CS 106A, Lecture 4
Java Console Programs; Expressions and Variables

reading:

Art & Science of Java, 2.1 - 2.4, 3.1 - 3.4

Console Programs

Console programs

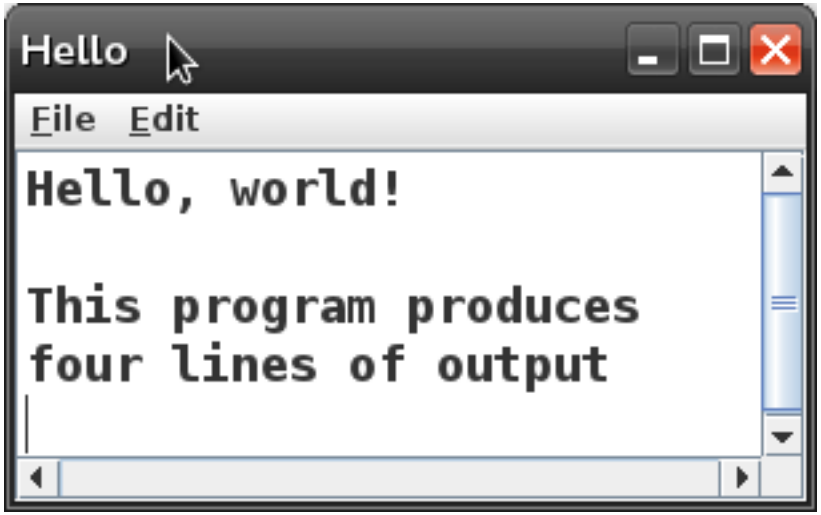
```
import acm.program.*;

public class Name extends ConsoleProgram {
    public void run() {
        statements;
    }
}
```

- Unlike Karel, many programs produce their behavior as text.
- **console**: Text box into which the behavior is displayed.
  - *output:* Messages displayed by the program.
  - *input:* Data read by the program that the user types.

The console

```
public class Hello extends ConsoleProgram {
    public void run() {
        println("Hello, world!");
        println();
        println("This program produces");
        println("four lines of output");
    }
}
```



- Its output:

```
Hello, world!

This program produces
four lines of output
```

- **console**: Text box into which the output is printed.

The println statement

- A statement that prints a line of output on the console.
  - pronounced "print-linn"

- Two ways to use `println` :
  - `println("`***`text`***`");`
    - Prints the given message as output.
    - A message is called a *string*; it starts/ends with a `"` quote character.
    - The quotes do not appear in the output.
    - A string may not contain a `"` character.
  - `println();`

    Prints a blank line of output.

Escape sequences

- **escape sequence**: A special sequence of characters used to represent certain special characters in a string.

`\t`	tab character
`\n`	new line character
`\"`	quotation mark character
`\\`	backslash character

  - Example:

```
println("\\hello\nhow\tare \"you\"?\\\\");
```

  - Output:

```
\hello
how     are "you"?\\
```

ConsoleProgram methods

- The `ConsoleProgram` contains these useful methods:

Method	Description
`pause(`***`ms`***`);`	halts for the given # of milliseconds
`setFont("`***`font`***`");`	changes text size/shape *("name-style-size")*
`setSize(`***`width, height`***`);`	sets console window's onscreen size
`setTitle("`***`text`***`");`	sets title bar text
`clearConsole();`	erase any text from the console

```
public class Hello extends ConsoleProgram {
    public void run() {
        setFont("Comic Sans MS-Bold-16");
        setSize(500, 300);
        println("Hello, world!");
        ...
```

Expressions and Variables

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"    → 0110100001101001
```

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?

Expressions, operators

- **expression**: A value or operation that computes a value.
  - Examples: `1 + 4 * 5`

    `(7 + 2) * 6 / 3`

  - The simplest expression is a *literal value*., like 42 .
  - A complex expression can use operators and parentheses.

- **operator**: Combines multiple values or expressions.

  `+ - * / %`

  add, subtract/negate, multiply, divide, modulus (remainder)

- As a program runs, its expressions are *evaluated*.

  `1 + 1` evaluates to `2` . (How would we print the output 1 + 1?)

Integer division

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

  `14 / 4` is `3`, not `3.5`. *(Java ALWAYS rounds down.)*

```
   3
4 ) 14
   12
    2
```

```
   4
10 ) 45
   40
    5
```

```
   52
27 ) 1425
   135
    75
    54
    21
```

- More examples:
  - `32 / 5` is `6`
  - `84 / 10` is `8`
  - `156 / 100` is `1`

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

Integer remainder %

- The % operator computes the remainder from integer division.

```
14 % 4                    is 2

218 % 5                   is 3
```

```
    3
  ----
4 ) 14
    12
    --
     2
```

```
    43
  ----
5 ) 218
    20
    --
     18
     15
     --
      3
```

- Applications of % operator:
  - Obtain last digit of a number: `230857 % 10` is `7`
  - Obtain last 4 digits: `658236489 % 10000` is `6489`
  - See whether a number is odd: `7 % 2` is `1`, but `42 % 2` is `0`

Precedence

- **precedence**: Order in which operators are evaluated.
  - Generally operators evaluate left-to-right.

    `1 - 2 - 3` is `(1 - 2) - 3` which is `-4`

  - But `* / %` have a higher level of precedence than `+ -`

    `1 + 3 * 4` is 13

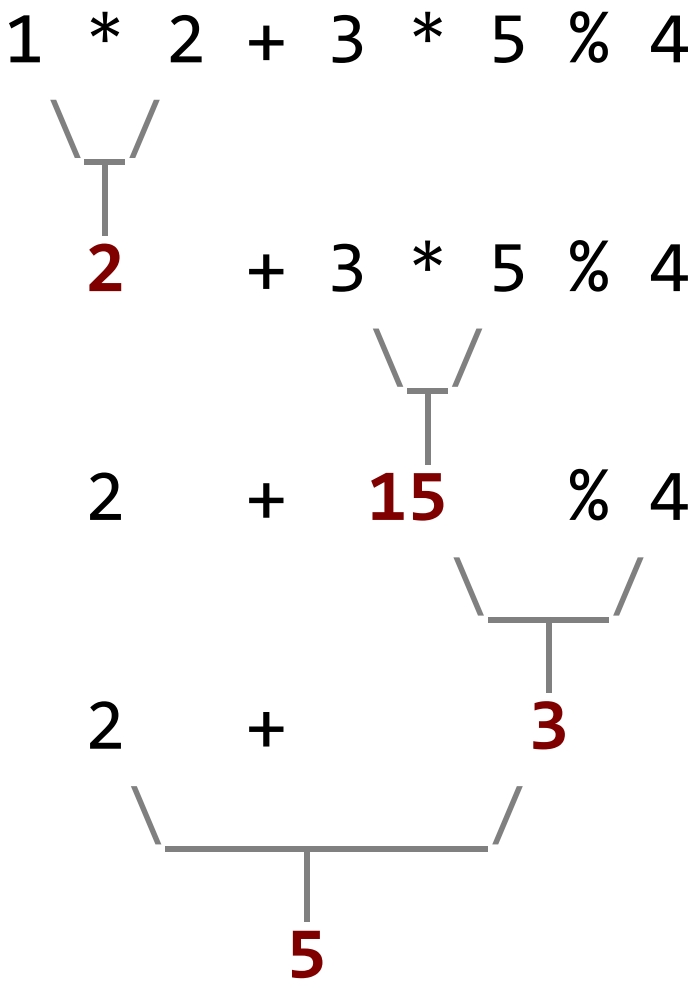
    ```
    6 + 8 / 2 * 3
    6 +   4   * 3
    6 +     12      is 18
    ```

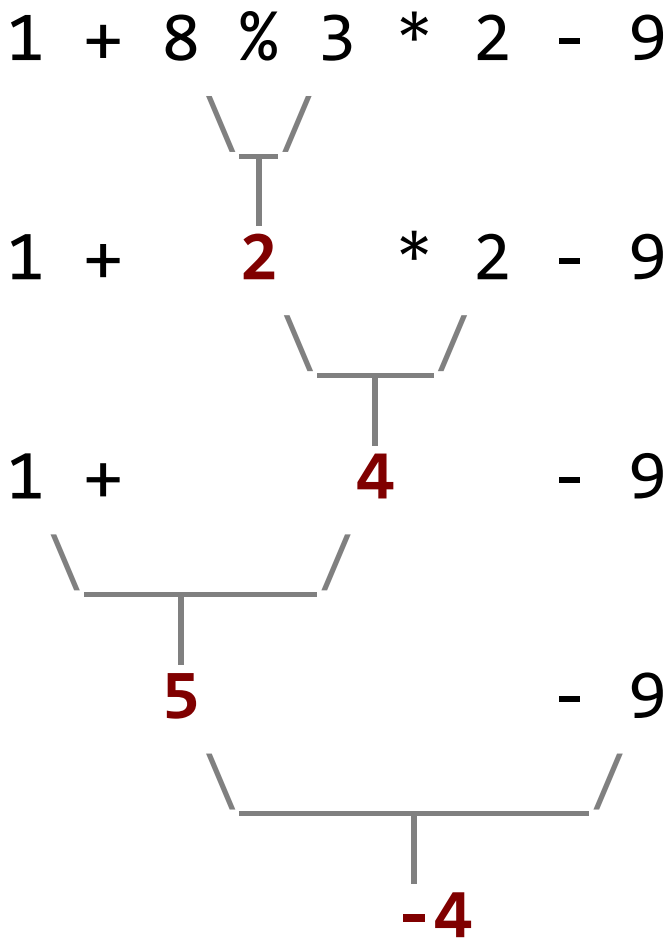
  - Parentheses can alter order of evaluation, but spacing does not:

    `(1 + 3) * 4` is 16

    `1+3 * 4-2` is 11

Precedence examples





Real numbers: double

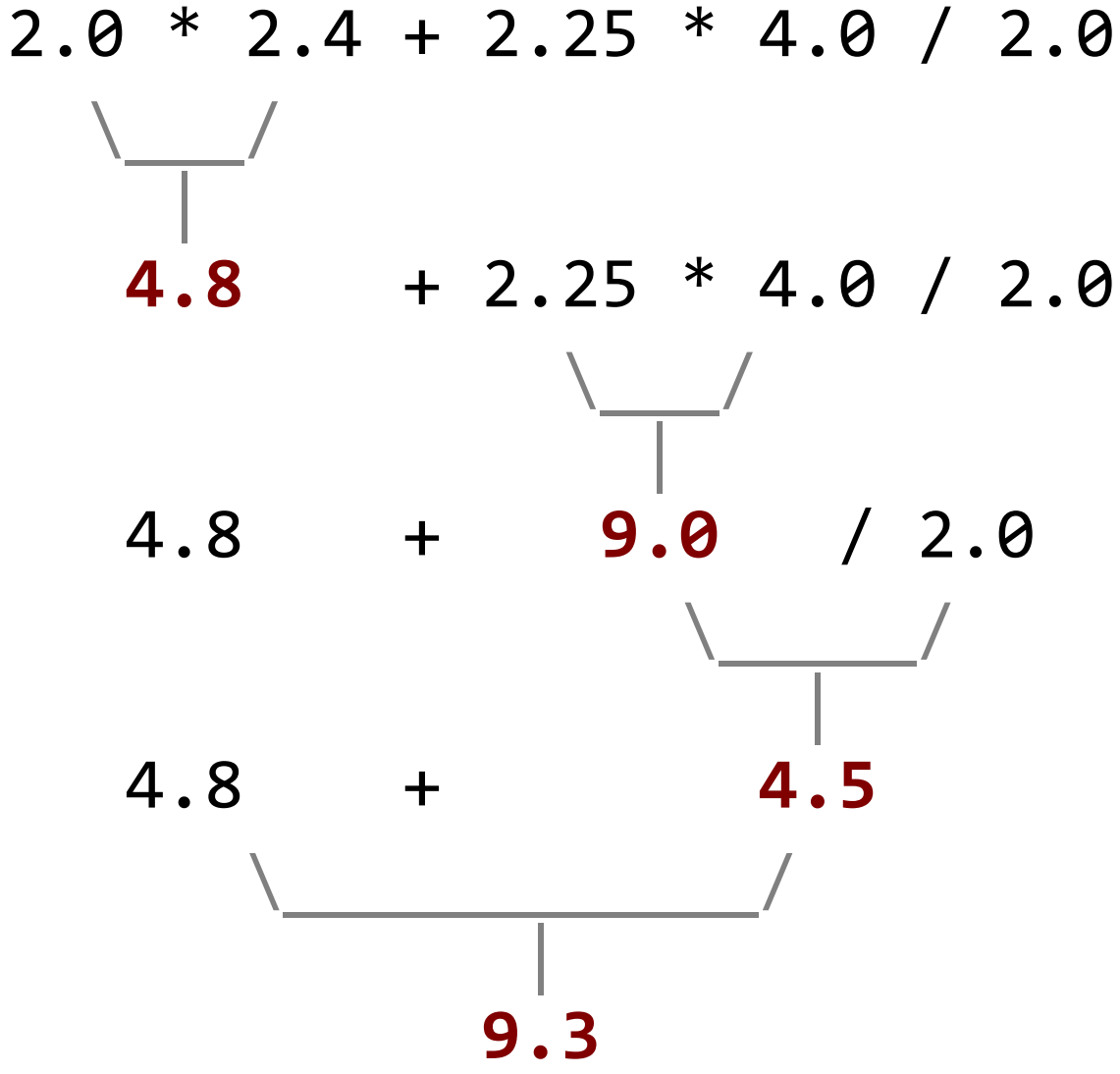
- Examples: `6.022` , `-42.0` , `2.143e17`
  - Placing `.0` or `.` after an integer makes it a `double`.

- The operators `+ - * / % ()` all still work with `double`.

  `/` produces an exact answer: `15.0 / 2.0` is `7.5`

  - Precedence is the same: `()` before `* / %` before `+ -`

Example w/ double

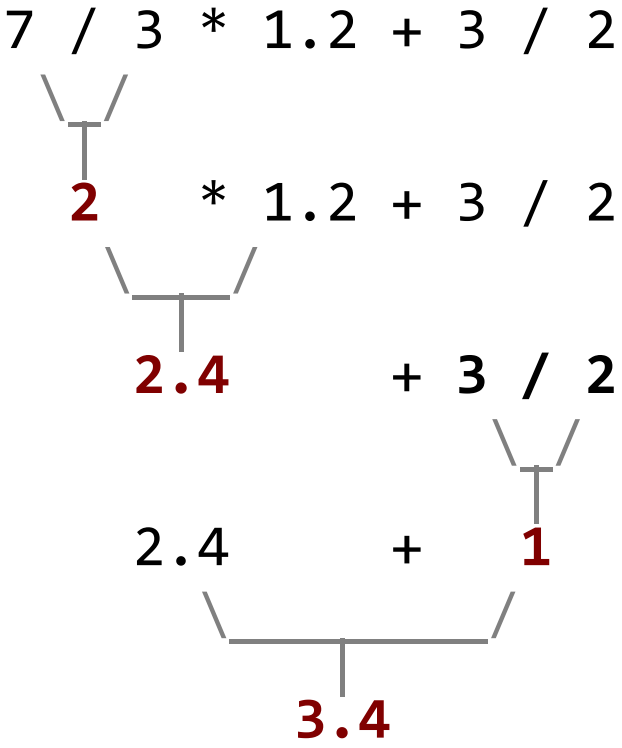


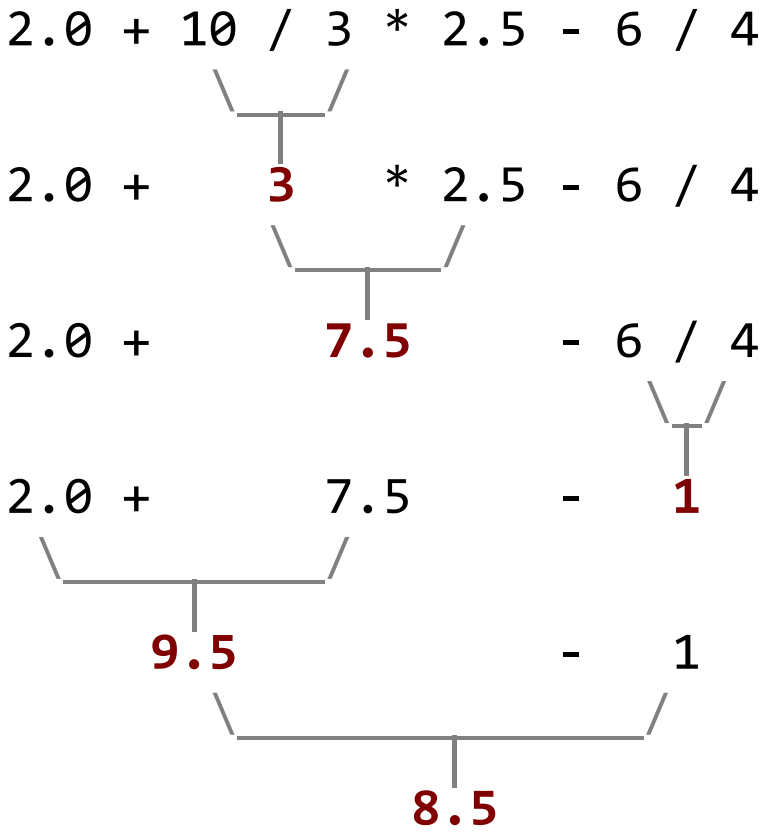
Mixing types

- When `int` and `double` are mixed, the result is a `double`.

  `3 * 4.2` is `12.6`

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





`3 / 2` is 1 above, not `1.5` .

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"
```

- Use + to print a string and an expression's value together.

```
println("Average: " + (95.1 + 71.9) / 2);
```

Output:

```
Average: 83.5
```

Redundant expressions

- What's bad about the following code?

```
public class Receipt extends ConsoleProgram {
    public void run() {
        println("Subtotal:");              // Compute total
        println(38 + 40 + 30);             // owed, assuming
        println("Tax:");                   // 8% tax and
        println((38 + 40 + 30) * .08);     // 15% tip
        println("Tip:");
        println((38 + 40 + 30) * .15);
        println("Total:");
        println(38 + 40 + 30 +
                (38 + 40 + 30) * .08 +
                (38 + 40 + 30) * .15);
    }
}
```

  - The subtotal expression `(38 + 40 + 30)` and others are 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 mobile 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

Declaration

- **variable declaration**: Sets aside memory for storing a value.
  - Variables must be declared before they can be used.

- Syntax:

```
type name;
```

```
int zipcode;
```

zipcode 

```
double myGPA;
```

myGPA 

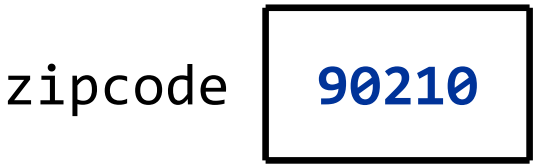
Assignment

- **assignment**: Stores a value into a variable.
  - The value can be an expression; the variable stores its result.

- Syntax:

```
name = expression;
```

```
int zipcode;
zipcode = 90210;
```




```
double myGPA;
myGPA = 1.0 + 2.25;
```

Declare / initialize

- A variable can be declared/initialized in one statement.
  - This is probably the most commonly used declaration syntax.

- Syntax:

```
type name = expression;
```

```
double tempF = 98.6;
```


```
int x = (11 / 2) + 3;
```

Using variables

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

```
int x = 3;
println("x is " + x);      // x is 3
println(5 * x - 1);        // 5 * 3 - 1
```

- You can assign a value more than once:

```
int x = 3;
println(x + " here");      // 3 here

x = 4 + 7;
println("now x is " + x); // now x is 11
```

Assignment and algebra

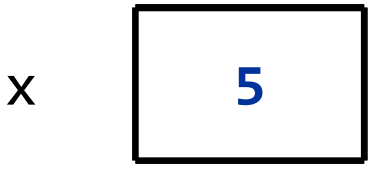
- Assignment uses = , but it is not an algebraic equation.

  = means, *"store the value at right in the 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;    // ???
```



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;
```


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

  - 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;
println(x);   // Error: x has no value
```

- You may not declare the same variable twice.

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

int y = 3;
int y = 5;    // Error: y already exists
```

  - How can this code be fixed?

Printing a variable

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

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

int enrolled = 11 + 17 + 4 + 19 + 14;
println("There are " + enrolled + " students.");
```

- Output:

```
Your grade was 83.2
There are 65 students.
```

Exercise: Receipt

- Improve the `Receipt` program using variables:

```
public class Receipt extends ConsoleProgram {
    public void run() {
        println("Subtotal:");              // Compute total
        println(38 + 40 + 30);             // owed, assuming
        println("Tax:");                   // 8% tax and
        println((38 + 40 + 30) * .08);     // 15% tip
        println("Tip:");
        println((38 + 40 + 30) * .15);
        println("Total:");
        println(38 + 40 + 30 +
                (38 + 40 + 30) * .08 +
                (38 + 40 + 30) * .15);
    }
}
```

Receipt solution

```
import acm.program.*;

public class Receipt extends ConsoleProgram {
    public void run() {
        // 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);
    }
}
```

Interactive programs

- **interactive program**: Reads input from the console.
  - The program pauses, waiting for the user to type a value.
  - The value typed by the user is stored in a variable.

- `ConsoleProgram` methods for reading user input:

Method	Description
`readInt("msg")`	reads an `int` value from the user
`readDouble("msg")`	reads a `double` value from the user
`readLine("msg")`	reads a one-line `String`
`readBoolean("msg")` `readBoolean("msg", "y", "n")`	reads a `boolean` value from the user

```
int ssn = readInt("Type your social security #: ");
```

User input example

```
public class UserInputExample extends ConsoleProgram {
    public void run() {
        int age = readInt("How old are you? ");
        int years = 65 - age;
        println(years + " years until retirement!");
    }
}
```

age	29
years	36

- Console (user input underlined):

```
How old are you?  29
36 years until retirement!
```

Exercise: Receipt2

- Modify our `Receipt` program so it prompts the user to type the subtotal cost of the meal and computes the total based on that.

```
What was the meal cost? 50
Tax: 4.0
Tip: 7.5
Total: 61.5

What was the meal cost? 125
Tax: 10.0
Tip: 18.75
Total: 153.75
```

Receipt2 solution

```
import acm.program.*;

public class Receipt2 extends ConsoleProgram {
    public void run() {
        // Calculate total owed, assuming 8% tax / 15% tip
        int subtotal = readInt("What was the meal cost? $");
        double tax = subtotal * .08;
        double tip = subtotal * .15;
        double total = subtotal + tax + tip;

        println("Tax: $" + tax);
        println("Tip: $" + tip);
        println("Total: $" + total);
    }
}
```

More practice

FibonacciSequence

- Write a console program **`FibonacciSequence`** that displays the numbers in the Fibonacci Sequence up to a given max.
  - The first two terms in the sequence are **0** and **1**.
  - Every subsequent term is the **sum** of the previous two terms.

```
This program lists the Fibonacci sequence.
Max value? 10000
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
610 987 1597 2584 4181 6765
```

  - The Italian mathematician Leonardo Fibonacci devised the Fibonacci sequence as a way to model the growth of a population of rabbits.

Overflow (extra) slides

Precedence exercise

Q: What is the result of the following expression?

```
1 + 2 * 3 + 7 * 2 % 5
```

A. 1
B. 2
C. 5
D. 11
E. 21

Quirks of real numbers

- Some `double` values print poorly (too many digits).

```
double result = 1.0 / 3.0;
println(result);    // 0.3333333333333
```

- The computer represents `doubles` in an imprecise way.

```
println(0.1 + 0.2);
```

  - Instead of `0.3`, the output is `0.30000000000000004`

Type casting

- **type cast**: A conversion from one type to another.
  - To promote an `int` into a `double` to get exact division from /
  - To truncate a `double` from a real number to an integer

- Syntax:

```
(type) expression
```

Examples:

```
double result = (double) 19 / 5;       // 3.8
int result2 = (int) result;            // 3
int x = (int) (10.0 * 10.0 * 10.0);    // 1000
```

More about type casting

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

```
double x = (double) 1 + 1 / 2;          // 1.0
double y = 1 + (double) 1 / 2;          // 1.5
```

- You can use parentheses to force evaluation order.

```
double average = (double) (a + b + c) / 3;
```

- A conversion to `double` can be achieved in other ways.
  - Common trick: Multiplying by `1.0`

```
double average = 1.0 * (a + b + c) / 3;
```