CS100M **December 12, 2000**
Final Exam **9:00 AM – 11:30 AM**

_____ _____
(Print last name, first name, middle initial/name) (Student ID)

Statement of integrity: I did not, and will not, break the rules of academic integrity on this exam:

_____
(Signature)

Instructions:

- Read all instructions *carefully*, and read each problem *completely* before starting it!
- This test is closed book – no calculators, reference sheets, or any other material allowed.
- Conciseness, clarity, and style all count. Show all work to receive partial credit, especially box diagrams.
- Carefully comment each loop and major variable.
- If *you* use **`break`** or **`System.exit`** to exit any control structure (except **`switch`**), you will lose points!
- You may **not** alter, supplement, or remove any code that surrounds the blanks and boxes.
- Only **one** statement, expression, modifier, type, or comment per blank!
- Use the backs of pages if you need more space or scrap. You may request additional sheets from a proctor.
- If you supply multiple answers, we will grade only **one**.

Core Points:

1. _____ (30 points) _____

2. _____ (15 points) _____

3. _____ (50 points) _____

4. _____ (75 points) _____

5. _____ (30 points) _____

Total: _____/(200 points) _____

MATLAB Reminders

[] (square brackets), e.g,

```
>> v = [7  12  5]
v =
      7  12  5
```

[] (empty array), e.g.,

```
>> v = []
v =
      []
```

() (parentheses), e.g,

```
>> v([1 2])
ans =
      7  12
```

{} (braces for cell array), e.g.,

```
>> w(2) = {'hi'}
w =
      []    'hi'
```

. (dot), e.g.,

```
>> z.x = 1
z =
      x: 1
```

: (colon), e.g,

```
>> 1:3
ans =
      1  2  3
```

'' (character, string of characters, empty string), e.g.,

```
>> x = 'ab'; x(1)
ans =
a
```

relations (**<**, **>**, **<=**, **>=**, **==**, **~=**),
boolean values (**0**, **1**), and
logical operators (**|**, **&**), e.g,

```
>> 1==0 | 1==1
ans =
      1
>> [1 2 3] == [1 2 4]
ans =
      1  1  0
```

abs (absolute value), e.g.,

```
>> abs([1 -2])
ans =
      1  2
```

any (return **1** if any element in a vector is non-zero), e.g.,

```
>> any([0,0,1,0])
ans =
      1
```

char (character), e.g.,

```
>> char([65 90 97 122])
ans =
      AZaz
```

end (last element of array), e.g.,

```
>> x = 1:4; x(2:end)
ans =
      2  3  4
```

isreal, e.g.,

```
>> ~isreal(i)
ans =
      1
```

linspace (linear space), e.g.,

```
>> linspace(0,2,3)
ans =
      0  1  2
```

mod (modulus, remainder), e.g.,

```
>> mod(4,2)
ans =
      0
```

rand (random number between 0 and 1), e.g.,

```
>> rand
ans =
      0.9501
```

zeros (fill array with value 0),
ones (fill array with value 1),

```
>> zeros(1,4)
ans =
      0  0  0  0
>> ones(1,4)
ans =
      1  1  1  1
```

sum (summation), e.g.,

```
>> sum(1:2)
ans =
      3
```

Problem 1 [30 points] *MATLAB & Java: cell arrays, array of objects, array of arrays, ragged arrays*

An *upper triangle array* is a ragged array that stores the elements above, and including, the main diagonal of an array. For example, the following square array contains an upper triangle array, as shown below:

1 + i	2 - i	0 - i
9 + 7i	-8 - 4i	7 + 0i
0 - 6i	4 + 2i	1 + 3i

square array

1 + i	2 - i	0 - i
	-8 - 4i	7 + 0i
		1 + 3i

upper triangle array

1a) [10 points] Complete the MATLAB function **`Problem1a(size)`** that creates a square array **`x`** of random complex numbers. The real and imaginary components will be integers between -9 and 9, inclusive. **`Problem1a`** will extract the upper triangle array from **`x`** as a sequence of column vectors which will each be stored in cell array **`y`**.

1b) [20 points] Complete the 8 blanks in Java class **`Problem1b`**, which produces a similar result as **`Problem1a`**. The **`main`** method will create two arrays:
- square array **`x`** composed of **`Complex`** numbers with integer components
- upper triangle array **`y`** that contains only references to 1-D **`Complex`** arrays, not values of specific **`Complex`** numbers. Hint: Use a ragged, column-major array for **`y`**.

Your code will store the upper triangle array of **`x`** in **`y`**, which is output to the user.

See below for example sessions for a 3×3 array:

Sample MATLAB Session	Sample Java Session

`>> Problem1a(3)`
`ans =`
`  -8.0000 - 5.0000i`
`ans =`
`   3.0000 + 0.0000i`
`   9.0000 + 4.0000i`
`ans =`
`  -9.0000 + 5.0000i`
`  -3.0000 + 4.0000i`
`        0 - 2.0000i`
	`> java Problem1b 3`
	`ans =`
	`   7+3i`
	`ans =`
	`  -9-1i`
	`   0+9i`
	`ans =`
	`  -3+5i`
	`  -6+6i`
	`   2+0i`

```
% Part 1a: Use MATLAB
function Problem1a(size)
% Display upper triangle array
% Create square array of dimension $size$ with random complex numbers:
  x = floor(rand(size)*19)-9+floor(rand(size)*19*i)-9*i;
% Preallocate a cell array:
  y = cell(1,size);
% Store the upper triangle of $x$ as column vectors in elements of $y$:
   for jj = ______ : ______

       y{ _________________ } = _________________ ; % content indexing of $y$
   end
% _____________________________________________________ (describe the statement below)
  y{:}
```

```
// Part 1b: Use Java
public class Problem1b {
   public static void main(String[] args) {

      final int SIZE = Integer.parseInt(_______); // convert 1st command-line argument to int
      Complex[][] x = makeSquare(SIZE);           // assign square Complex array
      Complex[][] y = makeTriangle(SIZE);         // assign "empty" triangle Complex array
      fillTriangle(x,y);                          // fill $y$ with triangle from $x$
      printTriangle(y);                           // display $y$
   }
   // Print upper triangle array column by column to resemble MATLAB's output:
      public static void printTriangle(Complex[][] y) {

         for(int col=0 ; ____________________________________________ ; col++) {
            System.out.println("ans =");

            for(int row=0 ; ____________________________________________ ; row++)

               System.out.println("  " + ____________________________________________________ );
         }
      }
   // Fill elements of upper triangle array $y$ which is extracted from square array $x$:
      public static void fillTriangle(Complex[][] x, Complex[][] y) {

         for(int col=0 ; ____________________________________________ ; col++)

            for(int row=0 ; ____________________________________________ ; row++)

               ____________________________________________________ ;
      }
   // Create and return "empty" Complex triangle array as a ragged, column-major array:
      public static Complex[][] makeTriangle(int size) {
         Complex[][] tmp = new Complex[size][];
            for(int col=0 ; col < size ; col++)

               ____________________________________________________ ;
         return tmp;
      }
   // Create and return a Complex square array:
      public static Complex[][] makeSquare(int size) {
         Complex[][] tmp = new Complex[size][size];
         for(int row=0 ; row < size ; row++)
            for(int col=0 ; col < size ; col++)
               tmp[row][col] = crand(9,19);
         return tmp;
      }
   // Return random Complex number:
      public static Complex crand(int low, int high) {
         return new Complex(rrand(low,high),rrand(low,high));
      }
   // Return random Complex integer:
      public static int rrand(int low, int high) {
         return (int) Math.floor(Math.random()*high)-low;
      }
} // Class Problem1b

class Complex {
   private int real;  // real component
   private int imag;  // imaginary component
   Complex(int real, int imag) { /* code not shown */ }
   public String toString() { /* code not shown */ }
} // Class Complex
```

Problem 2 [15 points] *Java: inheritance, "butterfly ballots"*

For each of the following 15 statements and/or lines of code, indicate whether or not it will cause a standard Java compiler to issue an exception or error by selecting either:

- Incorrect: causes error or exception
- Correct: compiler does not complain

```
class File {
   int id;
   protected int size;
   private int author_id;
   public File() { }
   public File(int i, int s, int a) { id=i; size=s; author_id=a; }
   private void useless1() { }
   protected void useless2() {
      useless1();  //.....................................1. Correct [ ]  Incorrect [ ]
   }
   public void whoDidIt() {
      System.out.println("Author "+author_id); //.............2. Correct [ ]  Incorrect [ ]
   }
} // Class File

class ImageFile extends File {
   private boolean color;
   public static int numColors=2;
   public ImageFile() { }
   public ImageFile(int i, int s, int a, boolean c) {
      super(i,s,a);
      color = c;
   }
   public void whoDidIt() {
      System.out.println("Artist "+author_id); //.............3. Correct [ ]  Incorrect [ ]
   }
   public static void strange1() {
      numColors++; //.....................................4. Correct [ ]  Incorrect [ ]
   }
   public void strange2() {
      numColors++; //.....................................5. Correct [ ]  Incorrect [ ]
      int twice = size*2; //..............................6. Correct [ ]  Incorrect [ ]
      super.useless1(); //................................7. Correct [ ]  Incorrect [ ]
   }
} // Class ImageFile

public class Problem2 {
   int size=3;
   public static void main(String[] args) {
      int tmp;
      File x = new File(1,1000,1013);
      tmp = x.id; //.....................................8. Correct [ ]  Incorrect [ ]
      tmp = size; //.....................................9. Correct [ ]  Incorrect [ ]
      tmp = x.author_id; //.............................10. Correct [ ]  Incorrect [ ]
      ImageFile pic = new ImageFile(101,9000,1234,true); //....11. Correct [ ]  Incorrect [ ]
      pic.useless1(); //................................12. Correct [ ]  Incorrect [ ]
      tmp = pic.numColors; //...........................13. Correct [ ]  Incorrect [ ]
      x = new ImageFile(); //...........................14. Correct [ ]  Incorrect [ ]
      pic = new File(); //..............................15. Correct [ ]  Incorrect [ ]
   }
} // Class Problem2
```

Problem 3 [50 points] *MATLAB: strings, characters*

Overview: Complete the following subfunctions of **`Problem3`** which models a word-guessing game. Starting with a "bag" of 104 letters ranging from A to Z, the user picks 7 random letters and attempts to form the longest possible English word using only those 7 letters. The user gets 1 point per letter used.

Description: The game activates from MATLAB's command window and has the following functions:

- **`Problem3`**, which is the driver that plays "1 round" of the game by initializing the data, filling the "bag," selecting the 7 letters, prompting the user to enter a word, collecting the points, and reporting the results. The results consist of the points earned, the remaining unused letters from the current round, and the count of letters remaining in the "bag".
- **`fillBag`**, which represents the full collection of all letters as a string initially stored in array **`bag`**.
- **`pickLetters`**, which randomly picks enough **`letters`** to "fill" the number of current openings represented by **`count`**. After removing **`letters`** from **`bag`**, **`pickLetters`** returns both the revised **`bag`** and chosen **`letters`**.
- **`enterWord`**, which prompts the user to enter a legal word **`my_word`**. Once the user enters a legal word, **`enterWord`** awards one point per letter as **`points`**, which is returned along with **`my_word`**.
- **`legal`**, which checks if a **`word`** contains **`letters`** from the original hand, does not exceed the number of total **`letters`**, and if it **`isEnglish`**.
- **`isEnglish`**, which checks if a dictionary contains **`word`**.
- **`getRemaining`**, which returns as a string the **`letters`** *not* used by **`word`** from the current round.
- **`removeOccurences`**, which takes an input string **`lettersToRemove`** and attempts to remove that string from the other input string **`word`**.
- **`findstr`**, which returns the starting indices of any occurrences of the shorter of the two strings in the longer.

Hint: You can use **`removeOccurences`** to greatly simplify your code for **`legal`** and **`getRemaining`**.

Sample Session:

```
>> Problem3
Number of letters in bag: 104
Your letters are: SIHALID
Form a word: HAIL
Final points: 4
Leftover letters: DIS
Remaining letters in bag: 97
```

```
function Problem3
% Initialize data for one game:
  count = 7;      % maximum letters to choose for each game
  bag = fillBag;  % form initial set of letters and store in $bag$
  points = 0;     % initial $points$
  disp(['Number of letters in bag: ',num2str(length(bag))]);

% Run one game:
  [my_letters,bag] = pickLetters(count,bag);      % grab $count$ letters from $bag$
  [points,my_word] = enterWord(count,my_letters); % get $my_word$ from user and award $points$
  remain = getRemaining(my_word,my_letters);      % store unused letters in $remain$

% Report results:
  disp(['Points for the word: ', num2str(points)]);
  disp(['Leftover letters: ', num2str(remain)]);
  disp(['Remaining letters in bag: ', num2str(length(bag))]);

function bag = fillBag()
% Store initial collection letters "in the bag":
  A2F = 'AAAAAAAABBBBCCCCDDDDEEEEEEEEEEEEEEFFFF';  % letters from 'A' to 'F'
  G2P = 'GGGHHHHIIIIIJJKKLLLLMMMMNNNNOOOOOOPPPP';  % letters from 'G' to 'P'
  Q2Z = 'QRRRRSSSSTTTTUUUVVVVWWWXYYYZ';            % letters from 'Q' to 'Z'

  bag = _____________________________________ ;  % concatenate strings to fill $bag$
```

[Problem 3 continues on next page]

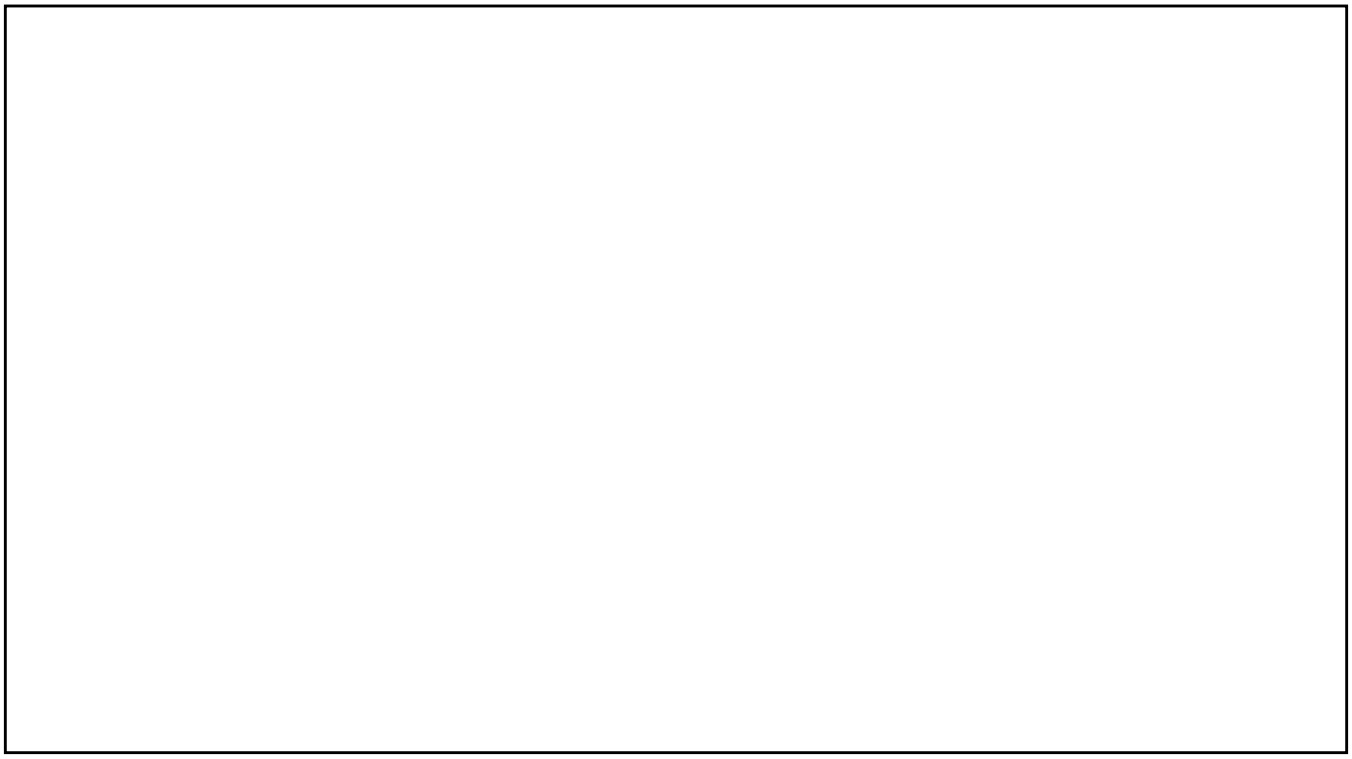
```
function [letters,bag]=pickLetters(count,bag)
% Extract $count$ $letters$ from $bag$. The user will pick from these $letters$:
  letters = '';                                  % initial array of $letters$ is empty
  for ii=1:count
     index = 1+floor(rand*(length(bag)));        % pick random $index$ to get random letter
     letters(ii) = ____________________ ;        % extract random letter using $index$
     % Update $bag$ by assigning $bag$ to an array without the extracted letter:
       bag = bag( ____________________________________________________ );
  end

function [points,my_word] =  enterWord(count,my_letters)
% Prompt user to enter word from current selection of letters:
  disp(['Your letters are: ',my_letters]);       % display letters user chooses from
  my_word = upper(input('Form a word: ','s'));   % prompt user to enter a word as a string
  while ____________________________________      % check legality of user's input word
     disp('The word you entered is not valid!');  % bad user!
     my_word = upper(input('Form a word: ','s')); % user enters another word
  end
  points = ____________________________ ;        % award 1 point per letter for $my_word$

function testValue = legal(letters,word)
% Check if $word$ contains characters that appear in $letters$, does not exceed the length of
% $letters$, and if it is English:
```

[Problem 3 continues on next page]

```
function remaining = getRemaining(word,letters)
% Find a string of characters not used by $word$ in the string $letters$. Return the
% leftover letters in the string $remaining$:
```



```
function flag = isEnglish(word)
% Return $1$ if $word$ is English
% (code not shown)

function word = removeOccurences(lettersToRemove, word)
% Remove letters contained in $lettersToRemove$ from $word$:
  for letterIndex = 1:length(lettersToRemove)
     removeIndexArray = findstr(word, lettersToRemove(letterIndex));
     if (~isempty(removeIndexArray))
        removeIndex = removeIndexArray(1);
        word = [word(1:removeIndex-1), word(removeIndex+1:end)];
     end
  end

function indicies=findstr(s1,s2)
% INDICIES = FINDSTR(S1,S2) returns the starting indices of any occurrences of the
% shorter of the two strings in the longer. Examples:
% >> s = 'abcd';
% >> findstr(s,'b') -> returns 2
% >> findstr(s,'x') -> returns []
% (code not shown)
```

[This space is intentionally left blank.]

Problem 4 [75 points] *Java: OOP, arrays of objects, simulation*

Problem: Using Java, complete the following blanks according to the following model.

Model: A program can simulate a group of **Worker**s removing **boxes** from **Tray**s on a conveyor **Belt** moving left to right. An equal number of **Tray**s appear for an equal number of **Worker**s. The simulation has all **Tray**s start empty in front of each **Worker**. In the first **run**, all **Tray**s move right one space to the next **Worker** and a new **Tray** appears with random **boxes** for the first (leftmost) **Worker**. Each **Worker** attempts to remove as many **boxes** as possible from her/his **Tray**. Because the **Worker**s get tired from moving **boxes**, their **efficiency** decreases for each **run**. If the last **Worker** (rightmost) fails to remove all contents of her/his **Tray**, the **Belt** stops. If no **boxes** remain, the simulation continues with a new **run** by repeating the process of shifting all **Tray**s to the right, giving the first **Tray** new **boxes**, and having the **Worker**s extract **boxes** from their current **Tray**.

Hints: A portion of the output for a simulation with four **Tray**s and **Worker**s has the following form:

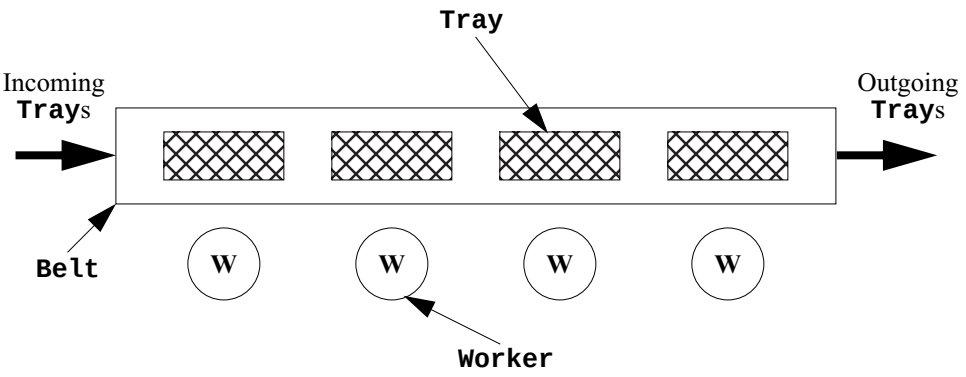
Portion of sample session:

```
Current run: 13
Old contents: 3 0 1 1
New contents: 1 0 1 0
Total boxes so far: 27
Press Enter to continue.

Current run: 14
Old contents: 5 1 0 1
New contents: 3 1 0 1
Total boxes so far: 29
Press Enter to continue.

The Workers are overloaded!
Leftovers: 1
```

Depiction of top view of simulation:




```
class Worker {
   private double efficiency;           // Worker efficiency initially ranges from 75% to 100%
   private final double FACTOR = 1.05;  // efficiency reduction factor
   // Construct a new Worker and choose a random $efficiency$:
      public Worker() { setEfficiency(); }
   // Set a random $efficiency$:
      private void setEfficiency() { efficiency = Math.random()*(1.00-0.75)+0.75; }
   // Reduce current Worker's $efficiency$:
      public void changeEfficiency() { efficiency /= FACTOR; }
   // Extract boxes from input Tray and update Tray's amount of boxes:
      public int extractBoxes(Tray tray) {
      // If Tray is empty, extract nothing:
         if ( ____________________________________ ) {
         // Based on efficiency, Worker extracts from $tray$ a portion of its boxes:
            int boxes = (int) Math.round(efficiency * ____________________________________ );
         // Remove boxes from current Tray:
            ____________________________________________________________________________ ;
         // Return amount of extracted boxes:
            return ____________________________________________________________________ ;
         }
      // Otherwise, return no boxes because Tray is empty:
         return _________________ ;
      }
} // Class Worker
```

```
class Tray {
   private final int MIN = 0;     // minimum amount of $boxes$ stored on a Tray
   private final int MAX = 5;     // maximum amount of $boxes$ stored on a Tray
   private int boxes;             // amount of boxes on current Tray
   // Construct an "empty" Tray:
      public Tray() { }
   // Return the number of $boxes$ on current Tray:
      public int currentBoxes() { return boxes; }
   // Reduce the number of $boxes$ on current Tray by input $r$:
      public void removeBoxes(int r) { if (!isEmpty()) boxes=boxes-r; }
   // Store a random amount of $boxes$ on the current Tray:
      public void fillTray() { boxes = (int) Math.floor(Math.random()*(MAX-MIN+1))+MIN; }
   // Return a Boolean value depending on whether or not current Tray is empty:
      public boolean isEmpty() { return (boxes == 0); }
} // Class Tray
public class Problem4 {
   // Welcome and start simulation:
      public static void main(String[] args) {
         System.out.print("Welcome to BELT! Enter a size: ");
         TokenReader in = new TokenReader(System.in);
         int size = in.readInt(); // number of Workers (also equal to number of Trays)
         doSimulation(size,in);  // start simulation
      }
   // Perform simulation:
      public static void doSimulation(int size, TokenReader in) {
      // Initialize arrays and other values:
         Worker[] workers = new Worker[size]; // create array to hold Workers
         Tray[] trays = new Tray[size];       // create array to hold Trays
         for(int i=0;i<size;i++) {
            workers[i]=new Worker();
            trays[i]=new Tray();
         }
         Belt belt = new Belt(workers,trays); // create $belt$ which has Workers and Trays
         int run = 1;                         // runs of the simulation so far
      // Perform runs of simulation until last Tray has leftover boxes:
         while( ____________________________________________________________________ ) {
            System.out.println("Current run: "+ ________ ); // increment $run$
            ____________________________________________ ; // perform one run of simulation
            in.waitUntilEnter();                            // pause before next run
         }
      // Report results:
         System.out.println("The Workers are overloaded! Stopping simulation.");
         System.out.println("Leftovers: "+ ____________________________________________ );
      }
   // Perform one run of the simulation where all Workers extract boxes from Trays:
      public static void doOneRun(Belt b) {
         ____________________________________________ ; // shift Trays to the right
         ____________________________________________ ; // refill first Tray
         ____________________________________________ ; // display original contents of Trays
         ____________________________________________ ; // extract boxes from Trays
         ____________________________________________ ; // display new contents of Trays
         ____________________________________________ ; // reduce Workers' efficiencies
         ____________________________________________ ; // report results of current $run$
   }
} // Class Problem4
```

[Problem 4 continues on next page]

```
class Belt {
   private int totalBoxes;    // total number of boxes extracted during all runs in simulation
   private Worker[] workers;  // Workers that extract boxes from the Trays
   private Tray[] trays;      // Trays carried on the current Belt (Trays carry boxes)

   // Construct a new Belt object which contains Workers and Trays:
      Belt(Worker[] workers, Tray[] trays) {
         ____________________________________________________ ; // assign $workers$
         ____________________________________________________ ; // assign $trays$
      }

   // Shift Trays to the next Worker on the right. Hint: loop from right to left!
      public void shiftTrays() {
         for(int i = ________________________________ ; ____________________________________ ; _____ )
            ____________________________________________________________________________________ ;
      }

   // Refill first (leftmost) Tray with new amount of boxes:
      public void refillFirstTray() {
         ____________________________________________ ; // create new first Tray
         ____________________________________________ ; // fill Tray with boxes
      }

   // Extract boxes from all Trays: Each Worker takes boxes from Tray in front of him/her:
      public void extractContents() {
         for(int i = ______ ; ____________________________________ ; _____ )
            // Increment $totalBoxes$:
               ____________________________________________________________________________ ;
      }

   // Print current contents of all Trays, using $age$:
      public void printContents(String age) {
         System.out.print(age+" contents: ");
         for(int i = ______ ; ____________________________________ ; _____ )
            System.out.print( ____________________________________________________________ );
         System.out.println();
      }

   // Decrease each Worker's efficiency for next run:
      public void decreaseEfficiency() {
         for(int i = ______ ; ____________________________________ ; _____ )
            ____________________________________________________________________________________ ;
      }


   // Report results of the simulation for all runs:
      public void report() {
         System.out.println("Total boxes so far: "+ ____________________________________ );
      }

} // Class Belt
```

Problem 5 [30 points] *Java: algorithms, searching, efficiency*

Background: You will complete class **`Game`** to play an "automatic" number-guessing game. Unlike a game played by a user, a Java program will play against itself by picking a random number and then attempting to guess it. You will choose how the program searches for the random number in your solution.

Problem: Main Class **`Problem5`** activates a new **`Game`**. Complete class **`Game`** which has the following members:

- instance variables **`target`** (the program's randomly picked number); **`guess`** (the program's guess of **`target`**); **`count`** (number of **`guess`**es to reach **`target`**); **`LOW`** and **`HIGH`** (range of **`target`**); **`STOP`** (maximum number of **`guess`**es the program will attempt to reach **`target`**).
- constructor **`Game`** that picks a random number **`target`**, searches for that **`target`**, and reports the results of the **`Game`**.
- method **`reportGame`** that reports the results of the current **`Game`** based on whether or not the program found **`target`**.
- method **`pickTarget`** that returns a random integer between **`0`** and **`100`**, inclusive.
- method **`guessForTarget`** that searches for the value of **`target`**. See below for suggestions of algorithms.

Algorithm: You must decide the most efficient algorithm without resorting to the assignment **`guess=target`**. For partial credit, you could simply check every number between **`LOW`** and **`HIGH`**, but you can do better. In fact, one algorithm can guess **`target`** in 7 or less guesses, guaranteed!

Hints: Java needs to guess *integers*. Since it is using **`Math.random()`**, you might need to use one or all of the following:

- **`(int) number`** to convert **`number`** to an integer
- **`Math.floor(number)`** to remove the decimal portion of **`number`**
- **`Math.ceil(number)`** to round **`number`** up for a decimal value higher than zero.

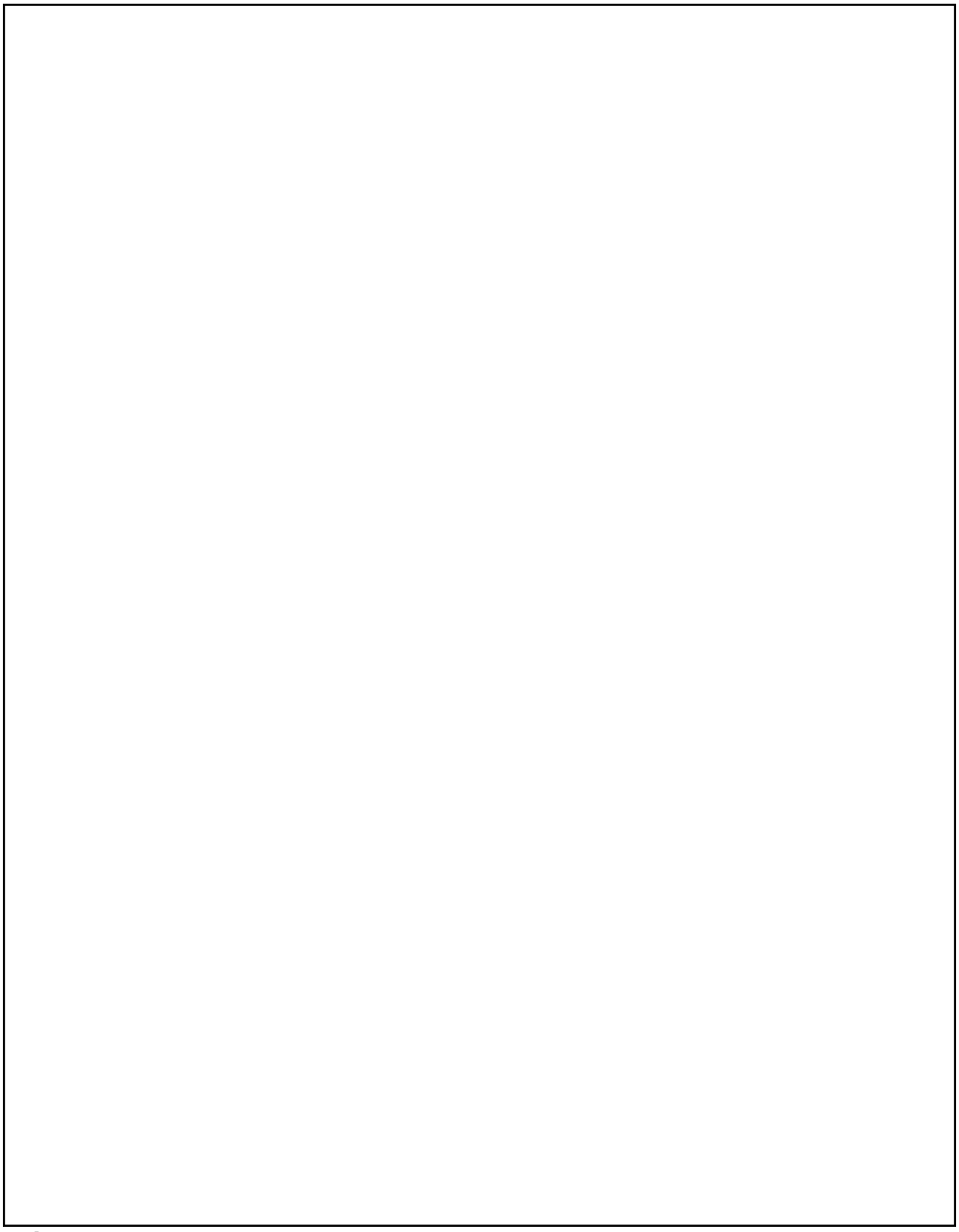
For example, **`(int) 4.0`** returns the integer **`4`**, **`Math.floor(4.12)`** yields **`4.0`**, and **`Math.ceil(4.01)`** yields **`5`**.

```
public class Problem5 {
    public static void main(String[] args) {
       new Game();
   }
} // Class Game

class Problem5 {
   private int target;                    // randomly chosen number
   private int guess;                     // current guess of $target$
   private int count;                     // number of guesses to reach $target$
   private final int LOW=0;               // lowest value of $target$
   private final int HIGH=100;            // highest value of $target$
   private final int STOP=HIGH-LOW;       // maximum number of guesses allowed

   // Construct a $Game$ object:
      public Game() {
         target = guessNumber();  // choose and set a random number $target$
         guessForTarget();        // search for $target$
         reportGame();            // report results of current game
      }
   // Report results of game:
      private void reportGame() {
         System.out.println(guess);
         if ( guess == target )
            System.out.println("Yes! Java took "+count+" guesses to find "+target);
         else
            System.out.println("Java couldn't find it. Boo for Java!");
      }
   // Return a random integer between $LOW$ and $HIGH$, inclusive:
      private int pickTarget() {
         return (int) Math.floor(Math.random()*(HIGH-LOW+1))+LOW ;
      }
```

```
    // Search for the value of $target$. Stop when $guess$==$target$:
       private void guessForTarget() {
```



```
       }
} // Class Game
```