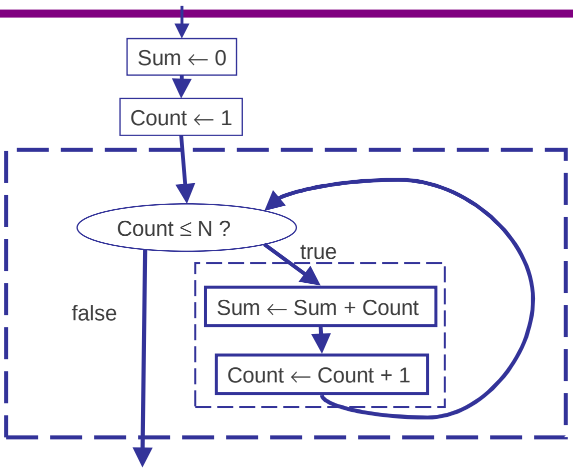
Any loop diagram can be translated using WHILE, but the looping pattern to the right is best translated into a FOR loop.



```
for (<initialization>; <test_condition>; <increment>)
{
   Body // excluding Increment Index
}
```

FOR loop to add 1 to N

?



Translate to Java using a FOR loop:

FOR Loop in Java: Summary

- Any FOR loop can always be formed as a WHILE loop
  - It does not give us any extra capability.
  - However, the notation is often more convenient.

- The FOR loop is usually used when we know how many times the loop body is to be executed.

- The FOR loop:

```
for (<initialization>; <test_condition>; <increment>)
{
   // body
}
```

- In most cases, the initialization part initializes a counter, the test condition tests if the counter is within the limit, and the increment part modifies the counter.

Arrays in Java

- An array variable is declared with the type of the members.
  - For instance, the following is a declaration of a variable of an array with members of the type double:
    `double[] anArray;`
- When an array variable is declared, the array is NOT created. What we have is only a name of an array.
  - `anArray` will have the special value `null` until it is created.

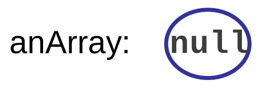
Creating an array

- To create the array, operator `new` is used.
- We must provide the number of members in the array, and the type of the members. These cannot be changed later.
```
double[] anArray;
anArray = new double[5];
```
  or, combining the declaration and the array creation:
```
double[] anArray = new double[5];
```
- When an array is created, the number of positions available in the array can be accessed using a field called length with the dot operator. For instance, `anArray.length`has a value 5.

111

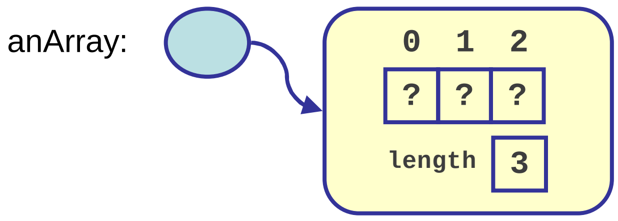
Memory for Arrays

```
double[] anArray ;
```



"Thou shalt not follow the NULL pointer, for chaos and madness await thee at its end."
– H. Spencer

```
anArray = new double[3]
```



112

Accessing array members in Java

- Array members are accessed by indices using the subscript operator `[]`. The indices are integers starting from 0. (This is the same as in our algorithm models.)
- For instance, if `anArray` is an array of three integers, then:
  - the first member is `anArray[0]`
  - the second member is `anArray[1]`,
  - and the third member is `anArray[2]`.
- The indices can be any expression that has an integer value.

  If an index is out of range, i.e., less than `0` or greater than `length-1`, a run-time error occurs.

Initializing array members

- Array members can be initialized individually using the indices and the subscript operator.

```
int [] intArray = new int[3];
intArray[0] = 3;
intArray[1] = 5;
intArray[2] = 4;
```

- Array members may also be initialized when the array is created:

```
int [] intArray;
intArray = new int [] { 3, 5, 4 };
```

Partial initialization of an Array

- An array may be partially initialized.

```
int [] intArray = new int [5];
intArray[0] = 3;
intArray[1] = 5;
intArray[2] = 4;
```

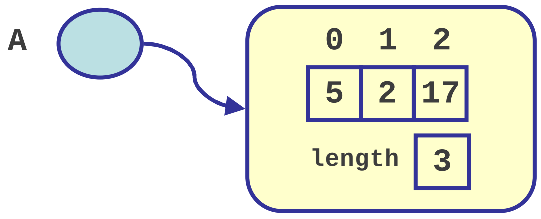
  - In this case, `intArray[3]` and `intArray[4]` are undefined.

- When an array is processed, we may need another variable (or variables) to keep track of the indices for which we have assigned values.

115

Reference Types

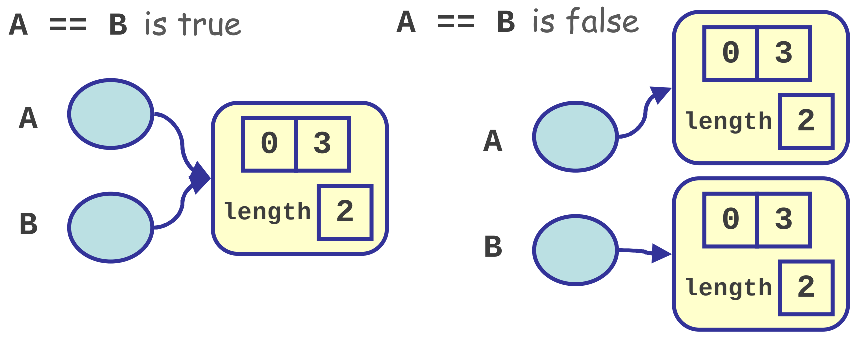
- An array type is a reference type, because of the “pointer” to the array.
- It is important to distinguish the reference (pointer) from the “item being pointed to”.
  - In the diagram below, **A** is the reference, and the array is what is being pointed to.
    - Java does not allow us to peek inside **A** to see what is in the pointer.



116

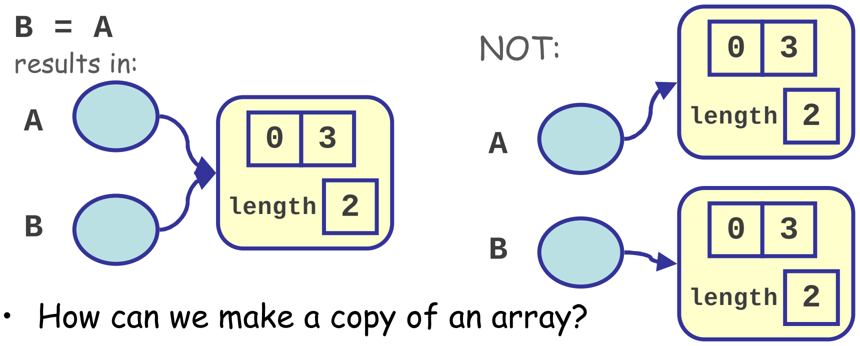
Reference Types

- What happens with assignment and comparison of reference types?
  - It is the references that are compared or assigned, not the arrays.



Reference Types

- Assignment only copies a reference, not the object to which is points.

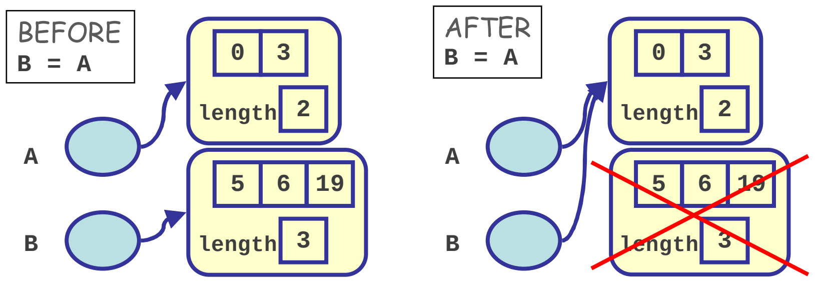


- How can we make a copy of an array?

118

Lost references

- With reference types, be careful that you don't "lose" an object to which a reference points.



- After the assignment, there is no reference to the second array. The second array will be forgotten by Java and cannot be recovered.

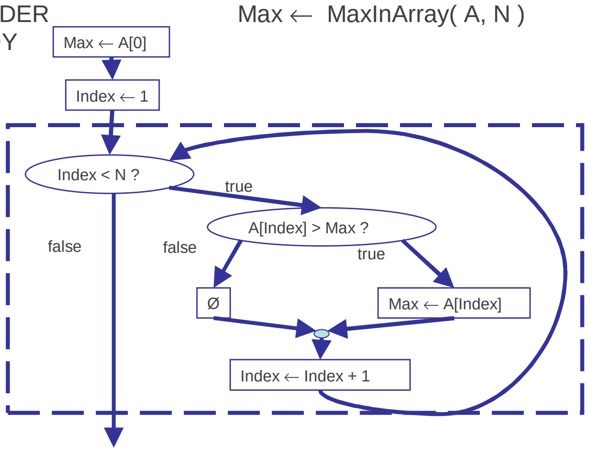
"Objects can be classified scientifically into three major categories: those that don't work, those that break down and those that get lost." – R. Baker

119

Algorithm: Maximum in Array

GIVENS:	N	(a positive integer)
	A	(array containing N values)
INTERMEDIATE:	Index	(indices for A)
RESULT:	Max	(maximum member of A)
HEADER	Max ←	MaxInArray(A, N)

BODY



120

Translated to a Program

?

```
import java.io.* ;
class
{
   public static void main (String args[ ]) throws IOException
   {
```

Code to read givens (next slide) goes here.

```
    }
}
```

Code to Read Givens

Option 1: ITI1120.readDoubleLine

Code to Read Givens

Option 2: read the values one by one

123

Historical note …

- 1976: Steve Jobs and Steve Wozniak created the first personal computer called Apple I.
- The computer sold for 666.66 $; it had 256 bytes of ROM, 4 K bytes of RAM and video output on the television set.

- In June 1975, Bill Gates and Paul Allen renamed their company Traf-O-Data into **Microsoft**.
- Produced MS-DOS, **Windows**, Basic-Microsoft, and later **Visual Basic**.

124

Translating an Algorithm to a Method

- Most algorithms are translated into METHODS, not programs, so they can be called by other algorithms.
- The program starts with a **`main`** method, which may read in some values and output the result. Method **`main`** calls one or more methods, each of which implements an algorithm.
  - In this way, method **`main`** acts as a dispatcher.

125

Java Methods

- Every method in Java has
  - a return type, a name, a parameter list, a body
- Return type: It specifies what is the type of the result of the method. If there is no result, this type is **`void`**. Final values of RESULTS and MODIFIEDS are passed back to the calling method, not printed out.
- Name: The same as the name of an algorithm.
- Parameter list: It specifies the name and the type of the parameters, in order. Initial values for GIVENS and MODIFIEDS are passed in as parameters.
- Body: Translation of the body of an algorithm exactly as before. INTERMEDIATES become local variables.

126

Differences in Methods between Java and Algorithm Models

- A Java method may return zero or one value.
  - It is not possible to return more than one value in Java, as can be done with our algorithm models. However...
    - This value may be a primitive type or a reference type. For example, a method may return an array.

- Java does not allow parameters of a primitive type to be MODIFIED.
  - Only reference types created with **new** can be modified.

127

Example

GIVENS:	A, B, C	(three values)
RESULT:	Avg	(average of A, B, C)
INTERMEDIATE:	Sum	(sum of A, B, C)
HEADER	Avg ←Avg3(A,B,C)	
BODY	Sum ← A + B + C	
	Avg ← Sum / 3	

```
// METHOD avg3--Finds the average of 3 values
// a,b,c are 3 given numbers

public static double avg3(int a, int b, int c)
{  // DECLARE VARIABLE/DATA DICTIONARY
   int sum ;          // Intermediate, sum of a, b, and c
   double avg ;              // result, average of a, b, c
   // BODY OF ALGORITHM
   sum = a + b + c ;
   avg = sum / 3.0 ;
   // RETURN RESULT
   return avg ;
}
```

128

Method Template

```
// METHOD Name: Short description of what the method does,
// plus a description of the variables in the parameter list
```

Return type | Method name | Parameter list

```
public static double avg3(int a, int b, int c)
{
   // DECLARE VARIABLES/DATA DICTIONARY
   // intermediates and the result, if you give it a name

   // BODY OF ALGORITHM

   // RETURN RESULT
   return (a+b+c)/3.0 ;
}
```

expression
(value to return)

Method Accessibility

- If a method is **`public`**, it can be called from anywhere in a program.
- If a method is **`private`**, it can only be called from inside the class where it is defined.
- There are two other levels of access: **`protected`** and “package”, that are between public and private. We will not use them in this course.

The CALL statement

Model: X ← Avg3(10, J, K)
Java: **`X = avg3(10, J, K ) ;`**

- Here, **`avg3(10, J, K)`**is a method call statement. The method call statement evaluates to the value returned from the method, which can be assigned to a variable or used in any expression (of the right type).
  - Example:
    `Y = 10.2 * avg3( A, B, C ) + avg3( P, Q, -11 )`

131

A method with return type **`void`**

- If a method does not return a value, i.e., return type **`void`**, then a call activates the method to do whatever it is supposed to do.

```
public static void print3(int x, int y, int z)
{
   System.out.println("This method does not return anyvalue.");
   System.out.println( x );
   System.out.println( y );
   System.out.println( z );
}
```

- When the method is called by
  `print3( 3, 5, 7 );`
  it simply prints these arguments to the screen.

132

Program with Two Methods (p. 1)

```
// PROGRAM MethodExample --- This program illustrates a
//    main method that uses (calls, invokes) the method avg3 twice

class MethodExample
{
   public static void main (String[ ] args)
   {
      // DECLARE VARIABLES / DATA DICTIONARY
      int b,y;                   // intermediates
      double z, c;               // intermediates

      // BODY OF ALGORITHM
      b = 15;
      y = 7;
      z = avg3( b, -4, y );      // FIRST CALL
      c = avg3( 9, y, 2 + b );   // SECOND CALL

      // PRINT OUT RESULTS
      System.out.println("The sum of the averages is:  " +(z + c));
   }  // end of main

   // continue on the next page
```

Program with Two Methods (p. 2)

```
// continue from previous page

   // METHOD avg3 --Finds the average of 3 doubles.
   //      a,b,c are the 3 given integers.
   private static double avg3(int a, int b, int c)
   {
      // DECLARE VARIABLES / DATA DICTIONARY
      int sum;      // Intermediate, sum of a, b and c
      double avg;   // Result, average of a, b and c

      // BODY OF ALGORITHM
      sum = a + b + c;
      avg = sum / 3.0;

      // RETURN RESULT
      return avg;
    }
} // end of class
```

When a CALL is made ...

1. Execution of the calling method is suspended.
2. Memory is allocated for parameters and local variables of primitive types (**`int`**, **`double`**, **`char`**, **`boolean`**) in the called method.
3. Initial values for parameters of primitive types are COPIED from the corresponding arguments of the call.
4. Parameters of reference types are associated to the arrays or objects that the corresponding arguments refer to.
5. Execution of method body begins.

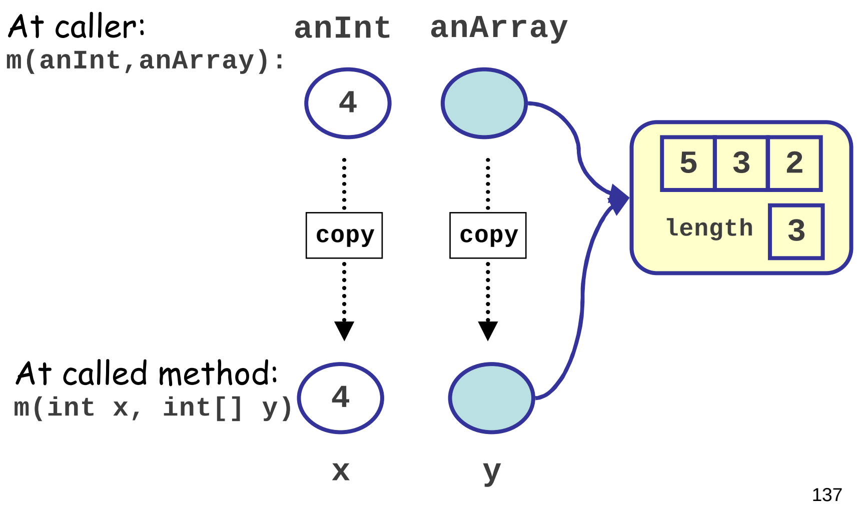
- When the method body finishes, the return value is COPIED back to the calling method and the calling method resumes execution. All other values in the called method are forgotten.

135

Trace the Program with 2 Methods ?

136

Passing primitive and reference types to a method



137

Arrays as Parameters

- An array is a reference type.
- When an array is passed from one method to another method, it is the reference that is passed to the method, not the array.
- The result is that there are (temporarily) two references to the same array.
- While we cannot modify the reference, we can modify the contents of the array. These changes to the array contents will remain after the method returns.
  - The copy of a variable of a primitive type is trashed when the method returns.
  - For an array, it is the copy of the reference that is trashed on return.

138

Trace this Program

?

```
class SwapTilYouDrop
{
   public static void main (String args[ ])
   {  int i = 0;
      int[ ] a = { 2, 4, 6, 8, 10, 12 } ;
      while ( i <= 2 )
      {
         arraySwap(a, i, 5 - i ) ;
         i = i + 1;
      }
      for ( i = 0 ; i <= 5 ; i = i + 1 )
      {  System.out.println( "A[" + i + "] is " + a[i] );}
   }
   // arraySwap : swaps values of x at positions i,j
   // Givens: x, an array, i,j, 2 indices in x
   public static void arraySwap(int[ ] x,int i,int j)
   {
      // DECLARE VARIABLES/DATA DICTIONARY
      int temp ; // Intermediate, holds x[i]
      // BODY OF ALGORITHM
      temp = x[i] ;
      x[i] = x[j] ;
      x[j] = temp;
   }
}
```

139

Trace this Program

```
class SwapExample
{
   public static void main (String args[ ])
   {  int i = 7; // declare i and initialize it to 7
      int x = 3;
      swap(i, x);
      System.out.println( "i = " + i );
      System.out.println( "x = " + x );

   }
   // swap : tries to swap 2 numbers
   // Givens: i,j, 2 numbers
   public static void swap(int i,int j)
   {
      // DECLARE VARIABLES/DATA DICTIONARY
      int temp ; // Intermediate, holds i
      // BODY OF ALGORITHM
      temp = i ;
      i = j ;
      j = temp;
   }
}
```

140

Type `char` (1)

- Characters are individual symbols, enclosed in single quotes
- Examples
  - letters of the alphabet (upper and lower case are distinct) `'A'`, `'a'`
  - punctuation symbol (e.g. comma, period, question mark)
  - single blank space
  - parentheses `'('`,`')'`; brackets `'['`,`']'`; braces `'{'`,`'}'`
  - single digit (`'0'`, `'1'`, … `'9'`)
  - special characters such as `'@'`, `'$'`, `'*'`, and so on.

141

Type `char` (2)

- Each character is assigned its own numeric code:
  - ASCII character set (ASCII = American Standard Code for Information Interchange)
    - 128 characters
    - most common character set (if you speak American English J)
    - used in older languages and computers
  - UNICODE character set
    - over 64,000 characters (international)
    - includes ASCII as a subset
    - used in Java

142

Collating Sequence

?

- In the computer, a character is stored using a numeric code.
  - The most commonly used characters in Java have codes in the range 0 through 127.
  - For these characters the UNICODE code is the same as the ASCII code.
- Examples:

character	`'a'`	`'A'`	`' '`	`'0'`	`'?'`
UNICODE value	97	65	32	48	63

- The numerical order of the character codes is called the collating sequence. It determines how the comparison operator works on characters:

  `'A' < 'a'` is

  while

  `'?' < ' '` is

143

Digit and Letter Codes

- Important features of the ASCII/UNICODE codes:
  - Codes for the digits are consecutive and ordered in the natural way (the codes for `'0'` to `'9'` are 48 through 57 respectively). Thus
    `'2' < '7'` is `true`
  - The same is true of the codes for the lower case letters (`'a'` to `'z'` have codes 97 through 122 respectively). Thus
    `'r' < 't'` is `true`
  - The same is true of the codes for the upper case letters (`'A'` to `'Z'` have codes 65 through 90 respectively).
    - Note they are smaller than the codes for the lower case letters.

144

Test for Upper case

?

- Suppose the variable **`x`** contains a value of type **`char`**.
- Write a Boolean expression that is TRUE if the value of **`x`** is an upper case letter and is FALSE otherwise.
  - Note that you don't need to know the actual code value of the characters!

145

Arithmetic on Characters

- Because the characters are stored as numbers, characters can be used in arithmetic expressions.
- Example: given N, compute the Nth letter (lower case) of the alphabet.
  (e.g. given N=1, return ‘a’, given N=5, return ‘e’)
- Java:

```
int n ;
char c ;// put the result in here
...  // code to get n’s value
```

146

Convert Upper Case to Lower Case

- Given a variable, UC, containing an upper case character, convert the character to lower case and store the result in variable LC.

147

Special characters

- Some characters are treated specially because they cannot be typed easily, or have other interpretations in Java.
  - new-line character `'\n'`
  - tab character `'\t'`
  - single quote character `'\''`
  - double quote character `'\"'`
  - backslash character `'\\'`

- All of the above are single characters, even though they appear as two characters between the quotes.

- The backslash is used as an escape character: it signifies that the next character is not to have its “usual” meaning.

148

Type conversion

- In general, one CANNOT convert a value from one type to another in Java, except in certain special cases.
- When a type conversion is allowed, you can ask for a type conversion as follows (this is called "casting"):

`(double) 3` gives `3.0`	
`(int) 3.5` gives `3`	(note loss of precision!)
`(int) 'A'` gives `65`	(this is the UNICODE value)
`(char) 65` gives `'A'`	

- WARNING: Type conversions with unexpected results have resulted in serious software problems.
  - One such error caused the self-destruction of the Ariane 501 rocket in 1996.
- The best strategy: DON'T mix types or values, unless absolutely necessary!

149

String Variables

- String variables have always presented a challenge for programming languages.
  - They have varying sizes, and for internal storage purposes, the computer would prefer to predict in advance the amount of storage needed for the value of a variable.
- As a result, strings have often been a "special case" in a programming language.

150

Strings

- String literals (constants) can be used to help make your program output more readable.
  - String literals are enclosed in double quotes:
    `"This is a string”`

- Watch out for:
  - **`"a"`** (a string) versus **`'a'`** (a character)
  - **`" "`** (a string literal with a blank that has length 1) versus **`""`** (an empty string: a string literal of length 0)
  - **`"257"`** (a string) versus **`257`** (an integer)
  - “ ” « » are not valid quotes in Java!

151

String Concatenation

- Strings can be CONCATENATED (joined) using the + operator:
  - **`"My name is" + "Diana"`** gives **`"My name isDiana"`**

- String values can also be concatenated to values of other types with the + operator.
  - **`"The speed is " + 15.5`** gives **`"The speed is 15.5"`**
  - Because one of the values for the + operator is a **`String`**, the double is temporarily converted to a **`String`** value **`"15.5"`** before doing the concatenation.

152

Strings in Java

- Strings in Java are also a reference type.
  - They are similar to an array of characters.
  - EXCEPT:
    - You don't need to use `new` to create a string
    - You don't use `[]` to access the characters in the string.

- There is a class (data type) `String` that provides many useful methods.

153

Useful `String` methods

- Suppose we have

```
    String message = "Hello World!";
```

  Then:
  - To find the length of a string:

```
    int theStringLength = message.length();
```

  - To find the character at position i (numbered from 0):

```
    int i = 4;
    char theChar = message.charAt( i );
```

- To change any primitive data type to a `String`:

```
 int anInteger = 17;
 String aString = String.valueOf( anInteger );
 // works for int, double, boolean, char
```

- To append one string after another (concatenation):

```
 String joinedString = string1 + string2;
```

154

Comparing Strings

- A **`String`** is a reference type and so they are NOT compared with ==.
- The **`String`** class has a method **`compareTo()`** to compare 2 strings.
  - The characters in each string are compared one at a time from left to right, using the collating sequence.
    - The comparison stops after a character comparison results in a mismatch, or one string ends before the other.
      - If str1 < str2, then **`compareTo()`** returns an **`int`** < 0
      - If str1 > str2, then **`compareTo()`** returns an **`int`** > 0
    - If the character at every index matches, and the strings are the same length, the method returns **`0`**

155

Comparing Strings

?

- What is the value of **`result`** for these examples?
  - Example 1:

```
String str1 = "abcde" ;
String str2 = "abcfg" ;
int result = str1.compareTo(str2);
```

  - Example 2:

```
String str1 = "abcde" ;
String str2 = "ab" ;
int result = str1.compareTo(str2);
```

156

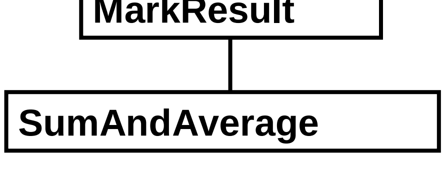
Problem Decomposition

- The best way to develop algorithms for all but the simplest problems is by Problem Decomposition (also called Top Down Design).
- An algorithm for problem P is developed by these steps:
  1. Identify subproblems (P1, P2, …, Pn), simpler than P, whose results would be useful in solving P.
  2. Finalize the header information (givens, results, etc., header) for P1…Pn but do not design their algorithms yet.
  3. Write the algorithm for P assuming algorithms exist for P1…Pn.
  4. Develop algorithms for P1…Pn.

157

Structure Chart

- A structure chart shows which other algorithms are used by each algorithm.
- Example: MarkResult uses SumAndAverage



- In general we draw a box for each algorithm, with the box for an algorithm above and connected to the boxes of the algorithms it uses.

158

Example

- What would the structure chart be if
  - P1 uses P2, P3, and P4
  - P4 uses P3 and P5

159

Bigger than Average

- Given a list of class marks, find out how many are bigger than the class average and return an array containing those that are bigger.

160

Validating numbers (example)

- Some credit cards use the following method to determine the validity of a card number: the number is valid if its last digit is equal to the last digit of the sum of the other digits.
- For example:
  - 5792 is invalid (5+7+9 = 21)
  - 4231628 is valid (4+2+3+1+6+2 = 18)
- Problem: Write a program that checks if a card number given by the user is valid. Use a loop to check more than one card, until the user enters the number zero.
- Note: The credit card numbers usually have 16 digits; the type **`int`** is not sufficient for representing such numbers. An int can only represent 4 digits.
- Assumption: The first 4 digits of the credit card number are not all zero.

Data structure for the numbers

- A **data structure** is used to organize the data used in a program.

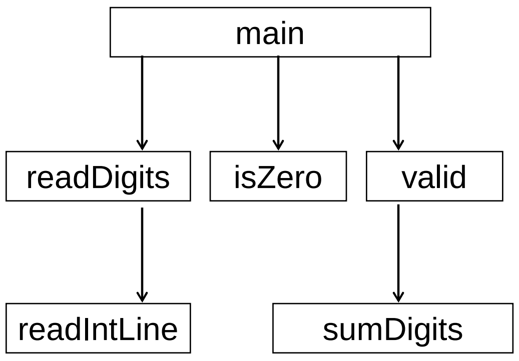
- In this problem, we use an array of 4 integers (*int*) to represent the credit card number.
  - The real numbers don't have the desired precision.

Designing the program

- A possible algorithm needs to do the following (not all the details are provided here):
  1. Read the card number input by the user (in a separate method).
  2. Check if the number is 0 (the first 4 digits).
  3. If the number is not 0, check if the number is valid or not (in a separate method).
  4. In order to check the validity of the number, call a method (sub-algorithm) that computes to sum of its first 15 digits.
  5. Display the result.
  6. Read another number input by the user.

163

Structure diagram



164

Programs with more than one Class

- A program may have more than one class. If you save all classes in a program in one directory, any class may call a `public` method in any other class in the same directory.
- When a (`static`) method is called from another class, use the name of the class with the dot operator.
  - For example, if we include class `Library` in the same directory in an assignment program, you may call a method such as `aMethod( )` by

```
Library.aMethod( );
```

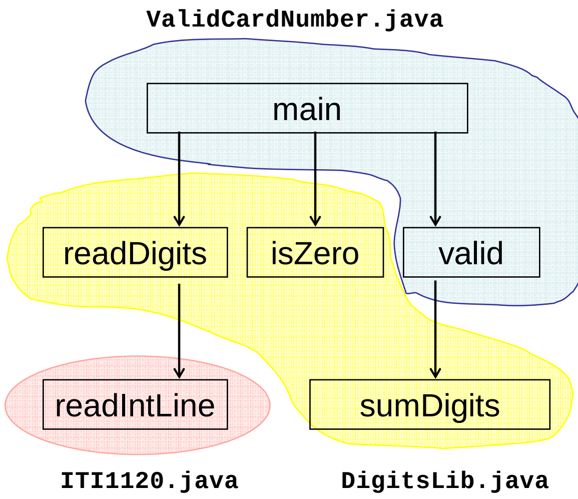
165

Library Classes

- Instead of putting all our methods in the same class as `main` (the class that contains our program) it is better to separate them into coherent groups and put each group in a class of its own.
- These classes will not be programs - they have no `main` method. Each will be a small library of methods that can be used by other methods.
- Such classes can be compiled on their own but cannot be run as standalone programs. They must be compiled before attempting to compile any other class that uses them.

166

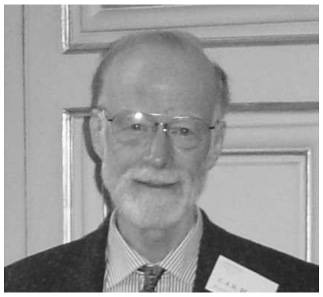
Structure diagram again



167

Historical note …

- Charles Antony Richard (Tony) Hoare, British computer scientist, developed, in 1960, the sorting algorithm (recursive) the most used: Quicksort.
- He also developed the Hoare logic used in software engineering for program verification and for programming by contracts.
- He is at the origin of the concurrent programming language CSP (Communicating Sequential Processes).

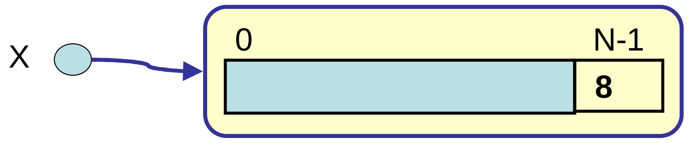


168

Recursion

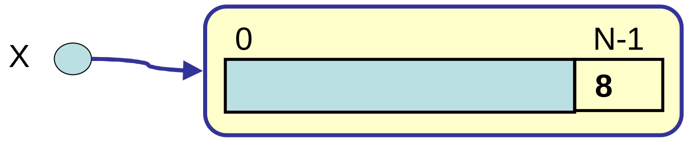
- Recursion is a problem-solving technique. Like problem decomposition, recursion involves finding sub-problems of problem P that are simpler than P to solve.
- In recursion the sub-problems are the same type of problem as P, but are simpler versions of it.

- Recursion Example 1 (basic idea): what is the sum of the numbers in positions 0…(N-1) of array X?



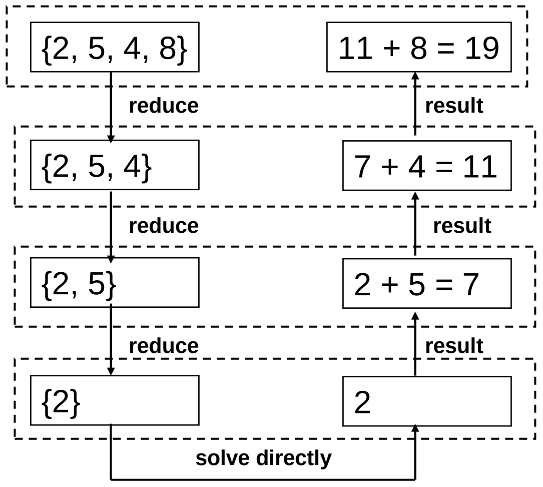
Recursion Example 2: basic idea

- What is the maximum value in positions 0…(N-1) of array X?



- (the maximum value in the shaded area is M)

Example for X = {2, 5, 4, 8}



171

Components of Recursion

There are 3 components to recursion:

1. A test to see if the problem is simple enough to solve directly (i.e. non-recursively): the "base case"

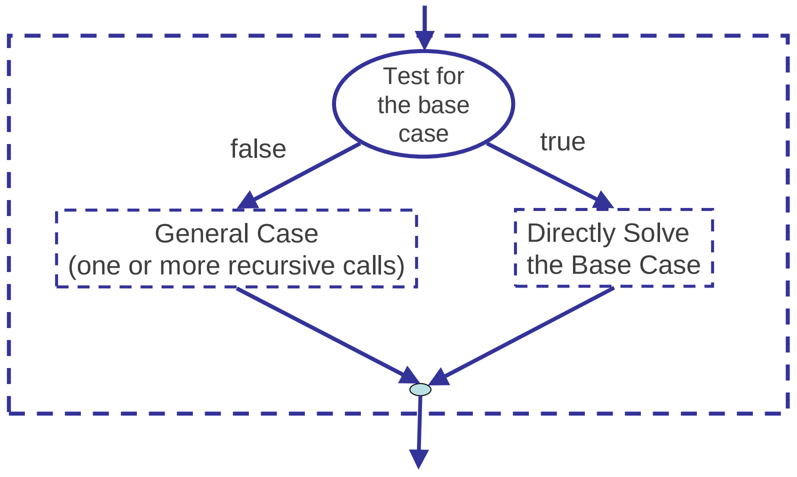
2. The solution for this simple problem.

3. A solution to the problem which involves solving one (or more) smaller versions of the same problem.

172

Template for Recursion

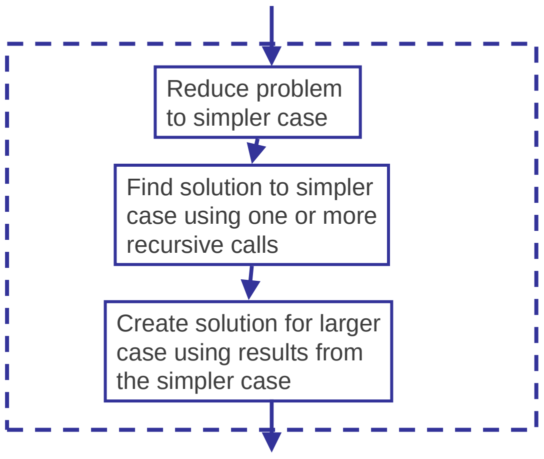
- Recursive algorithms usually have this form:


Test for the base case
false
true
General Case (one or more recursive calls)
Directly Solve the Base Case

173

Recursion: The General Case

- On the previous slide, the general case can be further decomposed into these steps:


Reduce problem to simpler case
Find solution to simpler case using one or more recursive calls
Create solution for larger case using results from the simpler case

174

Recursion Examples

1. What is the sum of the numbers in positions 0…(N-1) of array X?
2. What is the maximum value in positions 0…(N-1) of array X?
3. Find X^N where X and N are integers and $N \geq 0$, $X \geq 1$.
   - Direct algorithm
   - Alternative algorithm based on fact:
     $$X^N = X^I * X^{N-I}$$
   - Choose I to get most efficient version.
4. Given an array A of more than N numbers, return TRUE if all the numbers in positions 0…N of A are equal, and false otherwise.

175

More Recursion Examples

5. Calculate N !
6. Find the sum of 1+2+…+N.
7. Given an array A of N characters, reverse the values stored in positions Start to Finish.
8. Sort an array of numbers. (Put its values in increasing order.)

176

Recursive Sum of Array

?

- Write a recursive algorithm to find the sum of the values in array positions 0…(N-1):

GIVENS:

INTERMEDIATES:

RESULT:

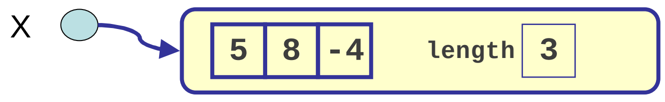
HEADER:

BODY:

```
        S      [          ]
        Sum    S + X[N-1]
```

177

Trace for this value of X:



178

Recursive Algorithms and Recursive Methods

- An algorithm that calls itself, with different GIVENS, is called a recursive algorithm.
- A Java method that calls itself, with different arguments, is called a recursive method.

179

Translate the Recursive Sum of Array to Java

?

180

Historical note …

- 1998: Larry Page and Sergey Brin, founded a company that revolutionized the world of search engines and the Internet: **Google!**
- Many applications, such GoogleMaps.
- 450 000 servers.
- More than a milliard qu per day!

181

Matrices

- An R × C matrix has R rows and C columns.
- Example. A 4 x 6 matrix of integers

$$M = \begin{bmatrix} 71 & 62 & 33 & 89 & 85 & 74 \\ 68 & 65 & 75 & 84 & 70 & 72 \\ 87 & 0 & 1 & 90 & 92 & 88 \\ 58 & 72 & 66 & 57 & 76 & 73 \end{bmatrix}$$

M[r][c] is the entry at row r and column c. (Indices start from 0).

182

Matrices and 2-dimensional Arrays

?

$$M = \begin{bmatrix} 71 & 62 & 33 & 89 & 85 & 74 \\ 68 & 65 & 75 & 84 & 70 & 72 \\ 87 & 0 & 1 & 90 & 92 & 88 \\ 58 & 72 & 66 & 57 & 76 & 73 \end{bmatrix}$$

- A matrix is represented in algorithms by a 2-dimensional array, i.e., an array of arrays.

- The matrix M is an array of 4 arrays, each with 6 members. If M is regarded as a 2-dimensional array, then

  M[1][2] is

  M[2][5] is

  M[4][1] is

  M[3] is

183

Max value in a matrix

?

GIVENS:

INTERMEDIATES:

RESULT:

HEADER:

BODY:

184

Alternative Algorithm

?

185

Diagonal Matrices

- A square matrix has the same number of rows and columns. If all its "off-diagonal" values are 0 it is a diagonal matrix.
- For example, in the following matrices,

$$M1 = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad M2 = \begin{bmatrix} 2 & 4 & 0 \\ 3 & 5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- M1 is a diagonal matrix and M2 is not a diagonal matrix.
- Write an algorithm that checks if a given square matrix is diagonal.

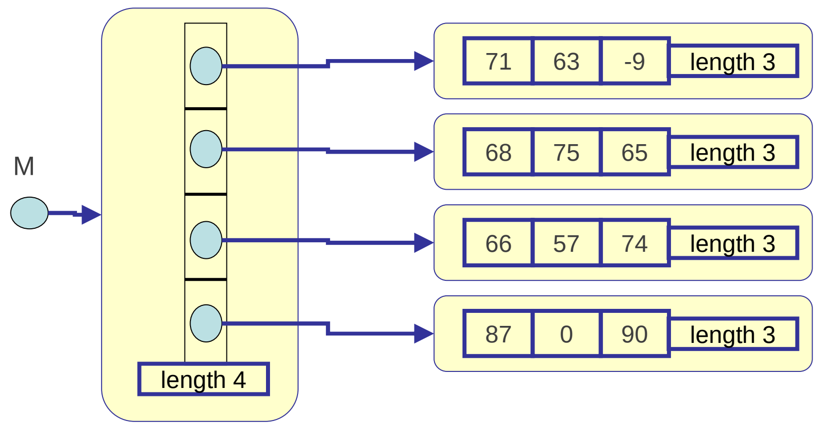
186

Diagonal-check algorithm

Efficient Version

2D Arrays in Java

- A 2D array in Java is literally an array of arrays (each entry in the array is itself an array).



Declaring a 2D Array

- To declare `m` to be a 2-dimensional array of integers:
```
    int [][] m;
```

- To create a 2x3 instance of a 2D array (i.e. allocate memory for the array and all the sub-arrays) and assign it to `m` :
```
    m = new int[2][3];
```

- To create an initialized 2 x 3 2-dimensional array :
```
    int [][] m;
    m = new int[][] { {1, 2, 3}, {4, 5, 6} };
```

- You may use `length` to find the dimension of a 2-dimensional array, or any sub-array:
  `m.length` is
  `m[0].length` is
  What about `m[0][0].length`?

Max value in a matrix (1)

?

- Translate the algorithm for the maximum value in a matrix to Java:
  - Note: `Integer.MIN_VALUE`is the most negative allowable integer for a Java `int`, and can be used for –∞.

191

Max value in a matrix (2) in Java

- The header for the method for finding the maximum value in an array is:

```
public static int arrayMax(int[] a, int n)
```

192

Reading a Matrix

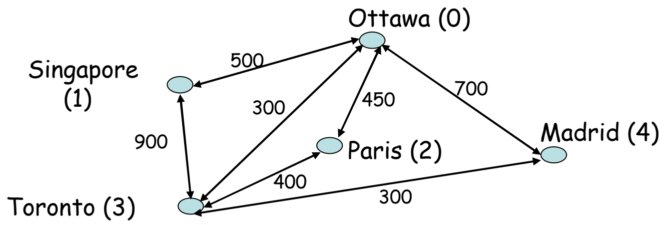
?

- Write Java code to read in a matrix row by row (first it reads in the number of rows and columns, then it asks for the values in row 0, then it asks for the values in row 1, etc.). All values are read one per line.

193

Adjacency Matrix

- Escape Airlines has flights between certain cities. The flights and their costs can be represented as a graph in which an edge between city X and city Y with a weight (label) of W means Escape Airlines has a flight between X and Y costing W dollars.



194

Matrix Representation

- This graph can be represented with an adjacency matrix. There is a row and a column for each city, and COST[X][Y] is the cost of a flight from X to Y if one exists and is infinity (∞) if there is no such flight.

$$\text{COST} = \begin{bmatrix} 0 & 500 & 450 & 300 & 700 \\ 500 & 0 & \infty & 900 & \infty \\ 450 & \infty & 0 & 400 & \infty \\ 300 & 900 & 400 & 0 & 300 \\ 700 & \infty & \infty & 300 & 0 \end{bmatrix}$$

- Here, “infinity” is actually a very large number, greater than any number.
  - In Java: a predefined constant is available for the largest possible integer: `Integer.MAX_VALUE`

Find Cheap Direct Flights

?

- Suppose you live in one of Escape’s cities and have $D to spend. Write an algorithm that returns an array of the cities you can afford to fly to **directly** (and how many there are).

Translate to Java

?

197

Deleting rows and columns

- Escape Airlines has decided to stop flying to and from city X (e.g. Paris, X=2), and the city numbers greater than X have all been reduced by 1 (e.g. Madrid is now city 3).
  - This problem can be solved by deleting a row and a column that correspond to the city.
- Write two algorithms, one to delete a row from a matrix, the other to delete a column.
- We are not going to change the size of the matrix, but we are going to shift the rows and columns up and put zeros in the last row and column.

198

Historical note …

- Barbara Liskov was the first women to have obtained her Ph.D. in computer science in US (in 1968, from Stanford University).
- She was at the origin of CLU, the first language that supported abstract data types (1975), that influenced many object-oriented languages, including Java.
- In 1993, she and Janette Wing have developed a specific definition of sub-types, the Liskov principle of substitution, used in object-oriented programming.

199

Student Information

- Suppose we want to store a collection of information about each student in a course.
- How to store all the information for one student?
  - ID (student number) (integer)
  - midterm mark (real)
  - final exam mark (real)
  - is taking this course for credit (Boolean)

- Why can't we use an array?

200

Records

- Like an array, a "record" allows several values to be stored in one variable.
- Records differ from arrays in 2 ways:
  - The values (called fields) in a record can be of different types.
  - Each field in a record has a NAME. A value is accessed by specifying the field name (not a subscript).
- Example (a single record with 4 fields):

field name	**field value**
id	1234567
midterm	60.0
exam	80.0
forCredit	TRUE

201

Using Records

- Suppose the preceding record was stored in a variable named R.
- To access the midterm mark:

```
R.midterm
```

- This refers to one field inside record R. A field can be used anywhere a variable of that type is allowed, e.g.,

```
T ← R.midterm + R.exam
R.forCredit ← false
```

- The whole record can be used in an assignment statement or passed as a parameter:

```
X ← R
```

202

Defining a Record Type

- When we discussed primitive types, we looked at:
  - What values does the type allow?
  - What operations one can do with values of that type?
- A record is a "user-defined" type that is built using types we have already:
  - Primitive types
  - Other user-defined types.
- Creating a record also has the two elements of primitive types:
  - What are the components of a record value?
  - What operations can we do with the record?

203

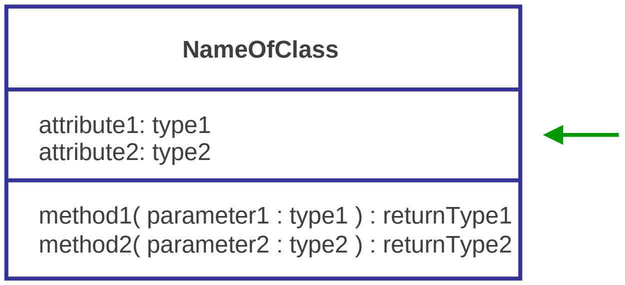
Classes and Objects

- An object can be considered to be like a record, in that there is a set of "attributes" – named data values stored in the object.
- Each object is created from a class.
- A "class" can be used as a template to create objects with identical sets of attributes.
  - The class can also contain methods (algorithms) to perform calculations on the attributes of objects created from the class (and/or external data).
- A method is called on an object using the **.** operator, in a similar manner to accessing a record field

```
result ← anObject.aMethod( aParameter )
```

204

Class Diagrams


NameOfClass
attribute1: type1
attribute2: type2
method1(parameter1 : type1) : returnType1
method2(parameter2 : type2) : returnType2

"Attributes" are like the field variables in a record

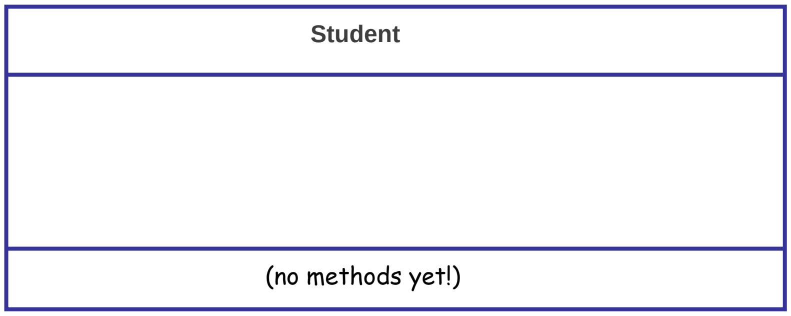
- This form of diagram is from a notation called the "Unified Modelling Language" or UML

205

First version of a Student Class

?

- For each student, we want to store their ID number, their midterm score, their exam score, and whether or not the student is taking the course for credit.



206

Translation to Java

```
public class Student
{
   public int id;
   public double midterm;
   public double exam;
   public boolean forCredit;
}
```

- The class is a "template" for how to construct Student objects.
  - Objects must be created using the `new` statement

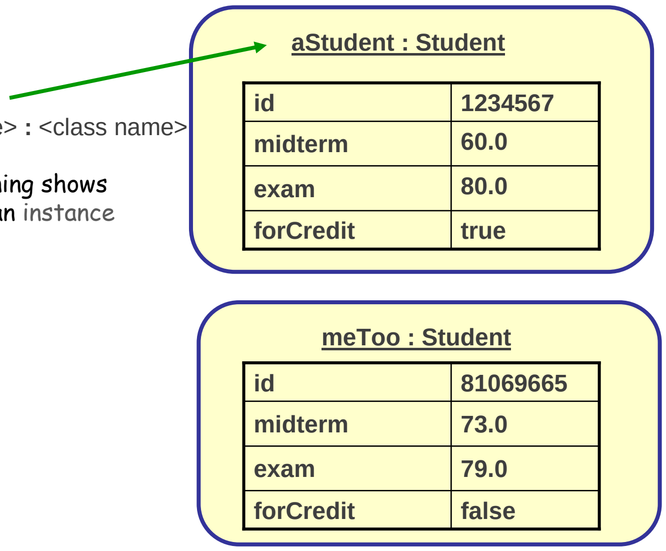
```
Student aStudent;
aStudent = new Student( );
```

207

Two Student objects

format:
<object name> : <class name>

(the underlining shows that this is an instance diagram)



aStudent : Student

id	1234567
midterm	60.0
exam	80.0
forCredit	true

meToo : Student

id	81069665
midterm	73.0
exam	79.0
forCredit	false

208

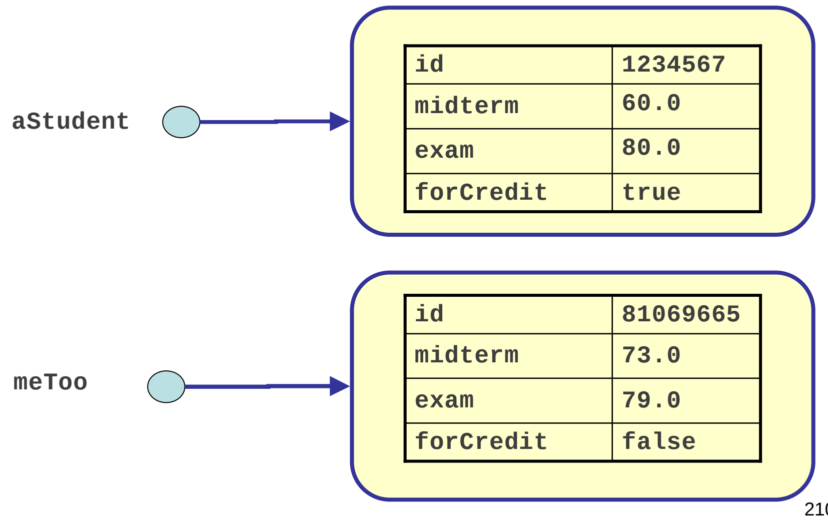
Object usage in Java

```
Student aStudent;                 // declare variable
aStudent = new Student();         // create new object
aStudent.id = 1234567;
aStudent.midterm = 60.0;
aStudent.exam = 80.0;
aStudent.forCredit = true;

Student meToo;
meToo = new Student();
meToo.id = 81069665;
meToo.midterm = 73.0;
meToo.exam = 77.0;
meToo.forCredit = false;
```

209

Java objects are references



210

Information hiding

- Suppose we want to modify the Student class to keep the course final mark, which is 20% of the midterm mark plus 80% of the final mark.
  - We could add a field finalMark to our class.

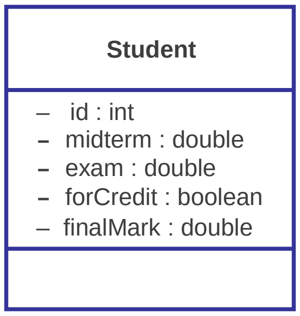
- We want to make sure that
  finalMark = (0.2 × midterm + 0.8 × exam)
  is always true for consistency.

- It would be useful to prevent anyone else from setting the value of finalMark arbitrarily.
  - Instead, if the final mark is to change, it should be done by changing the value of either midterm or exam.

- Restricting access to data is called "information hiding".

211

Private fields in a class



- The – in front of the variable indicates that the attribute is private.
- By declaring a field to be private, only methods declared inside the class are allowed access to the field value (either for viewing the value, or changing the value).

212

Accessors and Modifiers

- To allow access to private values from outside a class, we can use methods for this purpose:
  - "accessor": ask to view the value of a private field.
    - General form: No input parameters, returns the field value.
  - "modifier": ask to change the value of a private field.
    - General form: One input parameter (the new value for the field), and no return value.

- A naming convention that is often used:
  - accessor for name: the method getName()
  - modifier for name: the method setName(newName)

213

Accessors and Modifiers

- Examples for the forCredit field in the class:

  + getForCredit( ) : boolean

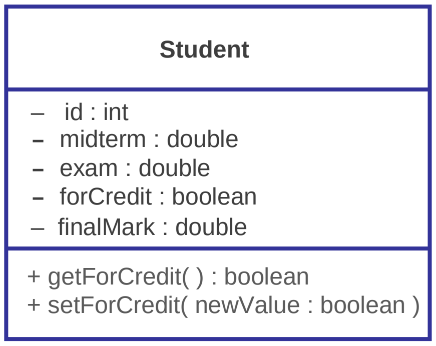
  - method to return the value of forCredit
  - the + indicates that the method has public visibility
  - the return type is boolean, and in UML notation, appears at the end of the method.

  + setForCredit(newValue : boolean)

  - method to change the value of forCredit
  - one parameter newValue, of type boolean
  - no return value

214

Class diagram with accessors and modifiers



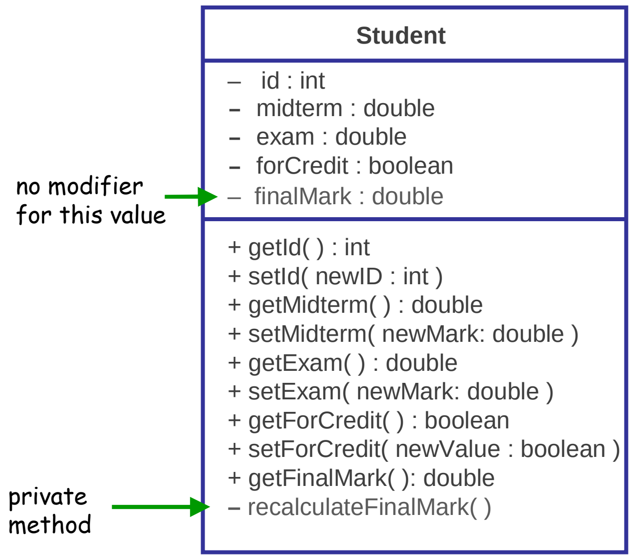
215

Back to Information Hiding

- To implement our strategy of hiding the finalMark field, we can do the following:
  - We will provide an accessor method for finalMark, but NOT a modifier method.
  - We can provide a helper method recalculateFinalMark() to recalculate the final mark if the midterm or exam marks are changed.
  - The modifier methods setMidterm() and setExam() will call recalculateFinalMark() so that they automatically update the final mark.
- We should also restrict access to recalculateFinalMark() because it isn't meant for use outside the class.

216

Student class with Information Hiding


Student
– id : int
– midterm : double
– exam : double
– forCredit : boolean
– finalMark : double
+ getId( ) : int
+ setId(newID : int)
+ getMidterm( ) : double
+ setMidterm(newMark: double)
+ getExam( ) : double
+ setExam(newMark: double)
+ getForCredit( ) : boolean
+ setForCredit(newValue : boolean)
+ getFinalMark( ): double
– recalculateFinalMark( )
no modifier for this value
private method

Translation to Java

```
public class Student
{
   // Attributes

   private int id;
   private double midterm;
   private double exam;
   private boolean forCredit;
   private double finalMark;

   // Methods

   public int getId()
   {
      // insert code here
   }
   public void setId( int newId )
   {
      // insert code here
   }
   public double getMidterm()
   {
      // insert code here
   }
   public void setMidterm( double newMark )
   {
      // insert code here
   }
   // continued at right
```

```
   // continued from left side

   public double getExam()
   {
      // insert code here
   }
   public void setExam( double newMark )
   {
      // insert code here
   }
   public boolean getForCredit()
   {
      // insert code here
   }
   public void setForCredit( boolean newValue )
   {
      // insert code here
   }
   public double getFinalMark()
   {
      // insert code here
   }

   private void recalculateFinalMark()
   {
      // insert code here
   }
} // end of class Student
```

Using Java Accessor and Modifier Methods

- If we now try the following:

```
Student aStudent = new Student();
aStudent.id = 1234567;            // error here
int myId = aStudent.id;           // error here
System.out.println( myId );
```

  the compiler will give us two errors, because we no longer have access to `id` outside the class.

- The compiler enforces the private access to `id` .

- Instead, use the modifier and accessor methods.

```
Student aStudent = new Student();
aStudent.setId( 1234567 );      //ok!
int myId = aStudent.getId( );   //ok!
System.out.println( myId );
```

219

Implementing Java accessors and modifiers

```
public class Student// not all attributes/methods shown!
      {
         // attribute
         boolean forCredit;

         // accessor:  return the requested value
         public boolean getForCredit()
         {
            return this.forCredit ;
         }

         // modifier: save the requested value in object's
         // attribute

         public void setForCredit( boolean newValue )
         {
            this.forCredit = newValue;
         }
      }
```

220

Where did `this` come from?

- When the fields of our Student class were public, we distinguished between the same field in two record objects with the variable name and the dot operator:
  - `aStudent.forCredit` versus `meToo.forCredit`
- Likewise, when a method inside the class wants to work with "the value of the field for the object on which I was called", `this` refers to the called object.
- During the call `aStudent.getForCredit()` `this` is a reference to `aStudent`
  - … and so `this.forCredit` is `aStudent.forCredit`, which is true.
- During the call `meToo.getForCredit()` `this` is a reference to `meToo`
  - … and so `this.forCredit` is `meToo.forCredit`, which is false.

221

Implementing Information Hiding

- The following implements our strategy where the final mark can only be changed by modifying the midterm or exam values.

```
public class Student
{
   // attributes and other methods would go here
   public void setMidterm( double newValue )
    {
       this.midterm = newValue;
       this.recalculateFinalMark( );
    }
    public void setExam( double newValue )
    {
       this.exam = newValue;
       this.recalculateFinalMark();
    }
    private void recalculateFinalMark()
    {
       this.finalMark = 0.2 * this.midterm + 0.8 * this.exam;
    }
 }
```

222

Benefits of Information Hiding (1)

- A common cause of problems is when all parts of a program have access to all program variables.
  - For example, when someone makes a change to a large program, the new code may make changes to data that some other part of the program assumed would not be modified.

"Successful software always gets changed." - *F. Brooks*

- With information hiding, we can keep the code better partitioned so that changes will be less likely to cause unwanted side effects.

223

Benefits of Information Hiding (2)

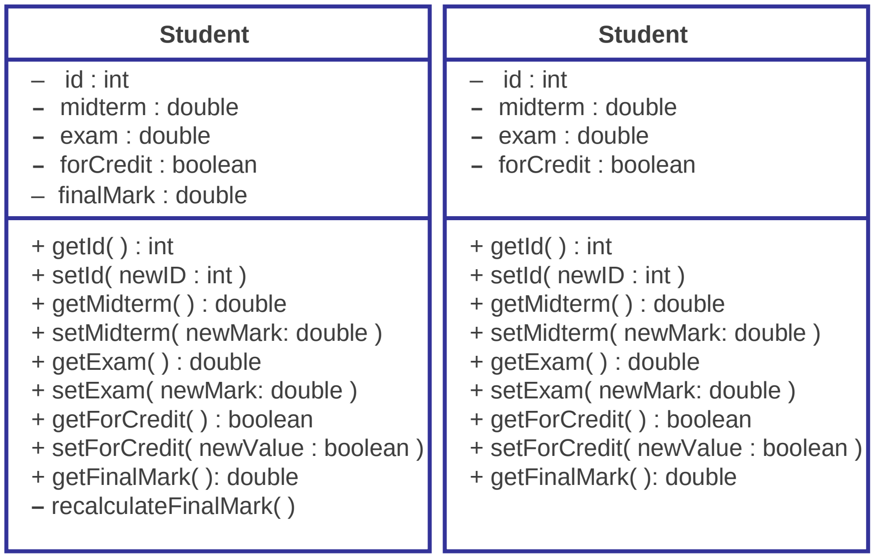
- We can also make changes inside a class that will not affect users of the class.
- Example: Suppose we decide that the finalMark field really doesn't need to be stored in the Student class.
  - Instead, we can calculate the final mark when anyone asks for it:

```
public double getFinalMark()
{
   return 0.2 * this.midterm + 0.8 * this.exam;
}
```

  - This means we can remove the helper method recalculateFinalMark( ), and the calls to it in setMidterm( ) and setFinal( ).
- Making these changes will not affect any user of the class:
  - For example, meToo.getFinalMark( ) still behaves as it did before.
  - Since recalculateFinalMark( ) was private, code outside the class was not able to call this method, and therefore it can be safely removed.
- So we don't have to change any code outside the class!

224

Compare Versions



`this`, again

- In most cases, we don't actually have to use `this` to refer to the object on which a method is called.
  - Inside the Student class:
    - `exam` can be used instead of `this.exam`
    - `recalculateFinalMark()`can be used instead of `this.recalculateFinalMark()`

- There are 2 occasions when we really do need `this`:
  1. An object wants to pass itself as a parameter to a method of another class.
  2. An object wants to return a reference to itself as the result of a method.

226

Historical note …

- The **Xerox Palo Alto Research Center** (PARC), founded in 1970, is at the origin of many important contributions:
  - The first workstation (Alto) with graphical user interface (GUI, with windows and icons) and mouse
  - The first text editor WYSIWYG
  - The InterPress language (predecessor of PostScript) for describing pages to be printed
  - The Ethernet protocol for local area networks
  - The Smalltalk object-oriented programming language, with graphical development environment (designed by Alan Kay)
  - The laser printer
  - ...

227

Object-orientation

- The approach we have taken with our student class is an "object oriented" approach:
  - We have a class that is a template for the creation of objects.
    - Student objects can be referred to as INSTANCES of the CLASS Student.
  - Object instances have instance methods that use the field values for a specific object
    - e.g. getFinalMark( ) will have different results for different objects because this.exam is different for different objects
  - If you want the object to do something for you, you have to ask it by calling a method on that object.
    - That is, you can't sneak inside an object from outside the class and change the field values.
  - You also can't call an instance method, without having an object to call it on:

    X ← 3.0 + getFinalMark( ) is meaningless. Whose final mark are we referring to?

228

Initialization of Objects

- When we create a new Student, we will usually want to provide values for all the fields in the object.

```
aStudent ← new Student( )
aStudent.id ← 1234567
aStudent.midterm ← 60.0
aStudent.exam ← 80.0
aStudent.forCredit ← True
```

- A special kind of method called a CONSTRUCTOR, can be used to initialize values inside an object as the object is being created.

```
aStudent ← new Student( 1234567, 60,0, 80.0, True)
```

229

Constructors

- A constructor is a special method in a class used to create an object.
  - the name of the method is the same as the class;
  - no return type
  - usually public;
  - may or may not have parameters.
- The parameters, if any, in a constructor are used to initialize the values of the object.
- Because there may be different ways to initialize an object, a class may have any number of constructors, distinguished from each other by different parameter lists.

230

Implementation in Java

- The following is a constructor that sets a value for all of the fields in the **Student**:

```
class Student
{
   // ... fields would be defined here ...
   public Student(int theId, double theMidterm, double theExam, boolean
                 isForCredit)
   {
      this.id = theId;
      this.midterm = theMidterm;
      this.exam = theExam;
      this.forCredit = isForCredit;
   }
   // ... Other methods ...
}
```

- This constructor could be used as follows:

```
Student aStudent = new Student(1234567, 60.0, 80.0, true);
```

231

Constructors of class Student

- If we are doing course registrations, we may only want to provide the ID number and whether the student is taking the course for credit. (We don't know the student's marks yet!)
- We could also provide the following constructor:

+ Student(theID : int, isForCredit : boolean)

```
public Student(int theID, boolean isForCredit )
{
   this.id = theID;
   this.midterm = 0.0;            // a "safe" value
   this.exam = 0.0;               // a "safe" value
   this.forCredit = isForCredit;
}
```

- When there is more than one constructor, they must have parameter lists that can be distinguished by the number, order, and type of parameters.

232

Add Constructors to the Class

```
Student
------------------------------------------------------------------------
–  id : int
–  midterm : double
–  exam : double
–  forCredit : boolean
------------------------------------------------------------------------
+ Student( theID : int, theMidterm : double, theExam : double, isForCredit: boolean )
+ Student( theID : int, isForCredit: boolean )

+ getId( ) : int
+ setId( newID : int )
+ getMidterm( ) : double
+ setMidterm( newMark: double )
+ getExam( ) : double
+ setExam( newMark: double )
+ getForCredit( ) : boolean
+ setForCredit( newValue : boolean )
+ getFinalMark( ): double
```

Constructors of class Student

- Here is a constructor with no parameters:

  + Student(   )

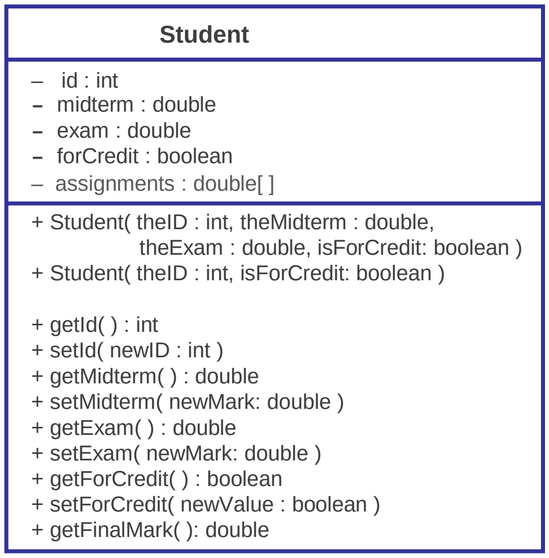
```
public Student( )
{
   this.id = 0;
   this.midterm = 0.0;
   this.exam = 0.0 ;
   thisl forCredit = false;
}
```

- A constructor without parameters is called a default constructor.
  - It is recommended to always define a default constructor that sets every field to a “safe” value.

- If, and only if, a class does not define any constructor, the Java compiler invisibly creates a default constructor that does nothing.

234

Array Fields in Classes



Student

– id : int
– midterm : double
– exam : double
– forCredit : boolean
– assignments : double[ ]

+ Student(theID : int, theMidterm : double, theExam : double, isForCredit: boolean)
+ Student(theID : int, isForCredit: boolean)

+ getId( ) : int
+ setId(newID : int)
+ getMidterm( ) : double
+ setMidterm(newMark: double)
+ getExam( ) : double
+ setExam(newMark: double)
+ getForCredit( ) : boolean
+ setForCredit(newValue : boolean)
+ getFinalMark( ): double

- A field of a class may have any type. In particular, a class may have a field of an array type.
- Add an array of double to class Student representing assignment marks:
- Remember, arrays are not created automatically! The array assignments will have to be created after a Student object is created.

235

Array Fields in Classes

```
public class Student
{
   private int id ;
   private double midterm ;
   private double exam ;
   private boolean forCredit;
   private double[] assignments;

   // methods
}
```

- The array `assignments`is still `null`.

236

Array field initialization

- Here is a constructor that creates and initializes an array in an object. The constructor has a parameter that is the number of assignments.

```
public Student( int numberOfAssignments )
{
   this.id = 0;
   this.midterm = 0.0;
   this.exam = 0.0 ;
   this.forCredit = false;
   this.assignments = new double[numberOfAssignments];

   // loop to initialize each item in array
   int index;
   for ( index=0; index < numberOfAssignments; index =index+1 )
   {
      this.assignments[index] = 0.0;
   }
}
```

237

Accessors for an Array Field

- An accessor for an array field could:
  - Return a reference to the entire array
    + getAssignments() : double[ ]
  - Return one of the values in the array, with an extra parameter to select the array index.
    + getAssignment (assignmentNumber: int) : double
- Which approach is better?
- Write an accessor for the length of an array **numAssignments**, that returns the number of assignments of a given **Student.**

238

Calculation of the Final Mark

?

- Write a Java method `getFinalMark( )` for our `Student` class that returns a `double` with a student's final mark, where:
  - The final mark is 55% of the exam, plus 20% of the midterm, plus 25% of the average of 5 assignments.

239

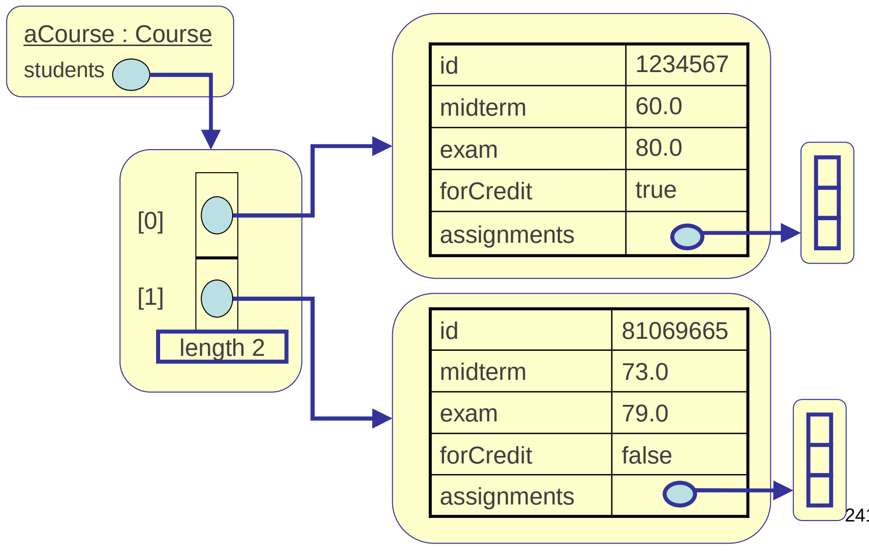
Course information

?

- Now that we have a class that stores information about one Student, how can we use this create a class Course that stores information about all students?

240

Array of Students



241

Accessing the Data(1)

- Change the 3rd student's midterm to 77.

- Change the 1st student's 2nd assignment mark to 99.

- Use the **`getFinalMark`**method to compute the term work mark of the 2nd student.

- Suppose **`arrayMax(A,N)`**is a class method that finds the maximum value in the first **`N`** positions of array **`A`**. Use this to compute the 2nd student's maximum assignment mark.

242

Accessing the Data(2)

- Count the number of students taking the course for credit.

Searching in an Array of Records

- Write a method that returns the position in a given course C of the student with ID = X (return -1 if there is no such student).

```
public __________
 findID(______________________)
{




}
```

Memory Allocation Method

- As before, it is handy to put the code for memory allocation into a constructor

```
public Course
       (int numberOfStudents, int numberOfAssignments)
 {
   int i ;
   students = new StudentRecord[numberOfStudents] ;
   for (i=0 ; i < numberOfStudents ; i=i+1)
     students[i] = new Student(numberOfAssignments);
 }
```

245

Creating and Accessing the Data

- Declare variable C to be a **Course** object (instance of the Course class):

- Allocate memory for C (137 students, each with 10 assignments):

- Compute the term work mark for the 3rd student in C:

246

Class Methods

- Object instances have instance methods that use the field values for a specific object, e.g.,
  `public double getFinalMark(){…}`
- Before introducing objects to represent records, all methods were class methods, e.g.,
  `public static double avg3(int a, int b, int c)`
- A class method is called as follows:
  ClassName . aMethodName( )
  - Just like the `Math` class!
- A class method CANNOT use any instance variables, because it is not associated with any particular object.

247

Summary of Class Design

- In an object-oriented language such as Java, designing a class is a large part of the effort to create software.

"Classes struggle, some classes triumph, others are eliminated." --Mao Zedong

- Decisions have to be made as to:
  - What information should be in the class?
    - What values should each object have?
    - What type are the values?
    - How do we initialize, set, and change the values?
  - What are the operations we may want to ask the class to perform?
    - What other instance methods are needed?
    - What other class methods are needed?
    - What are the algorithms for all of these methods?

248

Class case study: Fractions

- Specification for a Fraction class:
  - A fraction consists of a numerator and a denominator.
  - The numerator of a fraction can be any integer.
  - The denominator of a fraction can be any integer other than 0.
    - If the denominator is not specified at creation, it is assumed to be 1.
  - A fraction is always in "standard form"; that is
    - The greatest common divisor (GCD) of the numerator and denominator is always 1
    - The denominator is always positive.
      - Example: 6/-9 should be represented as -2/3
      - Special case: if the numerator is 0, the fraction is represented as 0/1
  - A fraction with denominator 1 should be displayed as the equivalent integer; otherwise in the form numerator/denominator.

249

Designing a Fraction class

?

- What information do we need to store in a Fraction?

- What operations do we need?
  - [Aside from creating fractions, the only mathematical operation we will implement is addition of two fractions]

250

Putting the Fraction in Standard Form

- To make sure that each `Fraction` instance will be in lowest terms, a helper method `simplify` will be used.
- Assume that you have a method `gcd(a,b)` that will return the greatest common divisor of two integers.
- Write a Java method to put a fraction into standard form.

Algorithm for GCD

- A recursive GCD algorithm for gcd(a,b):
  - If a mod b is 0, gcd(a, b) is b
    - a mod b is the remainder when a is divided by b
  - Otherwise, gcd(a,b) is gcd(b, a mod b)
- Question: will this algorithm always reach the base case?
  - Note that a mod b is at most b – 1.
- Write a recursive Java method to calculate gcd(a,b)

```
private static int gcd (int a, int b) // a class method
{   int result;
   int remainder;
   remainder = a % b;
   if ( remainder == 0 )
   {
      result = b;
   }
   else
   {
      result = gcd( b, remainder );
   }
   return result;
}
```

Fraction Constructors

?

- Write constructors for a `Fraction` that:
  - take 2 integers: the numerator and the denominator
  - takes 1 integer, representing an integer that is to be converted to a Fraction

253

Displaying Fractions

?

- Write a Java method to display a `Fraction`
- Sample usage:
  - `Fraction f1 = new Fraction ( 6, -9 );`
  - `f1.display( );`
- Result: `-2/3`

);

254

Adding Fractions

?

- Write a Java method that will add two Fractions.
  - Sample usage:

```
Fraction f1 = new Fraction( 1, 2 );
Fraction f2 = new Fraction( 1, 3 );
Fraction sum = f1.addTo( f2 );
sum.display( );
```

  - Result: `5/6`

255

Some final words…

- "At the source of every error which is blamed on the computer, you will find at least two human errors, one of which is the error of blaming it on the computer. "
  - Anonymous

- "We shall do a much better programming job, provided we approach the task with a full appreciation of its tremendous difficulty, provided that we respect the intrinsic limitations of the human mind and approach the task as very humble programmers. "
  - Alan Turing

256