Example: Interest Program

• Example - Write a program that calculates the interest that the Canarsie Indians would have accumulated if they had put the $24 that they had received for Manhattan Island in the bank at 5% interest.

  Input - none; all the values are fixed
  Output - Year and Principle
  Other Information -
    Principle is initially 24
    Interest = Interest Rate * Principle
    New Principle = Old Principle + Interest
Example: Interest Program

- Our initial algorithm is:
  1. Set the principle to 24
  2. For every year since 1625, add 5% interest to the principle and print out the principle.

Refining The Interest Algorithm

1. Set the principle to 24
2. For every year since 1625, add 5% interest to the principle and print out the principle.

2.1 FOR Year goes from 1625 TO Present:
   2.1.1 Add 5% interest to the principle
   2.1.2 Print the current principle
Refining The Interest Algorithm

1. Set the principle to 24

2.1 FOR Year goes from 1625 TO Present:

2.1.1 Calculate 5% Interest
2.1.2 Add the interest to the principle
2.1.3 Print the current principle

principle = 24;
Refining The Interest Algorithm

principle = 24;

2.1 FOR Year goes from 1625 TO Present:
2.1.1.1 Calculate 5% Interest
2.1.1.2 Add the interest to the principle
2.1.2 Print the current principle

for (year = 1625; year < present; year++) {

    interest = rate * principle;
    principle = principle + interest;
}

}
Refining The Interest Algorithm

```java
principle = 24;
for (year = 1625; year < present; year++) {
    interest = rate * principle;
    principle = principle + interest;
}
2.1.2 Print the current principle
}
System.out.println("year = " + year + 
    + "\tprinciple = " + principle);
```

The Interest Program

```java
public class Interest {
    // Calculate the interest that the Canarsie Indians could have accrued if they had deposited the $24 in an bank account at 5% interest.
    public static void main(String[] args) {
        final int present = 2014;
        int year;
        final double rate = 0.05;
        double interest, principle;

        // Set the initial principle at $24
        principle = 24;
```
// For every year since 1625, add 5% interest // to the principle and print out // the principle

for (year = 1625; year < present; year++) {
    interest = rate * principle;
    principle = principle + interest;

    System.out.println("year = " + year
        + ", principle = "
        + principle);
}

Output from the Compound Interest Program

• What will our output look like?
  year = 1625 principle = 25.2
  year = 1626 principle = 26.46
  year = 1627 principle = 27.783
  year = 1628 principle = 29.172150000000002
  ... ...
  year = 2010 principle = 3.6247719022233915E9
  year = 2011 principle = 3.8060104973345613E9
  year = 2012 principle = 3.996311022201289E9
  year = 2013 principle = 4.1961265733113537E9

• This does not look the way we expect monetary amounts to be written!
System.out.printf()

• The method `System.out.printf()` gives us a way to write output that is formatted, i.e., we can control its appearance.

• We write the method:
  ```java
  System.out.printf(ControlString, Arg1, Arg2, ...)
  ```

• The control string is a template for our output, complete with the text that will appear along with whatever values we are printing.

System.out.printf(): Some Simple Examples

• `System.out.printf()` will print whatever is in the control string with a few exceptions:
  ```java
  System.out.printf("This is a test");
  System.out.printf("This is a test").
  ```

  will produce:

  ```text
  This is a test
  This is a test
  ```

  If you want these to be on two separate lines:
  ```java
  System.out.printf("This is a test\n");
  System.out.printf("This is a test\n").
  ```
Special Characters

- There are a number of special characters that all begin with a backslash:
  - \n  new line
  - \b  backspace
  - \t  tab
- These can appear anywhere with a string of characters:
  System.out.printf("This is a test\nIt is!!\n");

%d and %f

- The specifiers %d and %f allow a programmer to specify how many spaces a number will occupy and (in the case of float values) how many decimal places will be used.
  - %n%d  will use at least n spaces to display the integer value in decimal (base 10) format.
  - %w.c%f  will use at least w spaces to display the value and will have exactly d decimal places.
Changing the width

<table>
<thead>
<tr>
<th>Number</th>
<th>Formatting</th>
<th>Print as:</th>
</tr>
</thead>
<tbody>
<tr>
<td>182</td>
<td>%2d</td>
<td>182</td>
</tr>
<tr>
<td>182</td>
<td>%3d</td>
<td>182</td>
</tr>
<tr>
<td>182</td>
<td>%5d</td>
<td>`182</td>
</tr>
<tr>
<td>182</td>
<td>%7d</td>
<td>`182</td>
</tr>
<tr>
<td>-182</td>
<td>%4d</td>
<td>-182</td>
</tr>
<tr>
<td>-182</td>
<td>%5d</td>
<td>`-182</td>
</tr>
<tr>
<td>-182</td>
<td>%7d</td>
<td>`~-182</td>
</tr>
</tbody>
</table>

Changing the width (continued)

<table>
<thead>
<tr>
<th>Number</th>
<th>Formatting</th>
<th>Print as:</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>%1d</td>
<td>23</td>
</tr>
<tr>
<td>23</td>
<td>%2d</td>
<td>23</td>
</tr>
<tr>
<td>23</td>
<td>%6d</td>
<td>..23</td>
</tr>
<tr>
<td>23</td>
<td>%8d</td>
<td>......23</td>
</tr>
<tr>
<td>11023</td>
<td>%4d</td>
<td>11023</td>
</tr>
<tr>
<td>11023</td>
<td>%6d</td>
<td>11023</td>
</tr>
<tr>
<td>-11023</td>
<td>%6d</td>
<td>-11023</td>
</tr>
<tr>
<td>-11023</td>
<td>%10d</td>
<td>......11023</td>
</tr>
</tbody>
</table>
Changing The Precision

<table>
<thead>
<tr>
<th>Number</th>
<th>Formatting</th>
<th>Prints as:</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.718281828</td>
<td>%8.5f</td>
<td>`2.71828</td>
</tr>
<tr>
<td>2.718281828</td>
<td>%8.3f</td>
<td><code>.</code>2.718</td>
</tr>
<tr>
<td>2.718281828</td>
<td>%8.2f</td>
<td><code>.</code>2.72</td>
</tr>
<tr>
<td>2.718281828</td>
<td>%8.0f</td>
<td><code>.</code>3</td>
</tr>
<tr>
<td>2.718281828</td>
<td>%13.11f</td>
<td>2.71828182800</td>
</tr>
<tr>
<td>2.718281828</td>
<td>%13.12f</td>
<td>2.718281828000</td>
</tr>
</tbody>
</table>

The revised *Compound* program

```java
public class Interest {
    // Calculate the interest that the Canarsie Indians could have accrued if they had deposited the $24 in an bank account at 5% interest.
    public static void main(String[] args) {
        final int present = 2014;
        int year;
        final double rate = 0.05;
        double interest, principle;

        // Set the initial principle at $24
        principle = 24;
```
// For every year since 1625, add 5% interest
// to the principle and print out
// the principle

for (year = 1625; year < present; year++) {
    interest = rate * principle;
    principle = principle + interest;

    System.out.printf
        ("year = %4d principle = $%13.2f\n",
         year, principle);
}

The output from the Revised Compound Program

Our output now looks like this:

year = 1625  principle = $  25.20
year = 1626  principle = $  26.46
year = 1627  principle = $  27.78
year = 1628  principle = $  29.17
... ... ... ...
year = 2001  principle = $2336560287.43
year = 2002  principle = $2453388301.80
year = 2003  principle = $2576057716.89
year = 2004  principle = $2704860602.73
Nested Loops

• **Nested loop** - A loop placed inside another loop.

```java
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= 10; j++) {
        System.out.print("*");
    }
    System.out.println(); // to end the line
}
```

• **Output:**

```
**********
**********
**********
**********
**********
```

• The outer loop repeats 5 times; the inner one 10 times. "Sets and reps" exercise analogy

---

Nested for Loop Exercise

• What is the output of the following nested for loops?

```java
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= i; j++) {
        System.out.print("*");
    }
    System.out.println();
}
```

• **Output:**

```
*  
** 
*** 
**** 
***** 
```
Nested `for` Loop Exercise

• What is the output of the following nested `for` loops?
  ```java
  for (int i = 1; i <= 5; i++) {
      for (int j = 1; j <= i; j++) {
          System.out.print(i);
      }
      System.out.println();
  }
  ```

• *Output:*
  
  1
  22
  333
  4444
  55555

Common Errors

• Both of the following sets of code produce *infinite loops:*
  ```java
  for (int i = 1; i <= 5; i++) {
      for (int j = 1; i <= 10; j++) {
          System.out.print("*");
      }
      System.out.println();
  }
  ```

  ```java
  for (int i = 1; i <= 5; i++) {
      for (int j = 1; j <= 10; i++) {
          System.out.print("*");
      }
      System.out.println();
  }
  ```
Complex Lines

- What nested `for` loops produce the following output?
- We must build multiple complex lines of output using:
  - an outer "vertical" loop for each of the lines
  - inner "horizontal" loop(s) for the patterns within each line

```
    ...1
    ...2
    ...3
    ...4
    ...5
```

Inner loop (repeated characters on each line)

```
  1
  2
  3
  4
  5
```

Outer loop (loops 5 times because there are 5 lines)
Outer And Inner Loop

• First write the outer loop, from 1 to the number of lines.
  ```java
  for (int line = 1; line <= 5; line++) {
    ...
  }
  ```
• Now look at the line contents. Each line has a pattern:
  – some dots (0 dots on the last line), then a number
    ....1
    ...2
    ..3
    .4
    5
• Observation: the number of dots is related to the line number.

Mapping Loops To Numbers

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

• What statement in the body would cause the loop to print:
  ```java
  4 7 10 13 16
  ```
  ```java
  for (int count = 1; count <= 5; count++) {
    System.out.print((3 * count + 1) + " ");
  }
  ```
Loop Tables

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

- To see patterns, make a table of count and the numbers.
  - Each time count goes up by 1, the number should go up by 5.
  - But count * 5 is too great by 3, so we subtract 3.

<table>
<thead>
<tr>
<th>count</th>
<th>Number to print</th>
<th>5 * count</th>
<th>5 * count - 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>15</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
<td>20</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>22</td>
<td>25</td>
<td>22</td>
</tr>
</tbody>
</table>

Loop tables question

- What statement in the body would cause the loop to print: 17 13 9 5 1

- Let's create the loop table together.
  - Each time count goes up 1, the number printed should ...
  - But this multiple is off by a margin of ...

<table>
<thead>
<tr>
<th>count</th>
<th>Number to print</th>
<th>-4 * count</th>
<th>-4 * count + 21</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>17</td>
<td>-4</td>
<td>17</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>-8</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>-12</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>-16</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>-20</td>
<td>1</td>
</tr>
</tbody>
</table>
Nested for Loop Exercise

• Make a table to represent any patterns on each line.

<table>
<thead>
<tr>
<th>line</th>
<th># of dots</th>
<th>-1 * line</th>
<th>-1 * line + 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>-1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>-2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>-3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>-4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>-5</td>
<td>0</td>
</tr>
</tbody>
</table>

• To print a character multiple times, use a for loop.

```java
for (int j = 1; j <= 4; j++) {
    System.out.print(".");  // 4 dots
}
```

Nested for Loop Solution

• Answer:

```java
for (int line = 1; line <= 5; line++) {
    for (int j = 1; j <= (-1 * line + 5); j++) {
        System.out.print(".");  // 4 dots
    }
    System.out.println(line);
}
```

• Output:

```plaintext
....1
...2
..3
 .4
 5
```
Nested for Loop Exercise

• What is the output of the following nested for loops?
  for (int line = 1; line <= 5; line++) {
    for (int j = 1; j <= (-1 * line + 5); j++) {
      System.out.print(".");
    }
    for (int k = 1; k <= line; k++) {
      System.out.print(line);
    }
    System.out.println();
  }

  • Answer:
    ....1
    ...22
    ..333
    .4444
    55555

Nested for Loop Exercise

• Modify the previous code to produce this output:
  .....1
  ...2.
  ..3..
  .4...
  5....
Nested for Loop Exercise – the Answer

```java
for (int line = 1; line <= 5; line++) {
    for (int j = 1; j <= (-1 * line + 5); j++) {
        System.out.print(".");
    }
    System.out.print(line);
    for (int j = 1; j <= (line - 1); j++) {
        System.out.print(".");
    }
    System.out.println();
}
```

Drawing Complex Figures

- Use nested for loops to produce the following output.

- Why draw ASCII art?
  - Real graphics require a lot of finesse
  - ASCII art has complex patterns
  - Can focus on the algorithms

```
#=============#
| <><>          |
| <>....<>      |
| <>........<>  |
| <>.            |
| <>........<>  |
| <>....<>      |
| <><>          |
#=============#
```
Development Strategy

• Recommendations for managing complexity:
  1. Design the program (think about steps or methods needed).
     – Write an English description of steps required.
     – Use this description to decide the methods.
  2. Create a table of patterns of characters.
     Use table to write your for loops.

Pseudocode

• **Pseudocode** - An English description of an algorithm.
• **Example** - Drawing a 12 wide by 7 tall box of stars:

  ```
  print 12 stars.
  for (each of 5 lines) {
    print a star.
    print 10 spaces.
    print a star.
  }
  print 12 stars.
  ```

```
Algorithm in Pseudocode

1. Line
   • #, 16 =, #

2. Top half
   a. |
   b. spaces (decreasing)
   c. <>
   d. dots (increasing)
   e. <>
   f. spaces (same as above)
   g. |

3. Bottom half (top half upside-down)

4. Line
   • #, 16 =, #

Methods From Pseudocode

```java
public class Mirror {
    public static void main(String[] args) {
        line();
        topHalf();
        bottomHalf();
        line();
    }

    public static void topHalf() {
        for (int line = 1; line <= 4; line++) {
            // contents of each line
        }
    }

    public static void bottomHalf() {
        for (int line = 1; line <= 4; line++) {
            // contents of each line
        }
    }
}
```
public static void line() {
    // ...
}

Tables

• A table for the top half:
  – Compute spaces and dots expressions from line number

<table>
<thead>
<tr>
<th>line</th>
<th>spaces</th>
<th>line * -2 + 8</th>
<th>dots</th>
<th>4 * line - 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>
Writing the code

• Useful questions about the top half:
  – What methods? (think structure and redundancy)
  – Number of (nested) loops per line?

Partial Solution

// Prints the expanding pattern of <> for the top half of the figure.
public static void topHalf() {
    for (int line = 1; line <= 4; line++) {
        System.out.print("|");
        for (int space = 1; space <= (line * -2 + 8); space++) {
            System.out.print(" ");
        }
        System.out.print("<>");
        for (int dot = 1; dot <= (line * 4 - 4); dot++) {
            System.out.print(".");
        }
    }
}
System.out.print("<>");

for (int space = 1;
    space <= (line * -2 + 8); space++) {
    System.out.print(" ");
}

System.out.println("|");

Scaling The Mirror

• Let's modify our Mirror program so that it can scale.
  – The current mirror (left) is at size 4; the right is at size 3.
• We'd like to structure the code so we can scale the figure by changing the code in just one place.
Limitations of variables

• Idea: Make a variable to represent the size.
  – Use the variable's value in the methods.
• Problem: A variable in one method can't be seen in others.

```java
public static void main(String[] args) {
    int size = 4;
    topHalf();
    printBottom();
}

public static void topHalf() {
    for (int i = 1; i <= size; i++) {
        // ERROR: size not found
    }
}

public static void bottomHalf() {
    for (int i = size; i >= 1; i--) {
        // ERROR: size not found
    }
}
```
**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.

```java
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
```
Scope Implications

• Variables without overlapping scope can have same name.

```java
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
```

Scope Implications

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

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

• **Class constant** - A fixed value visible to the whole program.
  – Value can be set only at declaration; cannot be reassigned

• Syntax:
  ```
  public static final type name = value;
  ```

• Name is usually in ALL_UPPER_CASE

Class Constants Examples

```java
public static final int DAYS_IN_WEEK = 7;

public static final double INTEREST_RATE = 3.5;

public static final int SSN = 658234569;
```
Constants and Figures

• Consider the task of drawing the following scalable figure:

```
+/-/\/-/\/-/\/-/\/-/\/-/\+  Multiples of 5 occur many times
 |                       |
 |                       |
 |                       |
 |                       |
+/-/\/-/\/-/\/-/\+  
```

```
+/-/\/-/\  
 |    |
 |    |
+/-/\+  The same figure at size 2
```

Repetitive Figure Code

```java
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("/\");  
        }
    }

    public static void drawBody() {
    }
}
```
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("|");
    }
    System.out.println("|");
}

public class Sign {
    public static final int HEIGHT = 5;

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

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

- Modify the Mirror code to be resizable using a constant.

A mirror of size 4:

```
#================#  
|     <><>      |  
|    <>....<>    |  
|  <>........<>  |  
|<>............<>|  
|<>............<>|  
|  <>........<>  |  
|    <>....<>    |  
|     <><>      |  
#================#
```

A mirror of size 3:

```
#============#  
|    <><>    |  
|  <>....<>  |  
|<>........<>|  
|<>........<>|  
|  <>....<>  |  
|    <><>    |  
#============#
```
Using A Constant

- Constant allows many methods to refer to same value:

```java
public static final int SIZE = 4;

public static void main(String[] args) {
    topHalf();
    printBottom();
}

public static void topHalf() {
    for (int i = 1; i <= SIZE; i++) {  // OK
        ...
    }
}

public static void bottomHalf() {
    for (int i = SIZE; i >= 1; i--) {  // OK
        ...
    }
}
```

- Let's modify our loop table to use `SIZE`
  - This can change the amount added in the loop expression

<table>
<thead>
<tr>
<th>SIZE</th>
<th>line</th>
<th>spaces</th>
<th>dots</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1,2,3,4</td>
<td>6,4,2,0</td>
<td>0,4,8,12</td>
</tr>
<tr>
<td>3</td>
<td>1,2,3</td>
<td>4,2,0</td>
<td>0,4,8</td>
</tr>
</tbody>
</table>

```
#================#      #============#
|      <><>      |      |    <><>    |
|    <>....<>    |      |  <>....<>  |
|  <>........<>  |      |<>........<>|
|<>............<>|      |<>........<>|
|<>............<>|      |  <>....<>  |
|  <>........<>  |      |    <><>    |
|    <>....<>    |      #============#
|      <><>      |
#================#
```
Loop tables and constant

<table>
<thead>
<tr>
<th>SIZE</th>
<th>line</th>
<th>spaces</th>
<th>-2*line +</th>
<th>dots</th>
<th>4*line - 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1,2,3,4</td>
<td>6,4,2,0</td>
<td>8</td>
<td>0,4,8,12</td>
<td>4*line - 4</td>
</tr>
<tr>
<td>3</td>
<td>1,2,3</td>
<td>4,2,0</td>
<td>6</td>
<td>0,4,8</td>
<td>4*line - 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SIZE</th>
<th>line</th>
<th>spaces</th>
<th>-2<em>line + (2</em>SIZE)</th>
<th>dots</th>
<th>4*line - 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1,2,3,4</td>
<td>6,4,2,0</td>
<td>8</td>
<td>0,4,8,12</td>
<td>4*line - 4</td>
</tr>
<tr>
<td>3</td>
<td>1,2,3</td>
<td>4,2,0</td>
<td>6</td>
<td>0,4,8</td>
<td>4*line - 4</td>
</tr>
</tbody>
</table>

Partial Solution

```java
public static final int SIZE = 4;

// Prints the expanding pattern of <> for
// the top half of the figure.
public static void topHalf() {
    for (int line = 1; line <= SIZE; line++) {
        System.out.print("|");

        for (int space = 1;
             space <= (line * -2 + (2*SIZE));
             space++) {
            System.out.print(" ");
        }

        System.out.print("<>");
    }
}
```

for (int dot = 1; dot <= (line * 4 - 4); dot++) {
    System.out.print(".");
}
System.out.print("<>");
for (int space = 1; space <= (line * -2 + (2*SIZE)); space++) {
    System.out.print(" ");
}
System.out.println("|");

Observations About Constant

• It doesn't replace every occurrence of the original value.

for (int dot = 1; dot <= (line * 4 - 4); dot++) {
    System.out.print(".");
}
Observations About Constant

• It doesn't replace *every* occurrence of the original value.

```java
for (int dot = 1; dot <= (line * 4 - 4); dot++)
{
    System.out.print(".");
}
```