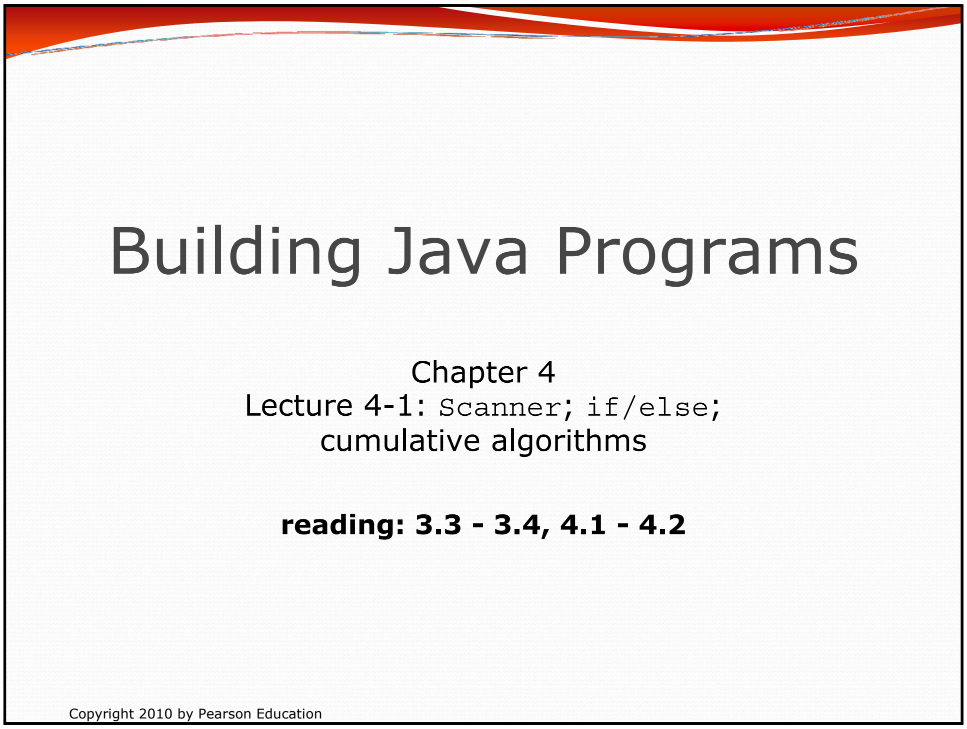
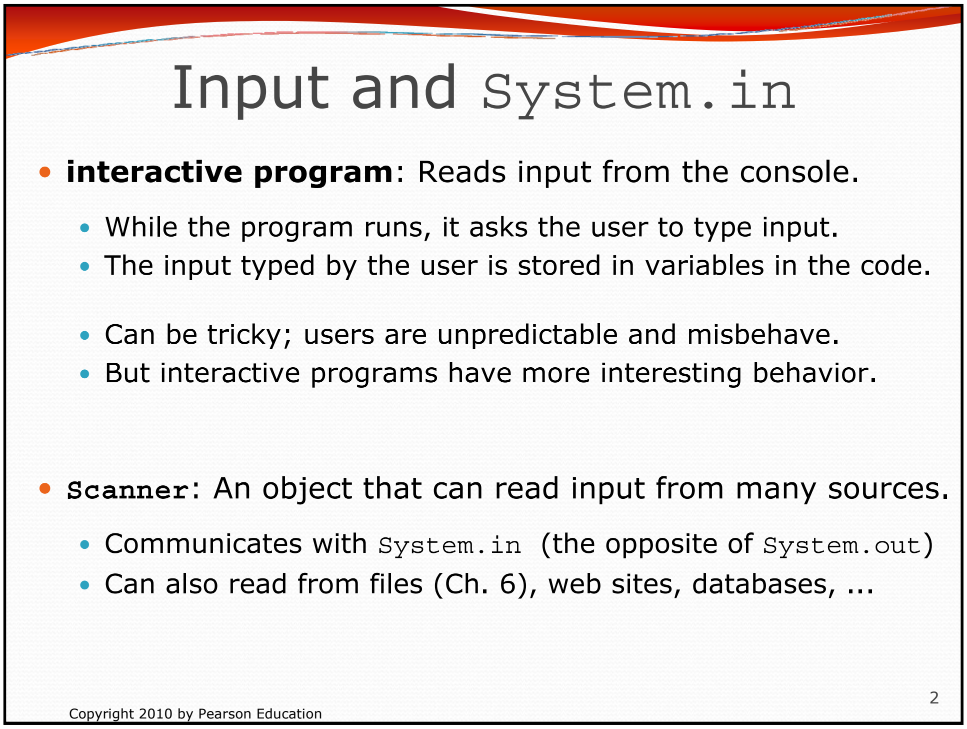


Building Java Programs

Chapter 4
Lecture 4-1: `Scanner`; `if/else`;
cumulative algorithms

reading: 3.3 - 3.4, 4.1 - 4.2

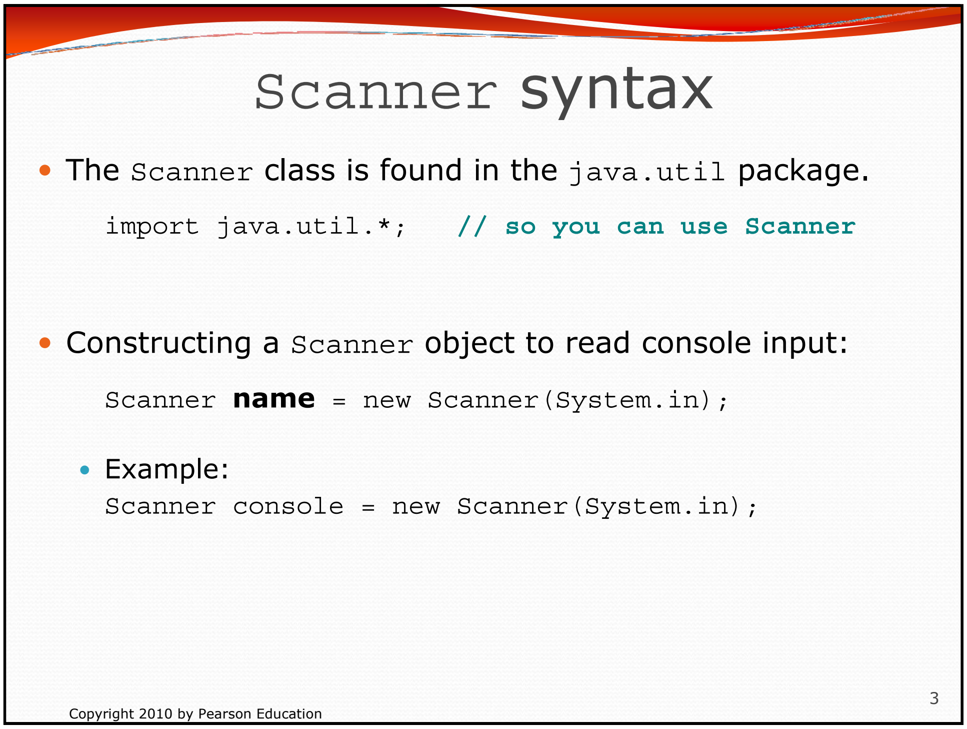


Input and `System.in`

- **interactive program**: Reads input from the console.
  - While the program runs, it asks the user to type input.
  - The input typed by the user is stored in variables in the code.

  - Can be tricky; users are unpredictable and misbehave.
  - But interactive programs have more interesting behavior.

- **`Scanner`**: An object that can read input from many sources.
  - Communicates with `System.in` (the opposite of `System.out`)
  - Can also read from files (Ch. 6), web sites, databases, ...



Scanner syntax

- The `Scanner` class is found in the `java.util` package.

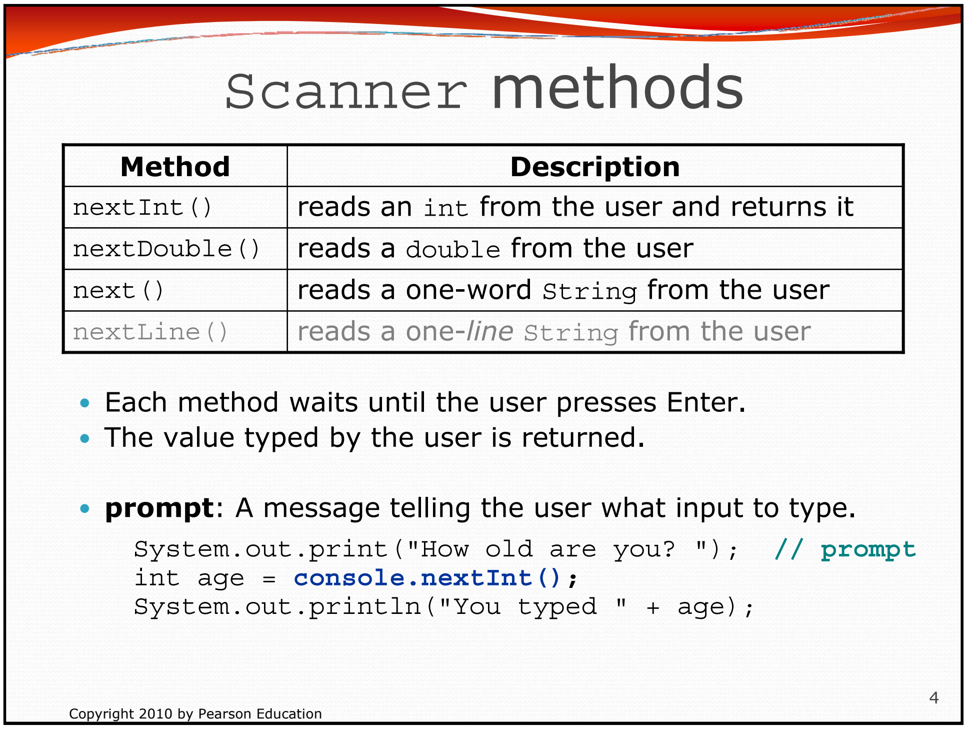
```
import java.util.*;    // so you can use Scanner
```

- Constructing a `Scanner` object to read console input:

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

  - Example:

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



Scanner methods

Method	**Description**
`nextInt()`	reads an `int` from the user and returns it
`nextDouble()`	reads a `double` from the user
`next()`	reads a one-word `String` from the user
`nextLine()`	reads a one-*line* `String` from the user

- Each method waits until the user presses Enter.
- The value typed by the user is returned.

- **prompt**: A message telling the user what input to type.

```
System.out.print("How old are you? ");  // prompt
int age = console.nextInt();
System.out.println("You typed " + age);
```

Scanner example

```
import java.util.*;    // so that I can use Scanner

public class UserInputExample {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);

        System.out.print("How old are you? ");
        int age = console.nextInt();

        int years = 65 - age;
        System.out.println(years + " years until retirement!");
    }
}
```

age 29

years 36

- Console (user input underlined):

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

Scanner example 2

- The Scanner can read multiple values from one line.

```
import java.util.*;    // so that I can use Scanner

public class ScannerMultiply {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);

        System.out.print("Please type two numbers: ");
        int num1 = console.nextInt();
        int num2 = console.nextInt();

        int product = num1 * num2;
        System.out.println("The product is " + product);
    }
}
```

- Output (user input underlined):

```
Please type two numbers: 8 6
The product is 48
```

Input tokens

- **token**: A unit of user input, as read by the `Scanner`.
  - Tokens are separated by *whitespace* (spaces, tabs, new lines).
  - How many tokens appear on the following line of input?

```
23  John Smith   42.0  "Hello world"  $2.50   "   19"
```

- When a token is not the type you ask for, it crashes.

```
System.out.print("What is your age? ");
int age = console.nextInt();
```

Output:

```
What is your age? Timmy
java.util.InputMismatchException
        at java.util.Scanner.next(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        ...
```

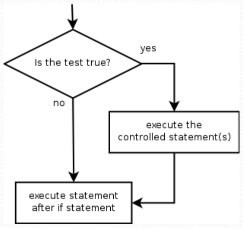
The `if/else` statement

reading: 4.1

The if statement

Executes a block of statements only if a test is true

```
if (test) {
    statement;
    ...
    statement;
}
```


Is the test true?
yes
no
execute the controlled statement(s)
execute statement after if statement

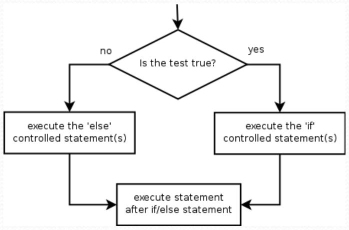
- Example:

```
double gpa = console.nextDouble();
if (gpa >= 2.0) {
    System.out.println("Application accepted.");
}
```

The if/else statement

Executes one block if a test is true, another if false

```
if (test) {
    statement(s);
} else {
    statement(s);
}
```


Is the test true?
no
yes
execute the 'else' controlled statement(s)
execute the 'if' controlled statement(s)
execute statement after if/else statement

- Example:

```
double gpa = console.nextDouble();
if (gpa >= 2.0) {
    System.out.println("Welcome to Mars University!");
} else {
    System.out.println("Application denied.");
}
```

Relational expressions

- `if` statements and `for` loops both use logical tests.

```
for (int i = 1; i <= 10; i++) { ...
if (i <= 10) { ...
```

  - These are `boolean` expressions, seen in Ch. 5.

- Tests use *relational operators*:

Operator	Meaning	Example	Value
`==`	equals	`1 + 1 == 2`	`true`
`!=`	does not equal	`3.2 != 2.5`	`true`
`<`	less than	`10 < 5`	`false`
`>`	greater than	`10 > 5`	`true`
`<=`	less than or equal to	`126 <= 100`	`false`
`>=`	greater than or equal to	`5.0 >= 5.0`	`true`

Logical operators

- Tests can be combined using *logical operators*:

Operator	Description	Example	Result
`&&`	and	`(2 == 3) && (-1 < 5)`	`false`
`||`	or	`(2 == 3) || (-1 < 5)`	`true`
`!`	not	`!(2 == 3)`	`true`

- "Truth tables" for each, used with logical values *p* and *q*:

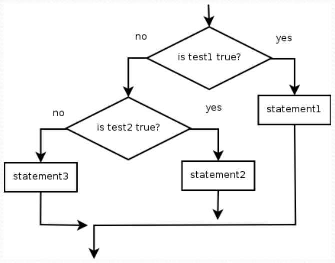
p	q	p `&&` q	p `||` q
`true`	`true`	`true`	`true`
`true`	`false`	`false`	`true`
`false`	`true`	`false`	`true`
`false`	`false`	`false`	`false`

p	`!`p
`true`	`false`
`false`	`true`

Nested `if/else`

Chooses between outcomes using many tests

```
if (test) {
    statement(s);
} else if (test) {
    statement(s);
} else {
    statement(s);
}
```



- Example:

```
if (x > 0) {
    System.out.println("Positive");
} else if (x < 0) {
    System.out.println("Negative");
} else {
    System.out.println("Zero");
}
```

Exercise

- Prompt the user to enter two people's heights in inches.
  - Each person should be classified as one of the following:
    - short (under 5'3")
    - medium (5'3" to 5'11")
    - tall (6' or over)
  - The program should end by printing which person is taller.

```
Height in feet and inches: 5 7
You are medium.

Height in feet and inches: 6 1
You are tall.

Person #2 is taller than person #1.
```

Cumulative algorithms

reading: 4.2

Adding many numbers

- How would you find the sum of all integers from 1-1000?

```
// This may require a lot of typing
int sum = 1 + 2 + 3 + 4 + ... + 999 + 1000;
System.out.println("The sum is " + sum);
```

- What if we want the sum from 1 - 1,000,000?
  Or the sum up to any maximum?
  - How can we generalize the above code?

A failed attempt

- An incorrect solution for summing 1-1000:

```
for (int i = 1; i <= 1000; i++) {
    int sum = 0;
    sum = sum + i;
}
// error: sum is undefined here
System.out.println("The sum is " + sum);
```

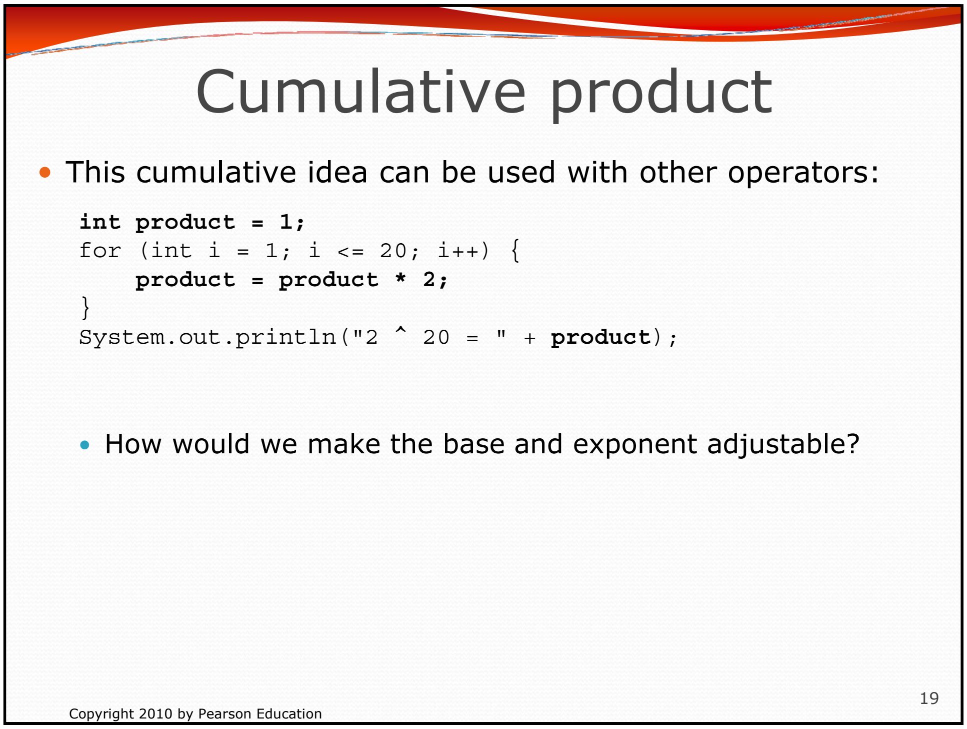
  - `sum`'s scope is in the `for` loop, so the code does not compile.

- **cumulative sum**: A variable that keeps a sum in progress and is updated repeatedly until summing is finished.
  - The `sum` above is an incorrect attempt at a cumulative sum.

Corrected cumulative sum

```
int sum = 0;
for (int i = 1; i <= 1000; i++) {
    sum = sum + i;
}
System.out.println("The sum is " + sum);
```

- Cumulative sum variables must be declared *outside* the loops that update them, so that they will still exist after the loop.

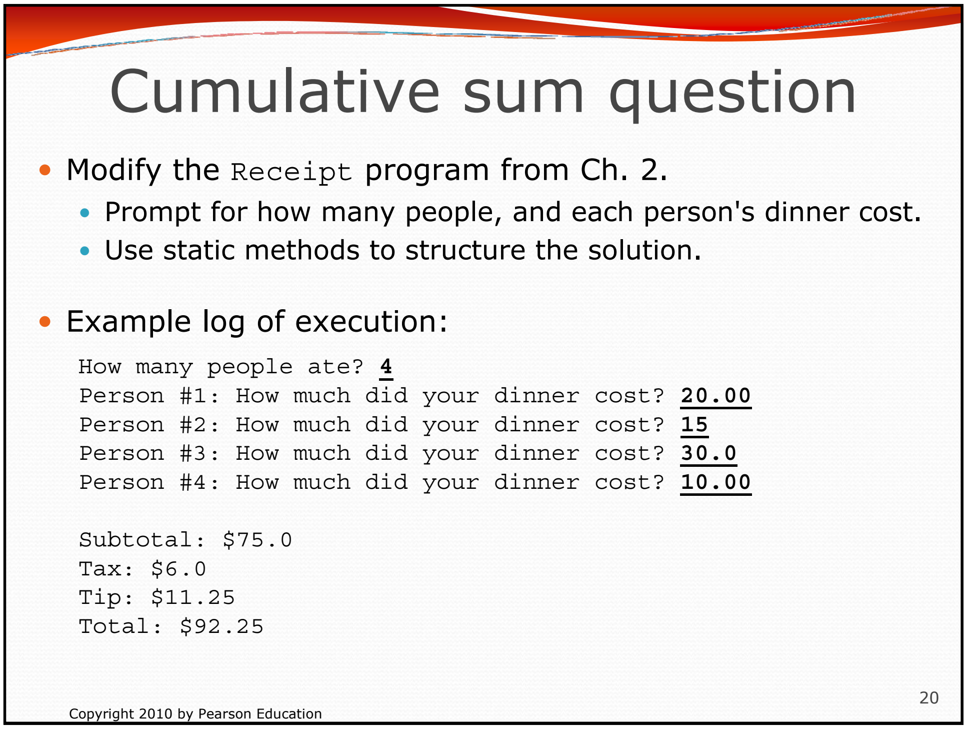


Cumulative product

- This cumulative idea can be used with other operators:

```
int product = 1;
for (int i = 1; i <= 20; i++) {
    product = product * 2;
}
System.out.println("2 ^ 20 = " + product);
```

  - How would we make the base and exponent adjustable?

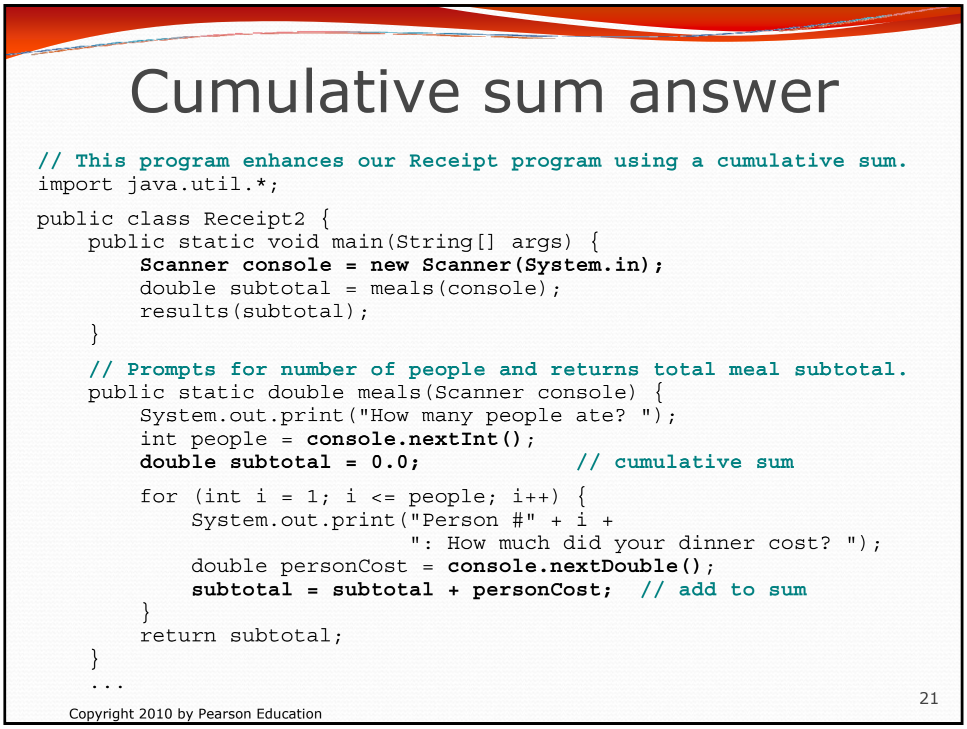


Cumulative sum question

- Modify the `Receipt` program from Ch. 2.
  - Prompt for how many people, and each person's dinner cost.
  - Use static methods to structure the solution.

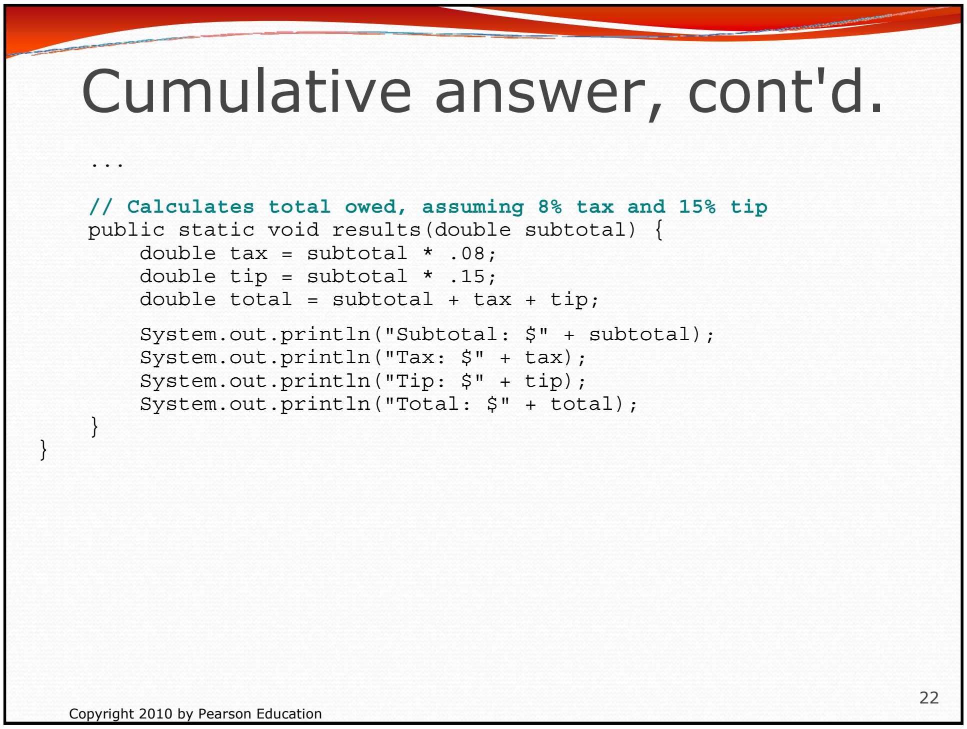
- Example log of execution:

```
How many people ate? 4
Person #1: How much did your dinner cost? 20.00
Person #2: How much did your dinner cost? 15
Person #3: How much did your dinner cost? 30.0
Person #4: How much did your dinner cost? 10.00

Subtotal: $75.0
Tax: $6.0
Tip: $11.25
Total: $92.25
```



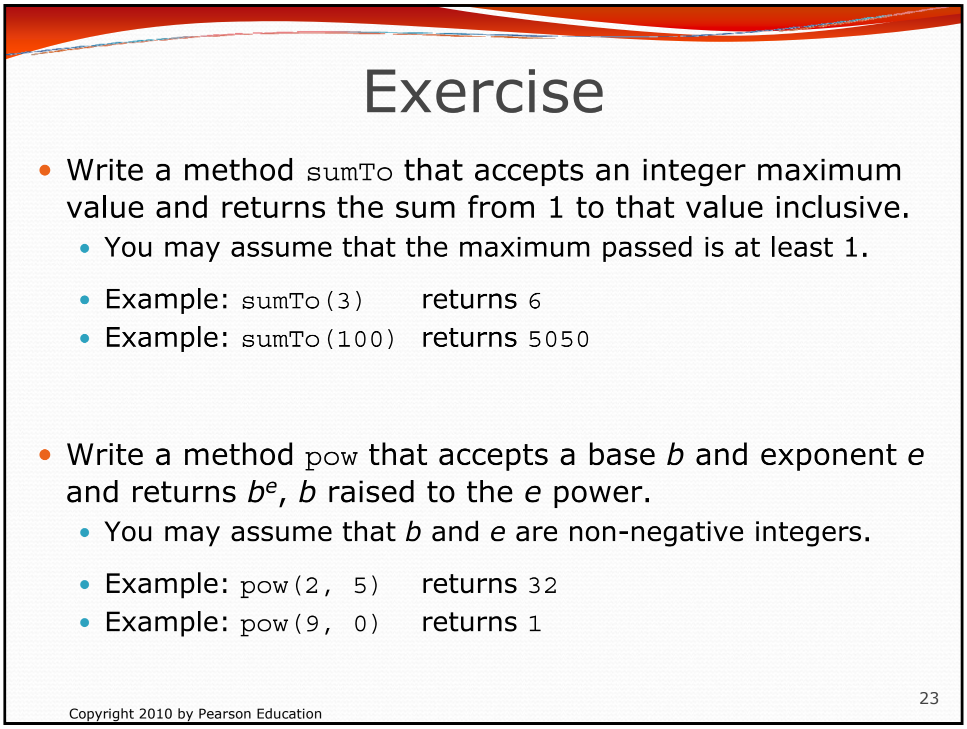
Cumulative sum answer

```
// This program enhances our Receipt program using a cumulative sum.
import java.util.*;

public class Receipt2 {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        double subtotal = meals(console);
        results(subtotal);
    }

    // Prompts for number of people and returns total meal subtotal.
    public static double meals(Scanner console) {
        System.out.print("How many people ate? ");
        int people = console.nextInt();
        double subtotal = 0.0;              // cumulative sum

        for (int i = 1; i <= people; i++) {
            System.out.print("Person #" + i +
                             ": How much did your dinner cost? ");
            double personCost = console.nextDouble();
            subtotal = subtotal + personCost;   // add to sum
        }
        return subtotal;
    }
    ...
```



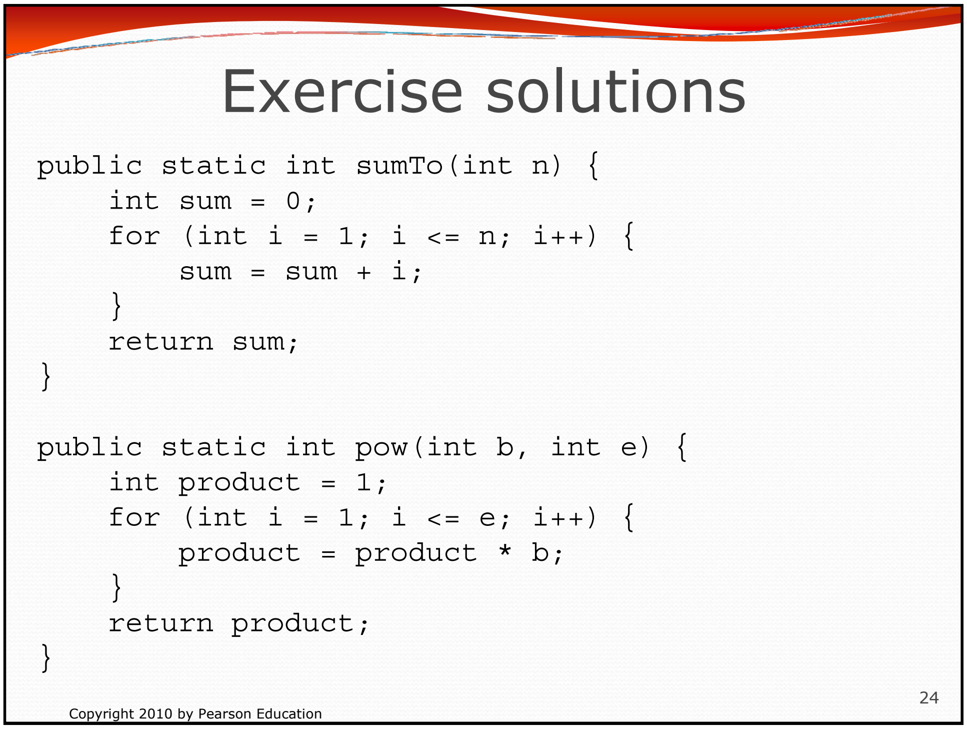
Cumulative answer, cont'd.

```
    ...

    // Calculates total owed, assuming 8% tax and 15% tip
    public static void results(double subtotal) {
        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);
    }
}
```



Exercise

- Write a method `sumTo` that accepts an integer maximum value and returns the sum from 1 to that value inclusive.
  - You may assume that the maximum passed is at least 1.
  - Example: `sumTo(3)` returns `6`
  - Example: `sumTo(100)` returns `5050`
- Write a method `pow` that accepts a base *b* and exponent *e* and returns b^e, *b* raised to the *e* power.
  - You may assume that *b* and *e* are non-negative integers.
  - Example: `pow(2, 5)` returns `32`
  - Example: `pow(9, 0)` returns `1`



Exercise solutions

```
public static int sumTo(int n) {
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        sum = sum + i;
    }
    return sum;
}

public static int pow(int b, int e) {
    int product = 1;
    for (int i = 1; i <= e; i++) {
        product = product * b;
    }
    return product;
}
```