CS 112 Introduction to Programming

Variables;
Type Casting;
Using Variables in for Loops

Yang (Richard) Yang
Computer Science Department
Yale University
308A Watson, Phone: 432-6400
Email: yry@cs.yale.edu

Admin

- PS2 questions?
- Interest in an Informal coding style session?
- Informal CS112 lunch together at Commons
Recap: Data Types

- Why data types?
  - Define the data representation, allowed operations, and semantics of operations (e.g., 1 / 2 vs 1.0 / 2.0)

- Java is a strong typed language: every variable, every literal has a type
  ```java
  int nA;
  nA = 4;
  int nB = 1;
  int total = nA * 4 + nB * 3;
  System.out.println( total / (nA + nB) );
  double GPA = 3.0 + 0.8;
  char lastNameInitial = 'Y';
  ```

Recap: Mixed Type

- Why mixed types?
  - Desired result in a different type
  - Natural expression, 4 / 8.0; “3” + 1

- Mix-type operations
  - Java tries a set of predefined conversion rules, e.g.,
    - numerical promotion
  - Conversion is per operator
Example: Mixed Arithmetic Expression

```
2.5 + 10 / 3 * 2.5 - 6 / 4
  2.5 + 3 * 2.5 - 6 / 4
    2.5 + 7.5 - 6 / 4
      2.5 + 7.5 - 1
        10.0 - 1
          9.0 (not 9!)
```

Practice: Mixed Arithmetic Expression

```
7 / 3 * 1.2 + 3 / 2
  2 * 1.2 + 3 / 2
    2.4 + 3 / 2
      2.4 + 1
        3.4
```
Data Conversion Rule: Numeric to Java String

- Occurs automatically when one operand is a number and the other a string in the “+” operator.

- The conversion is per-operator, affecting only its operands.

- This produces the convenient string concatenation operation.

Java String Concatenation Conversion: Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 + &quot;abc&quot; + 2</td>
<td>&quot;1abc2&quot;</td>
</tr>
<tr>
<td>&quot;abc&quot; + 1 + 2</td>
<td>&quot;abc12&quot;</td>
</tr>
<tr>
<td>1 + 2 + &quot;abc&quot;</td>
<td>&quot;3abc&quot;</td>
</tr>
<tr>
<td>&quot;abc&quot; + 9 * 3</td>
<td>&quot;abc27&quot;</td>
</tr>
<tr>
<td>4 - 1 + &quot;abc&quot;</td>
<td>&quot;3abc&quot;</td>
</tr>
</tbody>
</table>
Examples

- See IntOps.java
- Fix the GPA.java program

An Alternative: Type Casting

- **type cast**: An explicit, **FORCED** conversion from one type to another.
- Syntax:
  
  (type) expression

- Type casting has **high precedence** and casts only the item immediately next to it.

- You can cast either up (promotion) or down (truncate)
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

Outline

- Admin and recap
- Variables: more details
Variable Details

- A variable can only store a value of its own type.
- Mixed type allowed only if allowed by automatic numeric promotion

```java
int x;
x = 2.5;  // ERROR: incompatible types
```

```java
double myGPA = 4;
myGPA | 4.0

double avg = 11 / 2; avg | 5.0
```

Variable Details

- A variable can’t be used until it is assigned a value.

```java
int x;
System.out.println(x);  // ERROR: x has no value
```

- You may not declare the same variable twice.

```java
int x;
int x;  // ERROR: x already exists
```

```java
int x = 3;  // OK: declare and initialize
int x = 5;  // ERROR: x already exists
x = 5;     // this is OK
```
Update vs. Algebra

What happens here?

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

Example: Ruler

```java
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
```

```
<p>| | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 1 | 2 | 1 | 3 | 1 | 2 | 1 | 4 | 1 | 2 | 1 | 3 | 1 | 2 | 1
```
Update Shorthand

Since increment updates are common, Java introduces shorthand:

\[
\text{count} = \text{count} + \text{increment};
\]

\[
\text{count} += \text{increment};
\]

when increment is 1

\[
\text{count} ++;
\]

These expressions have the same effect

\[
\text{count} = \text{count} + 1;
\]

\[
\text{count} += 1;
\]

\[
\text{count} ++;
\]

Modify-and-assign

shortcuts to modify a variable's value

<table>
<thead>
<tr>
<th>Shorthand</th>
<th>Equivalent longer version</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable ++;</td>
<td>variable = variable + 1;</td>
</tr>
<tr>
<td>variable --;</td>
<td>variable = variable - 1;</td>
</tr>
<tr>
<td>variable += value;</td>
<td>variable = variable + (value);</td>
</tr>
<tr>
<td>variable -= value;</td>
<td>variable = variable - (value);</td>
</tr>
<tr>
<td>variable *= value;</td>
<td>variable = variable * (value);</td>
</tr>
<tr>
<td>variable /= value;</td>
<td>variable = variable / (value);</td>
</tr>
<tr>
<td>variable %= value;</td>
<td>variable = variable % (value);</td>
</tr>
</tbody>
</table>

int x = 2;
double gpa = 3.8;

x += 3; // x = x + (3) -> 5;
gpa --; // gpa = gpa - 1.0 -> 2.8;
x *= 2; // x = x * 2 -> 10;
x *= 2 + 1; // x = x * (2+1) -> 30;
General: Assignment/Modify-and-Assign as Operators

- You can consider assignment/modify-and-assign as operators, with a lower precedence than the arithmetic operators.

  First the expression on the right hand side of the += operator is evaluated.

  \[
  \text{answer} \quad += \quad \text{sum} / 4 + \text{MAX} * \text{lowest};
  \]

  Then the result is used to calculate in the variable on the left hand side.

Practice

- Compile the list of operators that we covered and their precedence levels.
Example: StockSuccess

- What is the result of adding $1000 on Jan. 1 of each year to a stock account fully invested in S&P500 index ETF?
  - 2008: -38.5%
  - 2009: 23.45%
  - 2010: 12.78%
  - 2011: 0.00%
  - 2012: 13.4%
  - 2013: 29.60%

Outline

- Admin
- Variables
- for Loops
The `for` Statement: Syntax

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

- **Reserved word**
  - `for`

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

- **The statement**
  - is executed until the `condition` becomes false

- **The increment portion**
  - is executed at the end of each iteration

Flowchart of a `for` loop

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

1. **Initialization**
2. **Condition evaluated**
   - If `true`
     - **Statement**
   - If `false`
     - **Increment**

Both semi-colons are always required.
The **for** Statement: Example

```java
for (int counter = 1; counter <= 3; counter++)
{
    System.out.println(counter);
}
```

// beginning of the next statement

**Establish initial value of control variable.**

**Determine if final value of control variable has been reached.**

**Body of loop (this may be multiple statements)**

**Increment the control variable.**

---

**Flexibility of for Loop with Counter**

- **Loop counter:**
  - can use any name, not just `i`
  - can start at any value, not just 1
  - only valid in the loop

- **Compare loop counter with target:**
  - `<` less than
  - `<=` less than or equal to
  - `>` greater than
  - `>=` greater than or equal to

- **Can increment, decrement, times, ...**

```java
for (int i = 1; i <= 6; i++)
{
    System.out.println("I am so smart");
}
```
Using for Loops

- Java's for loop statement performs a task many times.

```java
#include <stdio.h>
int main(void)
{
    int count;
    for (count = 1; count < 500; count++)
        printf("I will not throw paper airplanes in class.");
    return 0;
}
```

Using for Loops?

I NEED TO USE THE COMPUTER

HOLD ON. LET ME SEE IF ANYONE WROTE ON MY FACEBOOK WALL... NOPE.

NOW LET ME SEE IF ANYONE REPLIED TO ME ON TWITTER... NOPE.

NOW LET ME SEE IF ANYONE STARTED FOLLOWING MY TUMBLR... NOPE.

DONE?

LET ME CHECK FACEBOOK AGAIN. SOMEONE COULD HAVE WRITTEN SOMETHING IN THE LAST 30 SECONDS... NOPE.

NOW LET ME CHECK TWITTER AGAIN... NOW TUMLR...

NOW LET ME SEE IF ANYONE REPLIED TO ME ON TWITTER... NOW TUMBLR...

OUR SISTER HAS GONE INFINITY LOOPY.

NOW FACEBOOK. NOW TWITTER. NOW TUMBLR... NOW FACEBOOK. NOW TWITTER. NOW TUMBLR... NOW FACEBOOK. NOW TWITTER... NOW TUMBLR...
Using for Loops

for (int i = 1; i <= 3; i++)
{
    System.out.println(“Now Facebook”);
    System.out.println(“Now Twitter”);
    System.out.println(“Now Tumblr”);
}

Counting Down

- Write a program generating output
  T-minus 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, blastoff!

- Requirement: loop counter starts with 10 and counts down
Counting Down v1

- The update uses -- to count down.

```java
System.out.print("T-minus ");
for (int i = 10; i >= 1; i--) {
    System.out.print(i + ", ");
}
System.out.println("blastoff!");
```

Counting Down v2

- Requirement: loop counter starts with 1 and counts up:

```
T-minus 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, blastoff!
System.out.print("T-minus ");
for (int i = 1; i <= 10; i++) {
    // ???
}
System.out.println("blastoff!");
```
Mapping Loop# to Target Pattern

<table>
<thead>
<tr>
<th>i</th>
<th>number to print</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

\[ y = 11 - x \]

Counting Down

```java
System.out.print("T-minus ");
for (int i = 1; i <= 10; i++) {
    System.out.println(11 - i + "", ");
}
System.out.println("blastoff!");
```

\[ y = 11 - x \]
An “IQ Test” Format

? 10 9 8 7 6 5 4 3 2 1
-1 -1 -1 -1 -1 -1 -1 -1 -1

An “IQ Test” Format

? 10 9 8 7 6 5 4 3 2 1
-1 -1 -1 -1 -1 -1 -1 -1 -1
An “IQ Test” Format

\[
y = 11 - x
\]
Practice: Mapping loop# to numbers

```java
for (int count = 1; count <= 5; count++) {
    System.out.print(...);
}
```

- What statement in the body would cause the loop to print:

```
17 13 9 5 1
```

Mapping loop# to numbers

<table>
<thead>
<tr>
<th>Loop# i:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target:</td>
<td>21</td>
<td>17</td>
<td>13</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>21-4*i</td>
<td>-4</td>
<td>-4</td>
<td>-4</td>
<td>-4</td>
</tr>
</tbody>
</table>

```java
for (int count = 1; count <= 5; count++) {
    System.out.print(-4 * count + 21 + " ");
}
```
Practice: Mapping loop# to numbers

```
for (int count = 1; count <= 5; count++) {
    System.out.print( ... );
}
```

- What statement in the body would cause the loop to print:
  4 7 10 13 16

```
for (int count = 1; count <= 5; count++) {
    System.out.print(3 * count + 1 + " ");
}
```

Practice: Mapping loop# to numbers

```
for (int count = 1; count <= 5; count++) {
    System.out.print( ... );
}
```

- What statement in the body would cause the loop to print:
  2 7 12 17 22

```
for (int count = 1; count <= 5; count++) {
    System.out.print(5 * count - 3 + " ");
}
```
If I want to count down from 12, what should we change?

T-minus 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, blastoff!

System.out.print("T-minus ");
for (int i = 1; i <= 10; i++) {
    System.out.print(11-i + ", ");
}
System.out.println("blastoff!");

Problem: The code has two "magic" numbers 11 and 10, but they are not independent.

System.out.print("T-minus ");
for (int i = 1; i <= 10; i++) {
    System.out.print(11-i + ", ");
}
System.out.println("blastoff!");
**Counting Down: Revision**

```java
int N = 10;
System.out.print("T-minus ");
for (int i = 1; i <= N; i++) {
    System.out.print(N+1-i + "", "");
}
System.out.println("blastoff!");
```

**Code Summary**

```java
int N = 10;
System.out.print("T-minus ");
for (int i = N; i >= 1; i--) {
    System.out.print(i + "", "");
}
System.out.println("blastoff!");
```

```java
int N = 10;
System.out.print("T-minus ");
for (int i = 1; i <= N; i++) {
    System.out.print(N+1-i + "", "");
}
System.out.println("blastoff!");
```
Does the following program give the correct countdown?:

```java
int N = 10;
System.out.print("T-minus ");
for (int i = 1; i <= N; N++) {
    System.out.print(N+1-i + ", ");
}
System.out.println("blastoff!");
```

Answer: No. There is a typo (N for i)

Q: can the computer help me to find it (read my mind?)
Constant

- Use keywords to tell computer your intention
- If there is a `final` before a variable declaration, it is your promise to the computer that you will not modify it after declaration
- If you break your promise, the compiler will catch you

```java
final int N = 10;
System.out.print("T-minus ");
for (int i = 1; i <= N; i++) {
    System.out.print(N+1-i + ", ");
}
System.out.println("blastoff!");
```
Backup Slides

Type Conversions in Java

- **Identity conversion (i.e., no conversion)**
- **Conversions related to primitive data types**
  - ✓ widening primitive conversions
  - ✓ narrowing primitive conversions
- **Conversions related to classes**
  - ✓ widening reference conversions
  - ✓ narrowing reference conversions
  - ✓ we will cover these two cases later in the course; they are powerful tools to allow polymorphism
- **Conversions related to Strings**
  - ✓ string conversions: i.e., convert a numerical data to a string, e.g., the number 17 to the string “17”
Widening Primitive Conversions

- Widening primitive conversions are those that do not lose information about the overall magnitude of a numeric value.
- Java defines 19 primitive conversions as widening primitive conversions:
  - byte → short, int, long, float, double
  - short → int, long, float, double
  - char → int, long, float, double
  - int → long, float, double
  - long → float, double
  - float → double

- They are generally safe because they tend to go from a small data type to a larger one (such as a short to an int).
  - Can potential problems happen in some of the cases?

Narrowing Primitive Conversions

- Java defines 23 primitive conversions as narrowing primitive conversions:
  - byte → char
  - short → byte, char
  - char → byte, short
  - int → byte, short, char
  - long → byte, short, char, int
  - float → byte, short, char, int, long
  - double → byte, short, char, int, long, float

- Narrowing primitive conversions may lose either overall magnitude of a numeric value and/or precision.
Assignment during Declaration

- You can assign a value to a variable when declaring it.
  - This is called initialization

- Syntax:
  \[ <type> <name> = <expression>; \]

  - int x = (11 \% 3) + 12;  
    - x 14
  - double myGPA = 3.95;
    - myGPA 3.95

Example: Receipt

- Once given a value, a variable can be used in expressions:

```
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed
        // assuming 6% tax / 15% tip
        int subtotal = 38 + 40 + 30;
        double tax = subtotal * .06;
        double tip = subtotal * .15;
        double total = subtotal + tax + tip;

        System.out.println("Subtotal: " + subtotal);
        System.out.println("Tax: " + tax);
        System.out.println("Tip: " + tip);
        System.out.println("Total: " + total);
    }
}
```
Update Variables

- You can update variable values:

  ```java
  int x;
  x = 3;
  System.out.println(x + " here");  
  // 3 here
  
  x = 4 + 7;
  System.out.println("now x is " + x);  // now x is 11
  ```

Example: StockSuccessSimple

- Stock market average return 8.5%. What is the total asset after saving $1000 every year for 50 years?