Java Fundamentals

• Problem solving
  – NIM

• Rudiments of Java
  – Classes, objects, methods, constructors
  – Declarations, statements, output

• Backus Naur Form (BNF)
Goal: Problem Solving

- Problem solving
  - Defining the problem
  - Designing a solution
  - Implementing a solution
  - Testing your solution and debugging it
  - How to decide your solution “works”

- How to do this in an object-oriented style?
Problem Solving

• Algorithm
  – Precise, unambiguous set of steps to follow to solve a problem

• Simple game: NIM
  – n objects
  – players alternate turns
  – a player must pickup either 1 or 2 objects
  – loser is player who picks up last object
NIM

• What is optimal strategy for playing NIM?
• Must assume each player tries to win game
• What do we know about the game?
• If 1 object left,
  – Pick it up and declare self “loser” !!??!!
  – $n=1$ is a losing state for whoever has that turn
NIM

• If 2 objects left,
  – Pick up 1 object and force opponent to lose
  – Pickup 2 objects and lose; NO
  – n=2 is a winning state

• If 3 objects left,
  – Pickup 2 objects and force opponent to lose
  – n=3 is winning state
NIM Analysis

- A plays from 6 pieces:
  - A removes 2, B removes 2, A removes 1, A wins 😊
  - A removes 2, B removes 1, A removes 2, A wins 😊
  - A removes 1, B removes 2, A removes 2, A wins 😊
  - A removes 1, B removes 1, A removes 2, B removes 1, A loses 😞
  - So A always removes 2 to guarantee a win!
NIM Analysis

• Easier to reason about the game from its end to its beginning.
  – n=1 is a losing state, n=2, n=3 are winning states
  – n=4? if take 1 or 2 put opponent into n=3 or n=2, both winning states; therefore, n=4 is a losing state!
  – n=5? if take 1, put opponent into n=4 which is a loss for him. if take 2, put opponent into n=3 which is a win for him; therefore, will always take 1 and win!
NIM Strategy

• \( n = 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \)
  \ L \ W \ W \ L \ W \ W \ L \ W \ W \ L \ W \ W \)

• How many objects to remove?
  – Can be calculated each time
  – Can encode in a formula

• Rule:
  – If \( n \) is multiple of 3, remove 2
  – If \( n \) is not multiple of 3, remove 1
NIM(3)

• Simple game: NIM(3)
  – n objects
  – players alternate turns
  – player must pickup 1, 2 or 3 objects
  – loser is player who picks up last object

• What’s the optimal strategy for winning?
NIM(3)

- $n=1$ is losing state
- $n=2$ is winning state
- $n=3$ is winning state
- $n=4$ is winning state
- $n=5$ is losing state...
NIM(3) Strategy

\[ n = 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \]

\[ \begin{array}{cccccccccccc}
L & W & W & W & L & W & W & W & L & W & W & W & L
\end{array} \]

\[ r = 0 \ 1 \ 2 \ 3 \ 0 \ 1 \ 2 \ 3 \ 0 \ 1 \ 2 \ 3 \ 0 \]

For \( r = (n+3) \mod 4 \) where \( \mod \) yields remainder from integer division

- If \( r \) is not zero, remove \( r \) objects
- If \( r \) is zero, remove 1 object
NIM(k) Strategy

- Remove 1, 2, 3,...,(k-1), or k on each move

- Rule:
  - $r = (n+k) \%(k+1)$
  - If $r$ not zero, remove $r$ objects
  - If $r$ is zero, remove 1 objects

- Works for any game of this family.
Next Step

• Write a program that can play NIM against a person, using the winning strategy we derived

• Need to know intrinsic components of a Java program before doing this

• Basic idea we have used is same notion as in IBM Deep Blue chess program which beat Gary Kasparov last year!
Example 1, Airport

- **Objects** - airplanes, crew members, food trucks, baggage trams, etc.

- **Actions**
  - `removeBaggage` for baggage trams
  - `takeOff` for planes
  - `loadMeals` for food trucks
Fundamentals

- **Program** - set of interdependent classes with one specified as the distinguished class (where computation starts)

- **Class** (or type) - a description of attributes (properties) and operations (capabilities) shared by some objects in the problem being solved
  - e.g., plane, foodTruck, crew, baggageTram
Fundamentals

• Class
  – Each attribute (sometimes called instance variable) described by a Java declaration
    – e.g., Seating capacity of a 747
  – Each operation is a Java method
    – e.g., Assigning a flight schedule to a crew member

• Object - instance of a class; something with specific attribute values for which the class’s operations make sense
  – e.g., a specific crew member, a particular plane
Example 1, Airport

Crew Class Attribute
name
home phone
based at
job
specialties

Crew Class Operations
assign to flight number
schedule annual training refresher
takes vacation with startdate, enddate

a Crew object
Jane Doe
888-111-2323
EWR
coopilot
CPR, navigation
Example 2, NIM

Nim Game Attribute  a Nim Game object
Total stones  6

Nim Class Operations
Remove one stone
Remove two stones
Start game with pile of stones
Java Terms

- **Method** - operation consisting of a sequence of instructions
- **Statement** - a complete instruction
- **Identifier** - a name
  - Must begin with a letter
  - No embedded spaces allowed
  - Upper case and lower case distinguished
    - e.g., `takesVacation` and `TakesVacation` are different!
More Java Terms

- **Variable** - a data item of primitive type
  - **boolean** (Boolean) true, false
  - **int** (integer) -1, 0, 5
  - **double** (real number) 2.5, -.03

- Different than objects

- Used as auxiliary values in Java, to do simple calculations and as simple properties of objects
Variables

• **Declaration** creates storage for a variable
  
  ```
  int totalStones;
  int allStones, newpile;
  ```

• **Assignment** associates a value with a variable
  
  ```
  totalStones = 15;
  ```

• **Defined operations**
  – e.g., arithmetic, comparison
  
  ```
  totalStones - 2, totalStones > 20
  ```
Defining Syntax: Integers

<digit>  →  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<integer-number>  →  <digit>

<integer-number>  →  <integer-number>  <digit>

<integer-number>  can be 1,
<integer-number>  can be 2,
<integer-number>  can be 12,
<integer-number>  can be 21, etc.
Defining Syntax

• Bishop, p 20 “An identifier in Java consists of letters and digits, but must start with a letter. Spaces are not allowed and capital and small letters are considered different...”

• Sequence of letters and digits, starting with a letter
A BNF Definition - Identifier

<digit>  →  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<letter>  →  A | a | B | b | C | c | D | d | E | e | F | f | G | g | H | h | I | i | J | j | K | k | L | l | M | m | O | o | P | p | Q | q | R | r | S | s | T | t | U | u | V | v | W | w | X | x | Y | y | Z | z
<identifier>  →  <letter>
<identifier>  →  <letter>  <digit>
<identifier>  →  <letter>  <letter>
<identifier>  →  <identifier>  <letter>
<identifier>  →  <identifier>  < digit>
Identifier

<identifier> → <letter> can be A
<identifier> → <letter> <digit> can be x1
<identifier> → <letter> <letter> can be oK
<identifier> → <identifier> <letter> can be oKA, x1b or AA
<identifier> → <identifier> <digit> can be oKA1 or A123

Rule: have to build the construct by substituting right-hand-side for the nonterminal on the left of the rule.
Derivation of A123b

<identifier> → <identifier> <letter>

→ <identifier> <digit> <letter>

→ <identifier> <digit><digit> <letter>

→ <identifier> <digit> <digit> <digit> <letter>

→ <letter> <digit> <digit> <digit> <digit> <letter>

→ A <digit> <digit> <digit> <digit> <letter>

→ A 1 <digit> <digit> <digit> <letter>

→ A 1 2 <digit> <letter>

→ A 1 2 3 <letter>

→ A 1 2 3 b
Can be 3 rules in BNF, with \( \rightarrow \) read as “produces”,
\[
< \text{output-statement} > \rightarrow \text{System.out.println (} <\text{items}> \text{)};
\]
\[
< \text{output-statement} > \rightarrow \text{System.out.println (} \text{)};
\]
\[
< \text{output-statement} > \rightarrow \text{System.out.print (} <\text{items}> \text{)};
\]

or 1 rule written in shorthand, where \( | \) means “or”,
\[
< \text{output-statement}> \rightarrow \text{System.out.println (} <\text{items}> \text{)}; | \text{System.out.println (} \text{)}; | \text{System.out.print (} <\text{items}> \text{)};
\]
Backus Naur Form (BNF)

- A description language for the “shape” or syntax of programming language constructs
- Consists of terminals, nonterminals, rules
- Each rule corresponds to a block diagram in Bishop text
  - Nonterminal is in top box
  - Choices of right-hand-sides are in bottom box
  - Terminals are in plain font; nonterminals in italics; keywords (which are terminals) in boldface
Backus Naur Form

- Terminals
  - Atomic building blocks of the language
  - Keywords shown in color

- Nonterminals
  - Written as `< nonterminal name >`

- Rules for forming constructs use terminals, nonterminals and constructs we already have formed form other rules

  Nonterminal $\rightarrow$ right-hand-side