CS 112 Introduction to Programming

Review-II: Data Conversion, for Loop, Variable Scoping, Parameterized Methods, StdDraw

Sean Guo, Qiao Xiang, Yang (Richard) Yang
Computer Science Department
Yale University
208A Watson, Phone: 432-6400
Email: cs112ta@cs.yale.edu, yry@cs.yale.edu
Review Basic Concepts
Basic Concepts

- Data type conversion
  - Implicit (automatic/predefined) data conversion
  - Explicit (cast) data conversion
- Assignment as operator
- for loops
  - Map loop# to target patterns
  - Top-down decomposition
- Variable scoping
- Parameterized methods
- Java graphics: StdDraw
Basic Concepts

- Data type conversion
  - Implicit (automatic/predefined) data conversion
  - Explicit (cast) data conversion

- Assignment as operator

- for loops
  - Map loop# to target patterns
  - Top-down decomposition

- Variable scoping

- Parameterized methods

- Java graphics: StdDraw
Data Conversion

- Data conversion is the conversion of data from one type to another, e.g.,
  - an int -> a double,
  - a double -> an int,
  - an int -> a string

- Java data conversion is per-operator, occurring when the operator is evaluated according to the precedence rule

- Java has two types of data conversion:
  - Implicit (automatic/predefined) data conversion
  - Explicit (cast) data conversion
Example: Mixed Arithmetic Expression

\[
2.5 + \frac{10}{3} \times 2.5 - \frac{6}{4} \\
2.5 + 3 \times 2.5 - \frac{6}{4} \\
2.5 + 7.5 - \frac{6}{4} \\
2.5 + 7.5 - 1 \\
10.0 - 1 \\
9.0 \text{ (not 9!)}
\]
Java String Concatenation Conversion: Examples

1 + "abc" + 2 is "1abc2"
"abc" + 1 + 2 is "abc12"
1 + 2 + "abc" is "3abc"
"abc" + 9 * 3 + 1 is "abc271"
4 - 1 + "abc" is "3abc"
Type Casting Examples

double result = (double) 19 / 5;  // 3.8

int result2 = (int) result;  // 3

double x = (double) 1 + 1 / 2;  // 1.0

double y = 1 + (double) 1 / 2;  // 1.5
Basic Concepts

- Data type conversion
  - Implicit (automatic/predefined) data conversion
  - Explicit (cast) data conversion

- Assignment as operator

- for loops
  - Map loop# to target patterns
  - Top-down decomposition

- Variable scoping

- Parameterized methods

- Java graphics: StdDraw
Update vs. Algebra

What happens here?

```cpp
int items = 3;
items = items + 1; // ???
```

In programming language, assignment is just an operator, with a lower precedence than the arithmetic operators: the right is assigned to the left var.
public class Ruler {
    public static void main(String[] args) {
        String ruler = "1";
        ruler = ruler + " 2 " + ruler;
        ruler = ruler + " 3 " + ruler;
        ruler = ruler + " 4 " + ruler;
        System.out.println(ruler);
    }
}

% java Ruler
1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

"1"
"1 2 1"
"1 2 1 3 1 2 1"
Basic Concepts

- Data type conversion
  - Implicit (automatic/predefined) data conversion
  - Explicit (cast) data conversion

- Assignment as operator

- for loops
  - Map loop# to target patterns
  - Top-down decomposition

- Variable scoping

- Parameterized methods

- Java graphics: StdDraw
The for Statement: Syntax

Reserved word

The *initialization* portion is executed once before the loop begins

The statement is executed until the *condition* becomes false

```
for ( initialization; condition; increment )
statement;
```

Both semi-colons are always required

The *increment* portion is executed at the end of each iteration
Flowchart of a for loop

for ( initialization ; condition ; increment )
statement;

initialization

condition evaluated

ture
false

statement

increment
Basic Concepts

- Data type conversion
  - Implicit (automatic/predefined) data conversion
  - Explicit (cast) data conversion
- Assignment as operator
- for loops
  - Map loop# to target patterns
  - Top-down decomposition
- Variable scoping
- Parameterized methods
- Java graphics: StdDraw
Variable Scope

- **Scope**: The part of a program where a variable exists.

- **Basic rule**: from its declaration to the end of the enclosing `{ }` braces

- **Examples**
  - A variable declared in a `for` loop exists only in that loop.
  - A variable declared in a specific method exists only in that method.
  - A variable declared not inside any method but in a class is said to have class scope.
public class CountSum {
    static int N = 10;
    public static void main(String[] args) {
        countSum();
    }

    public static void countSum() {
        System.out.print("T-minus ");
        int sum = 0;
        for (int i = 1; i <= N; i++) {
            System.out.println(i);
            sum += i;
        }
        System.out.println("N: " + N);
        System.out.println("Sum: " + sum);
    } // end of countSum
} // end of class
Basic Concepts

- Data type conversion
  - Implicit (automatic/predefined) data conversion
  - Explicit (cast) data conversion
- Assignment as operator
- for loops
  - Map loop# to target patterns
  - Top-down decomposition
- Variable scoping
- Parameterized methods
- Java graphics: StdDraw
Method Parameterization

- Specify a parameter to control the behavior of a method
  - Methods with parameters solve an entire class of similar problems

- Redundancy removal/abstraction through generalization
  - The more general a building block, the easier to reuse it
  - We will learn more techniques on generalization/abstraction
Parameterization

- **parameter**: A value passed to a method by its caller, e.g.,
  - When *declaring* a method, we will state that it requires a parameter for the number of spaces.
  - When *calling* the method, we will specify the number.
Syntax: Declaring a Method with a Parameter

public static void <method_name>(<type> <param_name>) {
   <statement>(s);
}

- The parameter is called the formal argument

- Example:

```java
public static void chant(int times) {
   for (int i = 1; i <= times; i++) {
      System.out.println("Just a salad...");
   }
}
```
Basic Concepts

- Data type conversion
  - Implicit (automatic/predefined) data conversion
  - Explicit (cast) data conversion
- Assignment as operator
- for loops
  - Map loop# to target patterns
  - Top-down decomposition
- Variable scoping
- Parameterized methods
- Java graphics: StdDraw
Coordinate system

- You still set canvas size using numbers of pixels:
  `setCanvasSize(int w, int h)`
- But \((x, y)\) position is the coordinate in a normalized coordinate system
  - \([0, 1]\) as default
  - `setXScale(double x0, double x1)` to set x range
  - `setYScale(double y0, double y1)`
- Position \((0, 0)\) is at the window's lower-left corner.
  - \(x\) increases rightward and the \(y\) increases upward.

Example method rectangle:

- `rectangle(double cx, double cy, double halfWidth, double halfHeight);`
Example: StdDraw X

- Implement SimpleStdDrawX.java to draw an X using StdDraw
  - Try two approaches:
    1. Use the normalized coordinate system (0,0) -> (1,1)
    2. Set the Xscale and and Yscale
Exercise 1: Nested Loop
Exercise

What nested for loops produce the following output?

inner loop (repeated characters on each line)

outer loop (loops 5 times because there are 5 lines)
Exercise

What nested for loops produce the following output?

```
....1
...2
..3
 .4
 5
```

inner loop (repeated characters on each line)

outer loop (loops 5 times because there are 5 lines)

We must build multiple complex lines of output using:
- an outer "vertical" loop for each of the lines
- inner "horizontal" loop(s) for the patterns within each line
Outer and inner loop

- First write the outer loop, from 1 to the number of lines.
  
  ```java
  for (int line = 1; line <= 5; line++) {
    ...
  }
  ```

- Now look at the line contents. Each line has a pattern:
  - some dots (0 dots on the last line), then a number
    
    ```
    ...1
    ...2
    ..3
    .4
    5
    ```

  - Observation: the number of dots is related to the line number.
Nested for loop solution

- **Answer:**
  ```java
  for (int line = 1; line <= 5; line++) {
      for (int j = 1; j <= (-1 * line + 5); j++) {
          System.out.print("");
      }
      System.out.println(line);
  }
  ```

- **Output:**
  ```
  ....1
  ...2
  ..3
  .4
  5
  ```
Nested for loop exercise

What is the output of the following nested for loops?

```java
int N = 5;
for (int line = 1; line <= N; line++) {
    for (int j = 1; j <= (-1 * line + N); j++) {
        System.out.print(\".\");
    }
    for (int k = 1; k <= line; k++) {
        System.out.print(line);
    }
    System.out.println();
}
```

Answer:

```
....1
...22
..333
.4444
55555
```
Practice: Nested for loop exercise

- Write code to produce diagonal matrix:

  ....1
  ...2.
  ..3..
  .4...
  5.....
Write code to produce diagonal matrix:

```java
int N = 5;
for (int line = 1; line <= N; line++) {
    for (int j = 1; j <= (-1 * line + N); j++) {
        System.out.print(1);
    }
    System.out.print(line);
    for (int j = 1; j <= (line - 1); j++) {
        System.out.print(1);
    }
    System.out.println();
}
```
Review DrawX
Nested Loop Design Example: Drawing A Complex Pattern

- Use nested `for` loops to draw ASCII X
- A size SIZE ASCII X has 2 * SIZE rows
- Why draw ASCII art?
  - Real graphics will require some finesse (shortly)
  - ASCII art has complex patterns
  - Can focus on the algorithms
Final Pseudo Code

1. Bound
   • == , (SIZE - 2)*2 spaces, ==

2. for line = 1 to SIZE -1
   line spaces
   \
   2 (SIZE - 2) + 2 - 2 * line spaces
   /

3. for line = 1 to SIZE - 1
   SIZE - line spaces
   \
   (line - 1) * 2 spaces
   \

4. Bound
Exercise: Implement a Method that takes a string and a number, and print the string number of times.
Exercise 2: Parameterized Methods
**Exercise: Box Figures**

- Design Boxes.java to print the following boxes:

```
******
*   *
******

************
*           *
************

******
*   *
*   *

******
```
Summary: Box Figures Chaining

```java
public static void main(String[] args) {
    drawBox(5, 3);
    drawBox(10, 3);
    drawBox(5, 4);
}
```

```java
public static void drawBox(int width, int height) {
    // step 1. print width of "*
    repeat(" ", width);
    System.out.println();

    // step 2. loop a total of height - 2 times
    // for each row:
    // print *
    // print width -2 " 
    for (int i = 1; i <= height - 2; i++) {
        System.out.print("*");
        repeat(" ", width - 2);
        System.out.println("*");
    }

    // step 3. print width of "*
    repeat(" ", width);
    System.out.println();
}
```

```java
public static void repeat(String s, int times) {
    for (int i = 1; i <= times; i ++) {
        System.out.print(s);
    }
}
```
Procedural Heuristics

1. The main method should read as a concise summary of the overall set of tasks performed by the program.

2. Each method should have a clear set of responsibilities.

3. No method should do too large a share of the overall task.

4. Use method with parameters to remove redundancy with generalization, but do not over generalize.

5. Data should be declared/used at the lowest level possible (localized data).
Method Exercise

- Exercise: Change the DrawMatrix program to use a parameterized method for different sizes.

DrawMatrixMethods.java
Exercise 3: DrawMirror
Exercise: Drawing A Complex Pattern

- Use nested for loops to produce the following output

- Why draw ASCII art?
  - Real graphics require some finesse
  - ASCII art has complex patterns
  - Can focus on the algorithms

```
#================#
|      <><>      |
|    <>....<>    |
|  <>........<>  |
|<>............<>|
|<>............<>|
|  <>........<>  |
|    <>....<>    |
|      <><>      |
#================#
```
Pseudo-code algorithm

1. Line
   • #, 16 =, #

2. Top half
   • |
     • spaces (decreasing)
     • <>
     • dots (increasing)
     • <>
     • spaces (same as above)
     • |

3. Bottom half (top half upside-down)

4. Line
   • #, 16 =, #

#=================================#
|      <><><>      |
|    <>.....<>    |
|  <>........<>  |
|<>............<>|
|<>............<>|
|  <>........<>  |
|    <>.....<>    |
|      <><><>      |
#=================================#
2. Tables

- A table for the top half:
  - Compute spaces and dots expressions from line number

<table>
<thead>
<tr>
<th>line</th>
<th>spaces</th>
<th>line * -2 + 8</th>
<th>dots</th>
<th>4 * line - 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>

#=================================#

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#=================================#
public class Mirror {
    public static void main(String[] args) {
        line();
        topHalf();
        bottomHalf();
        line();
    }

    public static void topHalf() {
        for (int line = 1; line <= 4; line++) {
            // contents of each line
        }
    }

    public static void bottomHalf() {
        for (int line = 1; line <= 4; line++) {
            // contents of each line
        }
    }

    public static void line() {
        // ...
    }
}
3. Writing the code

- Useful questions about the top half:
  - What methods? (think structure and redundancy)
  - Number of (nested) loops per line?
// Prints the expanding pattern of <> for the top half of the figure.
public static void topHalf() {
    for (int line = 1; line <= 4; line++) {
        System.out.print("|");
        for (int space = 1; space <= (line * -2 + 8); space++) {
            System.out.print(" ");
        }
        System.out.print("<>");
        for (int dot = 1; dot <= (line * 4 - 4); dot++) {
            System.out.print(".");
        }
        System.out.print("<>");
        for (int space = 1; space <= (line * -2 + 8); space++) {
            System.out.print(" ");
        }
        System.out.println("|");
    }
}
Exercise 4: Parameterized Drawings
Exercise: Book Cover
(Color and Loop)

- White 500x600 drawing panel
- Three components at
  - (20, 415), (165, 330), (220, 85) with sizes 150, 120, and 240
  - Each component
    - Yale blue background
    - white “CS112" text left @ 1/2 horizontal, 4/5 vertical
    - 10 brown (red=192, green=128, blue=64) “bricks”
      - 2 pixel between two adjacent bricks

BookCover.java