CSE 1223: Introduction to Computer Programming in Java
Chapter 2 – Java Fundamentals
Recall From Last Time: Java Program

```java
import java.util.Scanner;

public class EggBasketEnhanced {
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        System.out.print("Enter the number of eggs in each basket: ");
        String eggsStr = keyboard.nextLine();
        int eggsPerBasket = Integer.parseInt(eggsStr);
        System.out.print("Enter the number of baskets: ");
        String basketStr = keyboard.nextLine();
        int numberOfBaskets = Integer.parseInt(basketStr);

        int totalEggs = numberOfBaskets * eggsPerBasket;

        System.out.println(eggsPerBasket + " eggs per basket.");
        System.out.println(numberOfBaskets + " baskets.");
        System.out.println("Total number of eggs is " + totalEggs);
    }
}
```
What Does EggBasketEnhanced Do?

- Take a look at the program and see if you can figure out what it does.
  - There are some things in here we haven’t talked about yet.
  - Can you guess what they’re doing based on their names and how we’re using them?
What Is a Program Variable?

```c
int numberOfBaskets;
```

- This is a *declaration* of an integer variable
- A **variable** is a named location to store data, i.e., a container for data
What Is a Program Variable?

- This is a *declaration* of an integer variable
- A **variable** is a named location to store data, i.e., a container for data
- Each variable can hold only one type of data; for example only integers, only floating point (real) numbers, or only characters
- All program variables **must** be *declared* with a *type* before being used
What Is a Program Type?

- A variable’s **type** determines the kind of values that a variable can hold and what operations can be applied to it.
  - Java has two different types – *primitive* types and *reference* types
    - We’ll discuss reference types later

- Some Java *primitive* types:
  - `int` (integer, whole values, e.g., 0, 1, -13, 231)
  - `double` (real values, e.g., 0.0, 3.1415, -2.72)
  - `char` (single character values, e.g., ‘a’, ‘3’, ‘$’)
  - `boolean` (only one of two values: `true`, `false`)
How Do We Assign/Change the Value of a Variable?

String eggsStr = keyboard.nextLine();
totalEggs = numberOfBaskets * eggsPerBasket;

- Assignment statement:
  \[
  \text{variable} = \text{expression};
  \]
- Assigns the value of the expression on the right side of = to the variable on the left side
  - It does not mean “equal” like in math!
- These expressions look different, but they’re both doing the same thing!
What Is an Expression?

- Program expressions are very much like arithmetic expressions you are familiar with (usual operators, parenthesis, precedence rules, etc.)
- Expressions can be evaluated to produce a value and they have a type (the type of the value of the expression)

```
numberOfBaskets * eggsPerBasket
```
What Is an Expression?

Expressions can also be the result of evaluating something known as a *function*

- The above are examples of *functions*
- Program functions are very similar to mathematical functions
  - Take inputs, produce outputs from those inputs
  - Such as \( y = f(x) \) – \( f(x) \) is a function where \( x \) is the input, \( f(x) \) is the output
- More on this later!
Numeric Operators

- Some common integer operators:
  - + (obvious)
  - - (obvious)
  - * (obvious)
  - / (integer division, e.g., 6/2=3, 5/2=2, 19/5=?)
  - % (mod operator, i.e., remainder of integer division, e.g., 6%2=0, 5%2=1, 19%5=?)

- Some common real operators:
  - +, -, *, / (real division)
    - Note that real division is *approximate*
    - Sometimes the results you get will be surprising!
Integer arithmetic

For integers, it is important to understand the / and % operators

- / is integer division. It is not exactly the same as division for real numbers
  - 4 / 2 = 2
  - But 5 / 2 = 2 also
  - Rule is “divide, then round closer to zero”
    - Not “round down” – “round toward zero”
    - So -5/2 = -2
Integer arithmetic

- For integers, it is important to understand the `/` and `%` operators
  - `%` is *integer remainder*
    - Works much like you might remember long division working when you were in grade school
    - 5 % 2 = 1
      - Because 5/2=2 with 1 remaining
    - 4 % 2 = 0
      - Because 4/2=2 with 0 remaining
    - 19 % 5 = 4
      - Because 19/5 = 3 with 4 remaining
    - etc.
Integer arithmetic

- For integers, it is important to understand the / and % operators
  - / and % work together
    - \((x / y) * y + (x \% y) = x\)
      - 19 / 5 = 3
      - 19 % 5 = 4
      - 5 * 3 = 15
      - 15 + 4 = 19
  - Use % to check whether a number is even or not
    - 10 % 2 = 0
    - 11 % 2 = 1
Some Expressions: 
What Are Their Values?

```plaintext
int i = 12, j = 5, k = -3;
double x = 2.1, y = -1.5, z = 3.0;
```

- \((i + j + k) / 3\)
- \((i / j) * j + (i \% j)\)
- \(x * x + y * y\)
- \((x + y + z) / (x - y - z)\)
- \(2.0 * z - (x + y)\)
Output Statements

System.out.println(output1 + output2 + ... + outputN);
System.out.print(output1 + output2 + ... + outputN);

- Concatenates the various outputs (quoted strings, variables, constants/numbers) and prints them to the screen (println adds a newline).
- What do the following statements output?

```java
int day = 19;
System.out.print("January " + day + ", " + 2004 + 
    " is Martin Luther King’s Day! ");
```
Input Statements

Given the import:
```
import java.util.Scanner;
```
and the declaration:
```
Scanner in = new Scanner(System.in);
```
- Declare and input a whole line (a string of characters):
  ```
  String s = in.nextLine();
  ```
- You can convert input to integer or double values with special functions:
  ```
  int val = Integer.parseInt(s);
  double dVal = Double.parseDouble(s);
  ```
To Sum Up...

So far, we have seen

- how to *input* values from the keyboard
- how to *output* messages/values to the screen
- how to create variables to store values
- how to store values in variables and compute new values with expressions
It’s Your Turn!

Now let’s put it all together: Write a Java program called `ComputeArea`, which asks the user for the width and height of a rectangle and computes and prints the area of the rectangle.
The `char` Type

- A variable of the `char` data type stores a single “printable” character
- A `char` constant or literal is a character in single quotes, e.g., ‘y’
- For example:
  ```java
  char answer = 'y';
  System.out.println(answer);
  ```
  prints (displays) the letter y
The String Type

- A string is a sequence of characters
- The String type is used to declare variables that store strings
- The String type is not a *primitive* type: it is known as a *class* or *reference* type
- A String constant is one or more characters in double quotes, e.g., “string constant”
- Examples:
  ```java
  char charVariable = 'a'; // single quotes
  String stringVariable = "a"; // double quotes
  String sentence = "Hello, world";
  ```
String Variables

- Declare a String variable:
  ```java
  String greeting;
  ```
- Assign a value to the variable:
  ```java
  greeting = "Hello!";
  ```
- Use the variable as a String argument in a method call:
  ```java
  System.out.println(greeting);
  ```
  causes the string Hello! to be displayed on the screen
Indexing Characters Within a String

- The index of a character within a string is an integer starting at 0 for the first character and gives the position of the character.
- For example:

  ```java
  String str = "This is a string";
  ```

<table>
<thead>
<tr>
<th>T</th>
<th>h</th>
<th>i</th>
<th>s</th>
<th>i</th>
<th>s</th>
<th>a</th>
<th>s</th>
<th>t</th>
<th>r</th>
<th>i</th>
<th>n</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>
Methods

- A *method* is an operation with a name and a list of arguments (possibly empty)
- A method can simply perform an action or it can return a value
  - A call to a method that does not return a value is a *statement*, e.g.,
    ```
    System.out.println("Hi there!");
    ```
  - A call to a method that returns a value is called a *function* and can be used as an *expression*, e.g.,
    ```
    int i = keyboard.nextInt();
    ```
Some String Methods

- The String type has many methods that allow us to manipulate String values/variables.
  - For now, this is the main distinction between *primitive* types and *reference* types.
    - Variables of reference types have methods that can be invoked with the syntax below.
    - Variables of primitive types do not!

- String methods are called/invoked with the following syntax:

  ```
  stringVar.methodName(arguments)
  ```
Some String Methods

- `int length()` returns the number of characters in the given string, e.g.,
  ```java
  String str = "This is a string";
  System.out.println(str.length());
  ```
  What’s the output?
Some String Methods

- **int length()**
  returns the number of characters in the given string, e.g.,
  ```java
  String str = "This is a string";
  System.out.println(str.length());
  ```
  What’s the output?
  16
String Methods cont.

- **charAt** *(int pos)*
  returns the character at position *pos* in the given string, e.g.,
  ```java
  String str = "This is a string";
  System.out.println(str.charAt(0));
  System.out.println(str.charAt(1));
  System.out.println(str.charAt(15));
  ```
  What’s the output?
String Methods cont.

- `char charAt(int pos)` returns the character at position `pos` in the given string, e.g.,
  
  ```java
  String str = "This is a string";
  System.out.println(str.charAt(0));
  System.out.println(str.charAt(1));
  System.out.println(str.charAt(15));
  System.out.println(str.charAt(15));
  
  What's the output?
  T
  h
  g
  ```
Some String Methods cont.

- `String substring(int start, int end)` returns the string starting at position `start` and ending at position `(end-1)` in the given string, e.g.,

```java
String str = "This is a string";
System.out.println(str.substring(0, 4));
System.out.println(str.substring(5, 7));
System.out.println(str.substring(0, 16));
```

What’s the output?
Some String Methods cont.

- String substring(int start, int end) returns the string starting at position start and ending at position (end-1) in the given string, e.g.,
  String str = "This is a string";
  System.out.println(str.substring(0,4));
  System.out.println(str.substring(5,7));
  System.out.println(str.substring(0,16));

  What’s the output?
  This
  is
  This is a string
Some String Methods cont.

- **int indexOf(String aString)** returns the position of the first occurrence of string `aString` in the given string (or -1 if not found), e.g.,

```java
String str = "This is a string";
System.out.println(str.indexOf("This"));
System.out.println(str.indexOf("is"));
System.out.println(str.indexOf("yoh"));
```

What’s the output?
Some String Methods cont.

- **int indexOf(String aString)** returns the position of the first occurrence of string `aString` in the given string (or -1 if not found), e.g.,

```java
String str = "This is a string";
System.out.println(str.indexOf("This"));
System.out.println(str.indexOf("is"));
System.out.println(str.indexOf("yoh"));
```

What's the output?

- 0
- 2
- -1
Some String Methods cont.

- **boolean equals(String aString)**
  returns true if the string *aString* is the same as the given string, false otherwise, e.g.,
  ```java
  String str = "test";
  System.out.println(str.equals("test"));
  System.out.println(str.equals("fish"));
  System.out.println(str.equals("TEST"));
  ```
  What’s the output?
Some String Methods cont.

- `boolean equals(String aString)`
  returns true if the string `aString` is the same as the given string, false otherwise, e.g.,

  ```java
  String str = "test";
  System.out.println(str.equals("test"));
  System.out.println(str.equals("fish"));
  System.out.println(str.equals("TEST"));
  
  What’s the output?
  
  true
  false
  false```
As we have seen before the + operator can be used to concatenate string values, e.g.,

String name = "Cindy";
String greeting = "Hi, " + name + "!";
System.out.println(greeting);

What is the output?
As we have seen before the + operator can be used to concatenate string values, e.g.,

```java
String name = "Cindy";
String greeting = "Hi, " + name + "!";
System.out.println(greeting);
```

What is the output?

Hi, Cindy!
Single Character Input

Given the import:

```java
import java.util.Scanner;
```

and the declaration:

```java
Scanner in = new Scanner(System.in);
```

- Declare and input a single character:
  ```java
  String s = in.nextLine();
  char c = s.charAt(0);
  ```

- Note that input here is actually the whole line, we’re just ignoring the rest of the line.
How do you print the following string?
The word "hard"

Would this do it?

```java
System.out.println("The word "hard"");
```

No, it would give a compiler error - it sees the string
The word between the first set of double quotes and
is confused by what comes after

Use the backslash character, "\\", to escape the special
meaning of the internal double quotes:

```java
System.out.println("The word \"hard\" ");
```
public class StringTest
{
    public static void main(String[] args)
    {
        String greeting = "Hello!";
        int len = greeting.length();
        char ch = greeting.charAt(3);
        String sub = greeting.substring(1, 3);
        int index1 = greeting.indexOf("lo");
        int index2 = greeting.indexOf("low");
        System.out.println("Length is: "+len);
        System.out.println("Char at 3 is: "+ch);
        System.out.println("Substring is: "+sub);
        System.out.println("Index of "lo" is: "+index1);
        System.out.println("Index of "low" is: "+index2);
    }
}
What Is The Output Of StringTest?

- Trace through the statements of StringTest and determine the output produced by the program.
<table>
<thead>
<tr>
<th>Program Line</th>
<th>Program state</th>
</tr>
</thead>
<tbody>
<tr>
<td>String greeting = &quot;Hello&quot;;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>greeting = “Hello”</td>
</tr>
<tr>
<td>int len = greeting.length();</td>
<td></td>
</tr>
<tr>
<td></td>
<td>greeting = “Hello”</td>
</tr>
<tr>
<td></td>
<td>len = 5</td>
</tr>
<tr>
<td>char ch = greeting.charAt(3);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>greeting = “Hello”</td>
</tr>
<tr>
<td></td>
<td>len = 5</td>
</tr>
<tr>
<td></td>
<td>ch = ‘l’</td>
</tr>
<tr>
<td>String sub = greeting.substring(1, 3);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>greeting = “Hello”</td>
</tr>
<tr>
<td></td>
<td>len = 5</td>
</tr>
<tr>
<td></td>
<td>ch = ‘l’</td>
</tr>
<tr>
<td></td>
<td>sub = “el”</td>
</tr>
<tr>
<td>Program Line</td>
<td>Program state</td>
</tr>
<tr>
<td>-------------------------------------------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>greeting = &quot;Hello&quot;</td>
<td>greeting = &quot;Hello&quot;</td>
</tr>
<tr>
<td>len = 5</td>
<td>len = 5</td>
</tr>
<tr>
<td>ch = ‘l’</td>
<td>ch = ‘l’</td>
</tr>
<tr>
<td>sub = “el”</td>
<td>sub = “el”</td>
</tr>
<tr>
<td>int index1 = greeting.indexOf(&quot;lo&quot;);</td>
<td>index1 = 3</td>
</tr>
<tr>
<td></td>
<td>greeting = “Hello”</td>
</tr>
<tr>
<td></td>
<td>len = 5</td>
</tr>
<tr>
<td></td>
<td>ch = ‘l’</td>
</tr>
<tr>
<td></td>
<td>sub = “el”</td>
</tr>
<tr>
<td>int index2 = greeting.indexOf(&quot;low&quot;);</td>
<td>index2 = -1</td>
</tr>
<tr>
<td></td>
<td>greeting = “Hello”</td>
</tr>
<tr>
<td></td>
<td>len = 5</td>
</tr>
<tr>
<td></td>
<td>ch = ‘l’</td>
</tr>
<tr>
<td></td>
<td>sub = “el”</td>
</tr>
<tr>
<td></td>
<td>index1 = 3</td>
</tr>
<tr>
<td></td>
<td>Index2 = -1</td>
</tr>
</tbody>
</table>
Program State

- Note how program variables are represented in the program state
  - Program variables carry through until they *fall out of scope*
    - *scope* describes where in the program the variable has a value
    - More on this later
    - For now, it’s enough to realize that variables keep their values from one line to the next
Your Turn, Again!

- Write a Java program called *BreakName*, which asks the user for his/her name in the form *First M. Last*, and outputs First, M., and Last on three different lines.
- In other words, after the name is read from input, the program needs to break it up in the three pieces (First, M., and Last) and output those one line at a time.
import java.util.Scanner;
public class BreakName
{
    public static void main(String[] args)
    {
        
    }
}
Program source code is primarily for human beings

- Remember – the compiler is needed to turn source code (readable by humans) into object code (understandable to the machine)

It is important to write program source in ways that make it readable for other programmers

- Code re-use is an important part of modern software development
Documentation and Style

- It is important to write program source in ways that make it readable for other programmers
  - Programmers should use good style when writing their programs to make their code easier to read by other programmers
    - Using a standard style for naming variables, formatting indentation and line breaks (“white space”) and other formatting issues improves the readability of code.
  - Programmers should provide comments in their code to indicate what the code is supposed to be doing
    - Always remember – you can read source code to see what a program does, but that doesn’t tell you what the programmer intended for the code to do. If the code is broken, the actual operation may differ dramatically from the intent!
Documentation and Style

- Use meaningful names for variables, programs, etc.
- Use indentation and line spacing as shown in the examples in the text
- Always include a comment that describes what the program is supposed to do at the top of the program file
  - This comment should also include the name of the programmer who wrote the program
- Use all lower case for variables, except capitalize internal words (eggsPerBasket)
Comments

- *Comment*—text in a program that the compiler ignores
- Does not change what the program does, only explains the program
- Write meaningful and useful comments
- Comment the *non*-obvious
- Assume a *reasonably* knowledgeable reader
- // for single-line comments
- /* … */ for multi-line comments
Documentation and Style

- Pay attention to your programming style and commenting practices!
  - Full credit on assignments can only be earned by programs that practice good style and are well documented!