Expressions,

Parameters

and

Return Values

Expressions

Determine the Ouputs

```
int limeTray = 17;
limeTray = limeTray + 1;
System.out.print("Lime: " + limeTray + " blocks.");
   .
   .
   .
limeTray = limeTray - 7;
System.out.print("Lime: " + limeTray + " blocks.");
```

Increment and decrement

shortcuts to increase or decrease a variable's value by 1

Shorthand	Equivalent longer version
`variable++;`	`variable = variable + 1;`
`variable--;`	`variable = variable - 1;`

```
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-assign

shortcuts to modify a variable's value

Shorthand	Equivalent longer version
variable += **value**;	**variable** = **variable** + **value**;
variable -= **value**;	**variable** = **variable** - **value**;
variable *= **value**;	**variable** = **variable** * **value**;
variable /= **value**;	**variable** = **variable** / **value**;
variable %= **value**;	**variable** = **variable** % **value**;

```
x += 3;                    // x = x + 3;

gpa -= 0.5;                // gpa = gpa - 0.5;

number *= 2;               // number = number * 2;
```

Using the Shortcuts

```
int limeTray = 17;
limeTray++;  limeTray = limeTray + 1;
System.out.print("Lime: " + limeTray + " blocks.");
   .
   .
   .
limeTray--;  limeTray = limeTray - 7;
System.out.print("Lime: " + limeTray + " blocks.");
```

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

- 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

Name	Description		Examples
`int`	integers	(up to $2^{31} - 1$)	`42, -3, 0, 926394`
`double`	real numbers	(up to 10^{308})	`3.1, -0.25, 9.4e3`
`char`	single text characters		`'a', 'X', '?', '\n'`
`boolean`	logical values		`true, false`

- Why does Java distinguish integers vs. real numbers?

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 `+ -`

Mixing types

- When `int` and `double` are mixed, the result is a `double`.
  - `4.2 * 3` is `12.6`
- The conversion is per-operator, affecting only its operands.

```
7 / 3 * 1.2 + 3 / 2
 \_/
  2    * 1.2 + 3 / 2
   \_____/
    2.4       + 3 / 2
                 \_/
                  1
    2.4       +
     \_____________/
          3.4
```

```
2.0 + 10 / 3 * 2.5 - 6 / 4
        \____/
2.0 +    3    * 2.5 - 6 / 4
         \_______/
2.0 +     7.5      - 6 / 4
                       \_/
                        1
2.0 +     7.5      -
 \_____________/
      9.5          -   1
       \_________________/
            8.5
```

String concatenation

- **string concatenation**: Using + between a string and another value to make a longer string.

```
"hello" + 42        "hello42"
1 + "abc" + 2        "1abc2"
"abc" + 1 + 2         "abc12"
1 + 2 + "abc"            "3abc"
"abc" + 9 * 3              "abc27"
"1" + 1                      "11"
4 - 1 + "abc"                  "3abc"
```

- Use + to print a string and an expression's value together.
  - `System.out.println("Grade: " + (95.1 + 71.9) / 2);`
  - Output: `Grade: 83.5`

Variables

Receipt example

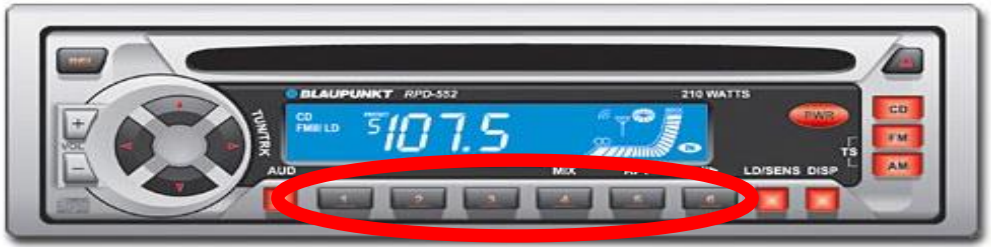
What's bad about the following code?

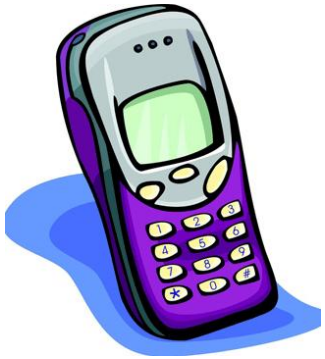
```
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed, assuming 8% tax / 15% tip
        System.out.println("Subtotal:");
        System.out.println(38 + 40 + 30);
        System.out.println("Tax:");
        System.out.println((38 + 40 + 30) * .08);
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

- The subtotal expression `(38 + 40 + 30)` is repeated
- So many `println` statements

Variables

- **variable**: A piece of the computer's memory that is given a name and type, and can store a value.
  - Like preset stations on a car stereo, or cell phone speed dial:





  - Steps for using a variable:
    - *Declare* it - state its name and type
    - *Initialize* it - store a value into it
    - *Use* it - print it or use it as part of an expression

Assignment and algebra

- Assignment uses = , but it is not an algebraic equation.

  = means, *"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?

```
int x = 3;
x = x + 2;    // ???
```

x	5

Assignment and types

- A variable can only store a value of its own type.
  - `int x = 2.5;  // ERROR: incompatible types`

- An `int` value can be stored in a `double` variable.
  - The value is converted into the equivalent real number.
  - `double myGPA = 4;`

myGPA	4.0

  - `double avg = 11 / 2;`

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.
  - ```
    double grade = (95.1 + 71.9 + 82.6) / 3.0;
    System.out.println("Your grade was " + grade);

    int students = 161 + 147;
    System.out.println("There are " + students +
                       " students in the course.");
    ```

- Output:

```
Your grade was 83.2
There are 308 students in the course.
```
```

Receipt question

Improve the receipt program using variables.

```
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) * .15 +
                           (38 + 40 + 30) * .08);
    }
}
```

Receipt answer

```
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);
    }
}
```

Scope

- **scope**: The part of a program where a variable exists.
  - From its declaration to the end of the `{ }` braces
    - A variable declared in a `for` loop exists only in that loop.
    - A variable declared in a method exists only in that method.

```
public static void example() {
    int x = 3;
    for (int i = 1; i <= 10; i++) {
        System.out.println(x);
    }
    // i no longer exists here
} // x ceases to exist here
```

i's scope

x's scope

Scope implications

- Variables without overlapping scope can have same name.

```
for (int i = 1; i <= 100; i++) {
    System.out.print("/");
}
for (int i = 1; i <= 100; i++) {    // OK
    System.out.print("\\");
}
int i = 5;                          // OK: outside of loop's scope
```

- A variable can't be declared twice or used out of its scope.

```
for (int i = 1; i <= 100 * line; i++) {
    int i = 2;                      // ERROR: overlapping scope
    System.out.print("/");
}
i = 4;                              // ERROR: outside scope
```

Consider Assignment 1 Pi

```
public class A1p1_Pi_Approx {
  public static void main( String [] args) {

      //Initialize Variables
      double Pi = 4.0;
      double denominator = 3;
      double numerator = 1.0;
      double sum = 1;

      //Add one term to series and prepare for the next one
      sum += numerator * -1 / denominator;
      denominator += 2;
      numerator *=-1;

      sum += numerator * -1 / denominator;
      denominator += 2;
      numerator *=-1;

      //Lots more terms, then Output result
```

Can't just move this to a method without scope issues

Parameters: Assignment 1 Pi

```
public static void main( String [] args) {

    //Initialize Variables
    double Pi = 4.0;
    double denominator = 3;
    double numerator = 1.0;
    double sum = 1;

    //Add terms to series
    addTerm(sum, numerator, denominator);
    addTerm(sum, numerator, denominator);
    //  . . . etc. . .

    //Output result
    System.out.println("Pi is approximately " + Pi*sum);
 }

 public static void addTerm(double series,
                            double nextTerm, double denom)
```

<see next slide for method >

Parameters: Assignment 1 Pi

```
p. s. v. main(…) {
   // from previous slide
    addTerm(sum, numerator, denominator);
    addTerm(sum, numerator, denominator);
    // . . . .
}



public static void addTerm(double series,
                           double nextTerm, double denom) {

    sum += numerator * -1 / denominator;
    denominator += 2;
    numerator *=-1;
}
```

But there is a ~~small~~ big problem . . . Let's learn about return values

First, another Parameter Example

- Consider the task of drawing the following scalable figure:

```
+/\/\/\/\/\/\/\/\/\+
|                   |
|                   |
|                   |
|                   |
|                   |
+/\/\/\/\/\/\/\/\/\+
```

Multiples of 5 occur many times

```
+/\/\/\/\+
|         |
|         |
+/\/\/\/\+
```

The same figure at size 2

Repetitive figure code

```
public class Sign {


    public static void main(String[] args) {
        drawLine();
        drawBody();
        drawLine();
    }

    public static void drawLine() {
        System.out.print("+");
        for (int i = 1; i <= 10; i++) {
            System.out.print("/\\");
        }
        System.out.println("+");
    }

    public static void drawBody() {
        for (int line = 1; line <= 5; line++) {
            System.out.print("|");
            for (int spaces = 1; spaces <= 20; spaces++) {
                System.out.print(" ");
            }
            System.out.println("|");
        }
    }
}
```

Using a parameter to control

```
public class Sign {

    public static void main(String[] args) {
        int size = 5;
        drawLine(size);
        drawLine(27);
        drawLine(size+5);
        drawBody(size);
        drawLine(size);
    }

    public static void drawLine(int width) {
        System.out.print("+");
        for (int i = 1; i <= width * 2; i++) {
            System.out.print("/\\");
        }
        System.out.println("+");
    }

    public static void drawBody(int height) {
        for (int line = 1; line <= height; line++) {
            System.out.print("|");
            for (int spaces = 1; spaces <= height * 4; spaces++) {
                System.out.print(" ");
            }
            System.out.println("|");
        }
    }
}
```

Parameters: Assignment 1 Pi

```
p. s. v. main(…) {
   // from previous slide
    sum = addTerm(sum, numerator, denominator);
    sum = addTerm(sum, numerator, denominator);
    // . . . .
}



public static double addTerm(double series,
                             double nextTerm, double denom) {

    series += numerator * -1 / denominator;
    denom += 2;
    nextTerm *=-1;
    return series;
}
```

More about return values next class