Introduction to Computer Programming

Lecture 3 – Primitive Data Types and Simple Counting Loops

Data Types

- **Type** - A category or set of data values.
  - Constrains the operations that can be performed on data
  - Many languages ask the programmer to specify types
  - Examples: integer, real number, string
Data Representation

- Internally, computers store everything as 1s and 0s
  
  104 → 01101000
  "hi" → 01101000110101

Java's primitive types

- **Primitive types** - 8 simple types for numbers, text, etc.
- Java also has **object types**, which we'll talk about later.
Java's primitive types

<table>
<thead>
<tr>
<th>Name</th>
<th>Descriptions</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>Integers (up to $2^{31}-1$)</td>
<td>42, -3, 0, 926394</td>
</tr>
<tr>
<td>double</td>
<td>Double-precision real number (up to $10^{308}$)</td>
<td>3.1, -0.25, 9.4e3</td>
</tr>
</tbody>
</table>
| char   | Single characters                 | 'a', 'X', '?', '
' |
| boolean| Logical values                    | true, false       |

• Why does Java distinguish integers vs. real numbers?

Expressions

• **Expression**:- A value or operation that computes a value.
  • Examples:  
    
    $$1 + 4 * 5$$
    $$\frac{(7 + 2) * 6}{3}$$
    $$42$$

  – The simplest expression is a **literal value**.
  – A complex expression can use operators and parentheses.
Arithmetic Operators

- **Operator** - Combines multiple values or expressions.
  + addition
  - subtraction (or negation)
  * multiplication
  / division
  % modulus (a.k.a. remainder)

Evaluating Expressions

- As a program runs, its expressions are **evaluated**.
  1 + 1 evaluates to 2
  System.out.println(3 * 4); prints 12
- How would we print the text 3 * 4?
Integer division with /

- When we divide integers, the quotient is also an integer.

  14 / 4 is 3 R 2, not 3.5 (Result Is 4)
  45 / 10 is 4 R 5 (Result Is 4)
  1425 / 27 is 52 R 21 (Result Is 4)
  32 / 5 is 6 R 2 (Result Is 4)
  84 / 10 is 8 R 4 (Result Is 4)
  156 / 100 is 1 R 56 (Result Is 4)

- Dividing by 0 causes an error when your program runs.

Integer division with %

- The % operator computes the remainder from integer division..

  14 / 4 is 3 R 2 (Remainder is 2)
  45 / 10 is 4 R 5 (Remainder is 5)
  1425 / 27 is 52 R 21 (Remainder is 21)
  32 / 5 is 6 R 2 (Remainder is 2)
  84 / 10 is 8 R 4 (Remainder is 4)
  156 / 100 is 1 R 56 (Remainder is 56)
Integer division with %

• Applications of % operator:
  – Obtain last digit of a number:
    \[ 230857 \% 10 \text{ is } 7 \]
  – Obtain last 4 digits:
    \[ 658236489 \% 10000 \text{ is } 6489 \]
  – See whether a number is odd:
    \[ 7 \% 2 \text{ is } 1, \quad 42 \% 2 \text{ is } 0 \]

Precedence

• **Precedence** - Order in which operators are evaluated.
  – Generally operators evaluate left-to-right.
    \[ 1 - 2 - 3 \text{ is } (1 - 2) - 3 \text{ which is } -4 \]
  – But \* / % have a higher level of precedence than + –
    \[ 1 + 3 \* 4 = 13 \]
    \[ 6 + 8 / 2 \* 3 = 6 + 4 \* 3 \]
    \[ = 6 + 12 \]
    \[ = 18 \]
Precedence

• Parentheses can force a certain order of evaluation:
  \[(1 + 3) \times 4\] is 16

• Spacing does not affect order of evaluation
  \[1 + 3 \times 4 - 2\] is 11

Precedence Example

\[
1 \times 2 + 3 \times 5 \mod 4 \\
= 2 + 3 \times 5 \mod 4 \\
= 2 + 15 \mod 4 \\
= 2 + 3 \\
= 5
\]
Precedence Example

\[
1 + 8 \% 3 * 2 - 9 \\
= 1 + 2 * 2 - 9 \\
= 1 + 4 - 9 \\
= 5 - 9 \\
= -4
\]

Precedence Questions

• What values result from the following expressions?
  9 / 5
  695 % 20
  7 + 6 * 5
  7 * 6 + 5
  248 % 100 / 5
  6 * 3 - 9 / 4
  (5 - 7) * 4
  6 + (18 % (17 - 12))
Real numbers (type `double`)

- Examples:
  - `6.022`, `-42.0`, `2.143e17`
  - Placing `.0` or `.` after an integer makes it a `double`.
- The operators `+-*/%()` all still work with `double`.
  - `/` produces an exact answer: `15.0 / 2.0` is `7.5`
  - Precedence is the same: `()` before `*/%` before `+-`

Real number example

```
2.0 * 2.4 + 2.25 * 4.0 / 2.0
= 4.8 + 2.25 * 4.0 / 2.0
= 4.8 + 9.0 / 2.0
= 4.8 + 4.5
= 9.3
```
Mixing Types

• When int and double are mixed, the result is a double.
  \[ 4.2 \times 3 \text{ is } 12.6 \]

• The conversion is per-operator, affecting only its operands.
  \[
  \begin{align*}
    7 / 3 \times 1.2 + 3 / 2 \\
    = 2 \times 1.2 + 3 / 2 \\
    = 2.4 + 3 / 2 \text{ (equals 1 not 1.5)} \\
    = 2.4 + 1
  \end{align*}
  \]

Mixing types

\[
2.0 + 10 / 3 \times 2.5 - 6 / 4 \\
= 2.0 + 3 \times 2.5 - 6 / 4 \\
= 2.0 + 7.5 - 6 / 4 \\
= 2.0 + 7.5 - 1 \\
= 9.5 - 1 \\
= 8.5
\]
String Concatenation

- **String Concatenation** - Using + between a string and another value to make a longer string.
  
  "hello" + 42 is "hello42"
  
  1 + "abc" + 2 is "1abc2"
  
  "abc" + 1 + 2 is "abc12"
  
  1 + 2 + "abc" is "3abc"
  
  "abc" + 9 * 3 is "abc27"
  
  "1" + 1 is "11"
  
  4 - 1 + "abc" is "3abc"

String Concatenation

- Use + to print a string and an expression's value together.
  
  System.out.println("Grade: " + (95.1 + 71.9) / 2);

- Output: Grade: 83.5
Receipt Example

What's bad about the following code?

```java
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed, 
        // assuming 8% tax / 15% tip 
        System.out.println("Subtotal:");
        System.out.println(38 + 40 + 30);
        System.out.println("Tax:");
        System.out.println((38 + 40 + 30) * .08);
        System.out.println("Tip:");
        System.out.println((38 + 40 + 30) * .15);
        System.out.println("Total:");
        System.out.println(38 + 40 + 30 +
                            (38 + 40 + 30) * .08 +
                            (38 + 40 + 30) * .15)
    }
}
```

Declaration

- **Variable Declaration** - Sets aside memory for storing a value.
  - Variables must be declared before they can be used.

- **Syntax:**
  ```java
  DataType Name;  
  ```

- The name is an *identifier*.
  ```java
  int x;  
  double myGPA;  
  ```
Assignment

• **Assignment** - Stores a value into a variable.
  – The value can be an expression; the variable stores its result.

• Syntax:

  \[
  \text{Name} = \text{Expression};
  \]

  \[
  \begin{align*}
  \text{int} & \ x; \\
  \text{x} & \ = \ 3; \\
  \text{double} & \ \text{myGPA}; \\
  \text{myGPA} & \ = \ 1.0 + 2.25;
  \end{align*}
  \]

Declaration/Initialization

• A variable can be declared/initialized in one statement.

• Syntax:

  \[
  \text{type Name} = \text{Expression};
  \]

  \[
  \begin{align*}
  \text{double} & \ \text{myGPA} = 3.95; \\
  \text{int} & \ x = (11 \ % \ 3) + 12;
  \end{align*}
  \]
Assignment and Algebra

- Assignment uses \( = \), but it is not an \emph{algebraic} equation.
  
  \( = \) means, \emph{"store the value at right in variable at left"}
  
The right side expression is evaluated first, and then its result is stored in the variable at left.

- What happens here?
  
  \begin{verbatim}
  int x = 3;
  x = x + 2;    // ???
  \end{verbatim}

Assignment and types

- A variable can only store a value of its own type.

  \begin{verbatim}
  int x = 2.5;
  // ERROR: incompatible types
  \end{verbatim}
Assignment and types

• An int value can be stored in a double variable.
• The value is converted into the equivalent real number.

  double myGPA = 4;  
  double avg = 11 / 2;

  myGPA 4.0
  avg 5.0

• Why does avg store 5.0 and not 5.5?

Compiler Errors

• A variable can't be used until it is assigned a value.

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

• You may not declare the same variable twice.

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

  int x = 3;
  int x = 5;           // ERROR: x already exists

• How can this code be fixed?
Printing a variable's value

• Use + to print a string and a variable's value on one line.

```java
double grade = (95.1 + 71.9 + 82.6) / 3.0;
System.out.println("Your grade was "+ grade);
int students = 11 + 17 + 4 + 19 + 14;
System.out.println("There are " + students + " students in the course.");
```

• Output:
  
  Your grade was 83.2
  There are 65 students in the course.

Receipt answer

```java
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed, assuming 8% tax / 15% tip
        int subtotal = 38 + 40 + 30;
        double tax = subtotal * .08;
        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);
    }
}
```
Why loops?

- Computers offer several advantages over calculators.
- If it is necessary, they can perform the same steps over and over again, simply by rerunning the program.
- But is this the only way to get a computer to perform the same action repeatedly? And is this the only reason for getting a computer to repeat itself?

Example: Average of three numbers

- Let’s take another look at our program that finds the average of three numbers:
  ```java
  import java.util.Scanner;
  public class AverageThree {
      public static void main(String[] args) {
          Scanner keyb = new Scanner(System.in);
          int value1, value2, value3;
          int sum, average;
          System.out.println("What is the first value?");
          value1 = keyb.nextInt();
          System.out.println("What is the second value?");
          value2 = keyb.nextInt();
          System.out.println("What is the third value?");
          value3 = keyb.nextInt();
          sum = value1 + value2 + value3;
          average = sum / 3;
          System.out.println("The average is "+ average);
      }
  }
  ```
Example : Average of three numbers (continued)

```java
text
System.out.println("What is the third value?");
value3 = keyb.nextInt();
sum = value1 + value2 + value3;
average = sum / 3;
System.out.println("Average = "+average);
```

- What would we do if we wanted the program to average 5 values instead of 3? or 10? or 100?
- This is clearly not the best way to write this!

---

**Loops**

- We need the ability to perform the same set of instructions repeatedly so we don’t have to write them over and over again.
- This is why C++ includes several ways of using repetition in a program.
- Each case where we repeat a set of statement is called a *loop*. 
Counting Loops

- The first type of loop is a **counting loop**.
- Counting loops are repeated a specific number of times.
- If you read the loop, you can easily figure out how many times its statements will be performed.

Example: Hello Again

- Example - Write a program that greets the user with "Hi there!" five times.
- We could write the program like this:

```java
import java.util.Scanner;

public class HelloAgain {
    // Hello again - this program writes "Hello, again" five times
    public static void main(String[] args) {
        System.out.println("Hello, again");
        System.out.println("Hello, again");
        System.out.println("Hello, again");
        System.out.println("Hello, again");
        System.out.println("Hello, again");
    }
}
```
## Increment and Decrement

*Shortcuts to increase or decrease a variable's value by 1*

<table>
<thead>
<tr>
<th>Shorthand</th>
<th>Equivalent longer form</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable++;</td>
<td>variable = variable + 1;</td>
</tr>
<tr>
<td>variable--;</td>
<td>variable = variable - 1;</td>
</tr>
</tbody>
</table>

```c
int x = 2;
x++; // x = x + 1;  // x now stores 3

double gpa = 2.5;
gpa--; // gpa = gpa - 1; // gpa now stores 1.5
```

## Modify-And-Assing

*Shortcuts to modify a variable's value*

<table>
<thead>
<tr>
<th>Shorthand</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<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>

```c
x += 3;  // x = x + 3;
gpa -= 0.5; // gpa = gpa - 0.5;
number *= 2;  // number = number * 2;
```
Repetition over a range

System.out.println("1 squared = " + 1 * 1);
System.out.println("2 squared = " + 2 * 2);
System.out.println("3 squared = " + 3 * 3);
System.out.println("4 squared = " + 4 * 4);
System.out.println("5 squared = " + 5 * 5);
System.out.println("6 squared = " + 6 * 6);

• Intuition - "I want to print a line for each number from 1 to 6"

Repetition over a range

• Intuition: "I want to print a line for each number from 1 to 6"
• The for loop does exactly that!

    for (int i = 1; i <= 6; i++) {
        System.out.println
          (i + " squared = " + (i * i));
    }

    – "For each integer i from 1 through 6, print ..."
Counting Loops

• We use a for loop to write counting loops
• In Java, it looks like this:
  ```java
  for (count = start; count <= finish; count++)
      statement
  ```
• or
  ```java
  for (count = start; count <= finish; count++)
      { statements
      }
  ```

Counting Loops (continued)

```java
for (count = start; count <= finish; count++)
    statement
variable used to count times through the loop
initial value of the counter
final value of the counter
```
Example: Rewriting *HelloAgain*

• Let's write the steps that our new version of that program must perform:

1. Write "Hi, there!" on the screen 5 times.

```
1. FOR i goes from 1 TO 5
1.1 Write “Hi, there!”
```

Refining *HelloAgain*

```
1. FOR i goes from 1 TO 5
1.1 Write “Hi, there!”
```

```
for (i = 1; i <= 5; i++)
```
Refining *HelloAgain*

for (i = 1; i <= 5; i++)

```
1.1 Write “Hi, there!”
```

```
/ *
System.out.println("Hi, there! ");
```

---

The New *HelloAgain*

```java
class HelloAgain2  {
    // HelloAgain2 - this is a better way to write
    // "Hello, again" five times
    public static void main(String[] args) {
        int i;
        for (i = 1; i <= 5; i++)
            System.out.println("Hello, again");
    }
}
```
Generalizing *HelloAgain*

- This program is also flawed; it gives us no choices as to how many times we can print “Hi, there!”
- We can let the user select how many times to print the message and making this version of the program more general is fairly easy:
- Our algorithm will start as:
  1. Find out how many times to print the message.
  2. Print "Hi, there!" that many times.

---

Generalizing *HelloAgain* (continued)

1. Find out how many times to print the message.
2. Print "Hi, there!" that many times.

```java
System.out.println("How many times do you want to say \"hello\"?");
totalTimes = keyb.nextInt();
```
Generalizing *HelloAgain* (continued)

```java
System.out.println("How many times do you want to " + "say \"hello\"?");
totalTimes = keyb.nextInt();
```

2. Print "Hi, there!" that many times.

```
2. FOR Count goes from 1 TO TotalTimes
2.1 Write “Hi, there!”
```

```java
for (count = 0; count < totalTimes; count++)
    System.out.println("Hello, again");
```
The Revised *HelloAgain*

```java
import java.util.Scanner;

public class HelloAgain3 {
    // HelloAgain3 - Write "Hello, again" as many times
    //               as the user wants
    public static void main(String[] args) {
        Scanner keyb = new Scanner(System.in);
        int i, count, totalTimes;

        System.out.println("How many times do you want to " + "say \"hello\"?");
        totalTimes = keyb.nextInt();
        for (count = 0; count < totalTimes; count++)
            System.out.println("Hello, again");
    }
}
```

**Loop walkthrough**

```java
for (int i = 1; i <= 4; i++) {
    System.out.println(i + " squared = " + (i * i));
}
```

Output:

1 squared = 1
2 squared = 4
3 squared = 9
4 squared = 16
Whoo!

![Diagram of loop walkthrough](image)
Multi-line loop body

```java
System.out.println("+----+");
for (int i = 1; i <= 3; i++) {
    System.out.println("\ / ");
    System.out.println("/    ");
}
System.out.println("+----+");
```

Output:
```
+----+
 \ / 
/    
/     
/      
/       
+----+
```

Expressions for counter

```java
int highTemp = 5;
for (int i = -3; i <= highTemp / 2; i++) {
    System.out.println(i * 1.8 + 32);
}
```

Output:
```
 26.6
 28.4
 30.2
 32.0
 33.8
 35.6
```
System.out.print

- Prints without moving to a new line
  - allows you to print partial messages on the same line
    ```java
    int highestTemp = 5;
    for (int i = -3; i <= highestTemp / 2; i++) {
        System.out.print((i * 1.8 + 32) + "  ");
    }
    ```
- Output:
  26.6  28.4  30.2  32.0  33.8  35.6
- Concatenate "  " to separate the numbers

Counting down

- The update can use -- to make the loop count down.
- The test must say > instead of <
  ```java
  System.out.print("T-minus ");
  for (int i = 10; i >= 1; i--) {
      System.out.print(i + ", ");
  }
  System.out.println("blastoff!");
  System.out.println("The end.");
  ```
- Output:
  
  T-minus 10, 9, 8, 7, 6, 5, 4, 3, 2, 1,
  blastoff!
  The end.
Example: Averaging \( n \) Numbers

- Let's get back to our original problem. We want to able to average any number of values.
- Let's start by outlining our algorithm:
  1. Find out how many values there are.
  2. Add up all the values.
  3. Divide by the number of values
  4. Print the result

Refining Avg\( n \)

1. Find out how many values there are.
2. Add up all the values.
3. Divide by the number of values
4. Print the result

```java
System.out.println("How many values are you going to enter?");
numValues = keyb.nextInt();
```
Refining \texttt{Avgn}

\begin{verbatim}
System.out.println
    ("How many values are you going to enter?");
numValues = keyb.nextInt();
2. Add up all the values.
3. Divide by the number of values
4. Print the result

2.1 For CurrentValue goes from 1 to NumValues:
2.1.1 Get the next value
2.1.2 Add it to the total
\end{verbatim}

Refining \texttt{Avgn}

\begin{verbatim}
System.out.println
    ("How many values are you going to enter?");
numValues = keyb.nextInt();
2.1 For CurrentValue goes from 1 to NumValues:
2.1.1 Get the next value
2.1.2 Add it to the total
3. Divide by the number of values
4. Print the result

2.0 Set the total to zero (initially there \textbf{are no} values)
\end{verbatim}
Refining Avgn

System.out.println
    ("How many values are you going to enter?");
numValues = keyb.nextInt();
2.0 Set the total to zero (initially there are no values)
2.1 For CurrentValue goes from 1 to NumValues :
2.1.1 Get the next value
2.1.2 Add it to the total
3. Divide by the number of values
4. Print the result

sum = 0.0;
for (currentValue = 1; currentValue <= numValues; 
currentValue++)
    {
    System.out.println("What is the next value?");
    value = keyb.nextInt();
    sum = sum + value;
    }
Refining Avg

```java
System.out.println
    ("How many values are you going to enter?");
numValues = keyb.nextInt();
sum = 0.0;
for (currentValue = 1; currentValue <= numValues;
    currentValue++)
    {  
    System.out.println("What is the next value?");
    value = keyb.nextInt();
    sum = sum + value;
}
3. Divide by the number of values
4. Print the result
average = sum / numValues;
System.out.println("The average is "+ average);
```

The AverageN Program

```java
import java.util.Scanner;

class AverageN {

    // AverageN - Find the average of N values
    public static void main(String[] args) {
        Scanner keyb = new Scanner(System.in);
        double sum, average, value;
        int    numValues, currentValue;

        // Find out how many values there are
        System.out.println
            ("How many values are you going to enter?");
        numValues = keyb.nextInt();
```
// Read in each value and add it to the sum
sum = 0.0;
for (currentValue = 1;  
currentValue <= numValues;  
currentValue++)  
{  
   System.out.println("What is the next value?");  
   value = keyb.nextInt();  
   sum = sum + value;
}

// Calculate and print out the average
average = sum / numValues;
System.out.println("The average is "+ average);