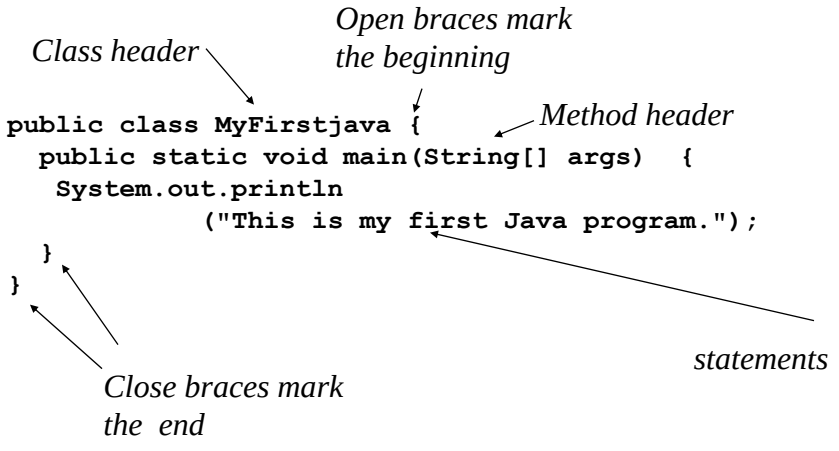
CSC 175 - Intermediate Programming

Lecture 1 - An Introduction to Programming in Java

A First Program



A First Program – What Does It Do?

```
System.out.println
        ("This is my first Java program.");
```

Prints the message

```
This is my first Java program.
```

Ends the line

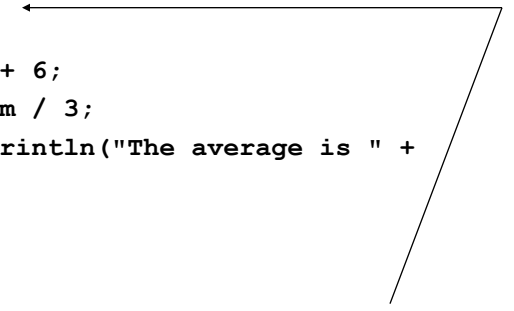
Writing Our Second Program

```
public class Average3  {
  public static void main(String[] args)  {
    int sum, average;
    sum = 2 + 4 + 6;
    average = sum / 3;
    System.out.println("The average is " +
    average);
  }
}
```

Tells the computer that **`sum`** *and* **`average`** *are* ***integers***

Writing Our Second Program

```
public class Average3a  {
  public static void main(String[] args)  {
    int sum;
    int average;
    sum = 2 + 4 + 6;
    average = sum / 3;
    System.out.println("The average is " +
    average);
  }
}
```



We could also write this as two **separate** *declarations.*

Variables and Identifiers

- Variables have names – we call these names ***identifiers***.
- Identifiers identify various elements of a program (so far the only such element are the variables.
- Some identifiers are standard (such as **`System`**)

Identifier Rules

- An identifier must begin with a letter or an underscore _
- Java is case sensitive upper case (capital) or lower case letters are considered different characters. `Average`, `average` and `AVERAGE` are three different identifiers.
- Numbers can also appear after the first character.
- Identifiers can be as long as you want but names that are too long usually are too cumbersome.
- Identifiers cannot be reserved words (special words like `int`, `main`, etc.)

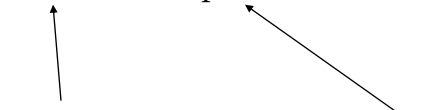
Some Illegal Identifiers

Illegal Identifier	Reason	Suggested Identifier
`my age`	Blanks are not allowed	`myAge`
`2times`	Cannot begin with a number	`times2` or `twoTimes`
`four*five`	* is not allowed	`fourTimesFive`
`time&ahalf`	& is not allowed	`timeAndAHalf`

Assignment Statements

- Assignment statements take the form:

variable = *expression*



Memory location where the value is stored

Combination of constants and variables

Expressions

- Expressions combine values using one of several ***operations***.
- The operations being used is indicated by the ***operator***:

+	Addition
-	Subtraction
*	Multiplication
/	Division

Expressions – Some Examples

2 + 5

4 * value

x / y

Another Version of Average

- Let's rewrite the average program so it can find the average any 3 numbers we try:
- We now need to:
  1. Find our three values
  2. Add the values
  3. Divide the sum by 3
  4. Print the result

The **`Scanner`** Class

- Most programs will need some form of input.
- At the beginning, all of our input will come from the keyboard.
- To read in a value, we need to use an object belonging to a class called Scanner:

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

Reading from the keyboard

- Once we declare keyb as Scanner, we can read integer values by writing:

variable `= keyb.nextInt();`

```
import  java.util.Scanner;

public class Average3b  {
  public static void main(String[] args)  {
    int sum, average;
    Scanner  keyb = new Scanner(System.in);

    System.out.println
                 ("What is the first value\t?");
    int  value1 = keyb.nextInt();

    System.out.println
              ("What is the second value\t?");
    int  value2 = keyb.nextInt();









    System.out.println
               ("What is the third value\t?");
    int  value3 = keyb.nextInt();

    sum = value1 + value2 + value3;
    average = sum / 3;
    System.out.println("The average is "
                            + average);
  }
}
```

Another example – calculating a payroll

- We are going to write a program which calculates the gross pay for someone earning an hourly wage.
- We need two pieces of information:
  - the hourly rate of pay
  - the number of hours worked.
- We are expected to produce one output: the gross pay, which we can find by calculating:
  - Gross pay = Rate of pay * Hours Worked

```
import java.util.Scanner;

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

    System.out.println
               ("What is your hourly pay rate?");
    double rate = keyb.nextDouble();

    System.out.println
              ("How many hours did you work?");
    double hours = keyb.nextDouble();

    double gross = rate * hours;
    System.out.println("Your gross pay is $"
                       + gross);
  }
}
```

Comments

- Our program is a bit longer than our previous programs and if we did not know how to calculate gross pay, we might not be able to determine this from the program alone.
- It is helpful as programs get much longer to be able to insert text that explains how the program works. These are called ***comments***. Comments are meant for the human reader, not for the computer.
- In Java, anything on a line after a double slash (**//**) is considered a comment.
- Longer comments can also be contained between **/*** and ***/**

```
import java.util.Scanner;

public class Payroll  {

  // This program calculates the gross pay for an
  // hourly worker
  // Inputs - hourly rate and hours worked
  // Output - Gross pay
  public static void main(String[] args)  {
    Scanner keyb = new Scanner(System.in);

    // Get the hourly rate
    System.out.println
            ("What is your hourly pay rate?");
    double rate = keyb.nextDouble();
```

```
    // Get the hours worked
    System.out.println
          ("How many hours did you work?");
    double hours = keyb.nextDouble();

    // Calculate and display the gross pay
    double gross = rate * hours;
    System.out.println("Your gross pay is $"
                       + gross);
  }
}
```

Character Data

- All of our programs so far have used variables to store numbers, not words.
- We can store single characters by writing:

  `char x, y;`

  – `x` and `y` can hold one and only one character
- For now, we use character data for input and output only.

Character Strings

- We are usually interested in manipulating more than one character at a time.
- We can store more than one character by writing:

```
String s = new String();
```

- If we want `s` can hold to have some initial value, we can write:

```
String s
        = new String("Initial value");
```

- For now, we use character data for input and output only.

A program that uses a character variable

```
import java.util.Scanner;

public class Polite  {
  // A very polite program that greets you by name
  public static void main(String[] args) {
    String name = new String();
    Scanner keyb = new Scanner(System.in);
    // Ask the user his/her name
    System.out.println("What is your name?");
    name = keyb.next();
    // Greet the user
    System.out.println("Glad to meet you, " + name);
  }
}
```

if and *if-else*

- Some problems may have a set of instructions that are only performed under some conditions. These require an **`if`** construct.
- Other problems may have two or more alternative sets of instructions depending on some condition(s). If there are two alternatives, it requires an if-else construct.

if and *if-else (continued)*

- The general form is:

```
if (expression)
  statement;
```

or

```
if (expression)
  statement;
else
  statement;
```

Example – Is It Negative?

- Example – Write a program that determine if a number is negative or non-negative
- Our ***algorithm*** (recipe for a program):
  - Get the number
  - Print whether its negative or non-negative

IsItNegative.java

```
import java.util.Scanner;

public class IsItNegative  {
  // Tell a user if a number is negative or
  // non-negative
  public static void main(String[] args)  {
    Scanner keyb = new Scanner(System.in);

    // Ask the user for a number
    System.out.println
               ("Please enter a number?");
    double number = keyb.nextDouble();
```

```
    // Print whether the number is negative or
    // not
    if (number < 0.0)
      System.out.println(number
                  + " is a negative number");
    else
      System.out.println(number
                + " is NOT a negative number");
  }
}
```

Relational operators

Operator	**Meaning**	**Example**
==	equals	`x == y`
!=	is not equal to	`1 != 0`
>	greater than	`x+1 > y`
<	less than	`x-1 < 2*x`
>=	greater than or equal to	`x+1 >= 0`
<=	less than or equal to	`-x +7 <= 10`

Example – Calculating Speed

- *Example* - Calculate the speed that you are driving from the distance and time that you have been driving. If you are going over the speed limit, print a warning message.
- We know the following about our problem:

  Available input:
  - Distance in miles
  - Time in hours

  Required output:
  - Speed in miles per hour
  - Warning message (if appropriate)

The Complete **`Speed`** Program

```
import java.util.Scanner;

public class Speed  {
  // Calculate the speed that you are traveling
  // from the distance and time that you have
  // been driving.
  // Print a warning if you are going over the
  // speed limit.
  public static void main(String[] args)  {
    Scanner keyb = new Scanner(System.in);

    // Read in the distance in miles and
    // time driven
    System.out.println
              ("How many miles have you driven?");
    double miles = keyb.nextDouble();
```

```
    System.out.println
              ("How many hours did it take?");
    double hours = keyb.nextDouble();

    // Calculate and print the speed
    double speed = miles / hours;
    System.out.println("You were driving at "
                  + speed + " miles per hour.");

    // Print the warning if appropriate
    if (speed > 55)
       System.out.println("**BE CAREFUL!**"
                  + "You are driving too fast!");
  }
}
```

Constants

- Let's re-examine the statement in our program **`ConvertPounds2`** that does the actual conversion:

  `kg = lbs / 2.2;`
- Where does come 2.2 from? (There are 2.2 pounds per kilogram)
- How would know why we use 2.2 if we are not familiar with the problem?

ConvertPounds

```
import java.util.Scanner;

public class ConvertPounds  {

  // Convert pounds to kilograms
  // Input - weight in pounds
  // Output - weight in kilograms
  public static void main(String[] args)  {
    Scanner keyb = new Scanner(System.in);
    final double lbsPerKg = 2.2;

    // Get the weight in pounds
    System.out.println
              ("What is the weight in pounds?");
    double lbs = keyb.nextDouble();









    // Ensure that the weight in pounds is
    // valid. If it is valid, calculate and
    // display the weight in kilograms
    if (lbs < 0)
      System.out.println(lbs
                  + " is not a valid weight.");
    else  {
      double kg = lbs / lbsPerKg;
      System.out.println("The weight is "
                          + kg + " kilograms");
    }
  }
}
```

Declaring Constants

•The general form of the constant declaration is:

```
final datatype ConstantName =
                    ConstantValue,
     AnotherConstantName =
                AnotherConstantValue;
```

•Let's take a look at a few examples of constants:

```
final double withholdingRate = 0.8;
final char  prompt = 'y';
final String     answer = "yes";
final int  maxPeople = 15,
    inchPerFt = 12;
final int speedLimit = 55;
```

Compound Decisions

- Being able to do more than one statement is helpful:

```
if (lbs < 0)
      System.out.println(lbs
            + " is not a valid weight.");
else  {
      kg = lbs / lbsperkg;
      System.out.println("The weight is "
            + kg + " kilograms");
}
```

Blocks

- Any place in a Java where a statement can appear, a block can also appear.
- A block is a set of statements between opening and closing braces (***{ }***).
- Example:

```
if (x > y)   {
  System.out.println("x is larger");
  max = x;
}
```

An Auto Insurance Program

- Example - Write a program to determine the cost of an automobile insurance premium, based on driver's age and the number of accidents that the driver has had.
- The basic insurance charge is $500. There is a surcharge of $100 if the driver is under 25 and an additional surcharge for accidents:

# of accidents	Accident Surcharge
1	50
2	125
3	225
4	375
5	575
6 or more	No insurance

An Auto Insurance Program (continued)

- Available input
  - Number of accidents
  - driver age
- Required output
  - Insurance charge.

The Final Insurance Program

```
import  java.util.Scanner;

public class CarInsurance  {
  // A program to calculate insurance premiums
  // based on the driver's age and accident
  // record.
  public static void main(String[] args)  {
    Scanner keyb = new Scanner(System.in);
    final double basicRate = 500;
    double rate;
    int  age, numAccidents;
    int  ageSurcharge = 0,
         accidentSurcharge = 0;
    boolean error = false, tooMany = false;
```

```
// Input driver's age and number of
// accidents
System.out.println
                ("How old is the driver?");
age = keyb.nextInt();

System.out.println("How many accidents has "
              + "the driver had?");
numAccidents = keyb.nextInt();

// Determine if there is an age surcharge
if (age < 0)
  error = true;
else if (age < 25)
  ageSurcharge = 100;
else
  ageSurcharge = 0;
```

```
// Determine if there is a surcharge
if (numAccidents < 0)
  error = true;
else if (numAccidents == 0)
  accidentSurcharge = 0;
else if (numAccidents == 1)
  accidentSurcharge = 50;
else if (numAccidents == 2)
  accidentSurcharge = 125;
else if (numAccidents == 3)
  accidentSurcharge = 225;
else if (numAccidents == 4)
  accidentSurcharge = 375;
else if (numAccidents == 5)
  accidentSurcharge = 575;
else
  tooMany = true;
```

```
    // Print the charges
    if (error)
      System.out.println("There has been an "
                 + " error in the data that "
                 + " you supplied");
    else if (tooMany)
      System.out.println("You have had too "
                  + "many accidents for me to "
                  + " insure you.");
    else {
      System.out.println("The basic rate is $"
                         + basicRate);
      if (ageSurcharge > 0)
        System.out.println("There is an extra "
                   + "surcharge of $"
                   + ageSurcharge
                   + " because the driver is"
                   + " under 25.");









      if (accidentSurcharge > 0)
        System.out.println("There is an extra "
                    + " surcharge of $"
                    + accidentSurcharge
                    + " because the driver had "
                    + numAccidents
                    + " accident(s).");

      rate = basicRate + ageSurcharge
                       + accidentSurcharge;
      System.out.println("The total charge is $"
                         + rate);
    }
  }
}
```

Loops

- We need the ability to perform the same set of instructions repeatedly so we don't have to write them over and over again.
- This is why Java includes several ways of using repetition in a program.
- Each case where we repeat a set of statement is called a ***loop***.

Counting Loops

- The first type of loop is a ***counting loop***.
- Counting loops are repeated a specific number of times.
- If you read the loop, you can easily figure out how many times its statements will be performed.

Example: Hello Again

- Example - Write a program that greets the user with "Hello, again!" five times.
- We could write the program like this:

```
import java.util.Scanner;

public class HelloAgain  {
  // Hello again - this program writes "Hello,
  // again" five times
  public static void main(String[] args)  {
    System.out.println("Hello, again");
    System.out.println("Hello, again");
    System.out.println("Hello, again");
    System.out.println("Hello, again");
    System.out.println("Hello, again");
  }
}
```

Counting Loops

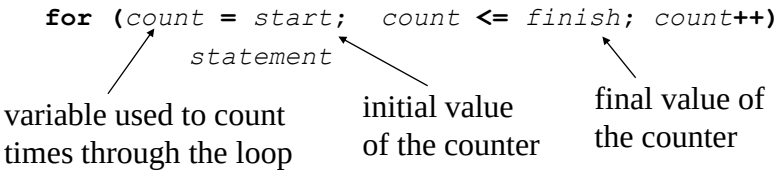
- We use a for loop to write counting loops
- In Java, it looks like this:

```
for (count = start;  count <= finish; count++)
        statement;
```

- or

```
for (count = start;  count <= finish; count++)  {
        statements
}
```

Counting Loops (continued)



The New *HelloAgain*

```
public class HelloAgain2  {
  // HelloAgain2 - this is a better way to write
  //               "Hello, again" five times
  public static void main(String[] args)  {
    int i;
    for (i = 1;  i <= 5;  i++)
      System.out.println("Hello, again");
  }
}
```

Generalizing *HelloAgain*

- This program is also flawed; it gives us no choices as to how many times we can print “Hi, there!”
- We can to let the user select how many times to print the message and making this version of the program more general is fairly easy:
- Our algorithm will start as:
  1. Find out how many time to print the message.
  2. Print "Hi, there!" that many times.

The Revised *HelloAgain*

```
import java.util.Scanner;

public class HelloAgain3  {
  // HelloAgain3 - Write "Hello, again" as many times
  //               as the user wants
  public static void main(String[] args)  {
    Scanner keyb = new Scanner(System.in);
    int i, count, totalTimes;

    System.out.println("How many times do you want to "
                       + "say \"hello\"?");
    totalTimes = keyb.nextInt();
    for (count = 0;  count < totalTimes;  count++)
      System.out.println("Hello, again");
  }
}
```

Example: Averaging ***n*** Numbers

- Let's get back to our original problem. We want to able to average any number of values.
- Let's start by outlining our algorithm:
  1. Find out how many values there are.
  2. Add up all the values.
  3. Divide by the number of values
  4. Print the result

The ***AverageN*** Program

```
import java.util.Scanner;

public class AverageN  {
  //AverageN - Find the average of N values
  public static void main(String[] args)  {
    Scanner keyb = new Scanner(System.in);
    double sum, average, value;
    int    numValues, currentValue;

    //Find out how many values there are
    System.out.println
        ("How many values are you going to enter?");
    numValues = keyb.nextInt();
```

```
    // Read in each value and add it to the sum
    sum = 0.0;
    for  (currentValue = 1;
          currentValue <= numValues;
          currentValue++)            {
      System.out.println("What is the next value?");
      value = keyb.nextDouble();
      sum = sum + value;
    }

    // Calculate and print out the average
    average = sum / numValues;
    System.out.println("The average is " + average);
  }
}
```

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

The Interest Program

```
public class Interest  {
  // Calculate the interest that the Canarsie
  // Indians could have accrued if they had
  // deposited the $24 in a bank account at
  // 5% interest.
  public static void main(String[] args)  {
    final int present = 2005;
    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
                + "\tprinciple = "
                +  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 = 2001 principle = 2.3365602874289446E9
year = 2002 principle = 2.4533883018003917E9
year = 2003 principle = 2.5760577168904114E9
year = 2004 principle = 2.704860602734932E9
```

•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:

  `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:

```
System.out.printf("This is a test");
System.out.printf("This is a test").
```

will produce:

```
This is a testThis is a test
```

If you want these to be on two separate lines:

```
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*`.`*d*`f` will use at least *w* spaces to display the value and will have exactly *d* decimal places.

Changing the width

Number	Formatting	Print as:
182	`%2d`	182
182	`%3d`	182
182	`%5d`	``182
182	`%7d`	````182
-182	`%4d`	-182
-182	`%5d`	`-182
-182	`%7d`	```-182

Changing the width (continued)

Number	Formatting	Print as:
23	%1d	23
23	%2d	23
23	%6d	….23
23	%8d	……23
11023	%4d	11023
11023	%6d	.11023
-11023	%6d	-11023
-11023	%10d	…..11023

Changing The Precision

Number	Formatting	Prints as:
2.718281828	%8.5f	`2.71828
2.718281828	%8.3f	```2.718
2.718281828	%8.2f	````2.72
2.718281828	%8.0f	````````3
2.718281828	%13.11f	2.71828182800
2.718281828	%13.12f	2.718281828000

The revised *Compound* program

```
public class Interest2  {
  // 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 = 2005;
    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\tprinciple = $%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
```

Integer Division

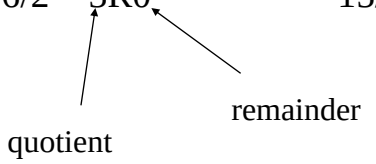
- Our compound interest program prints the values for every year where every ten or twenty years would be good enough.
- What we really want to print the results only if the year is ends in a 5. (The remainder from division by 10 is 5).

Integer Division (continued)

- Division of an integer by an integer produces an integer quotient:

5/3 = 1R2 16/3 = 5R1

6/2 = 3R0 15/4 = 3R3



Integer Division (continued)

- In Java, the / operator produces n integer quotient for integer division.
- If you want the remainder from integer division, you want to use the **%** operator

```
public class DivTest  {
  public static void main(String[] args)  {
    // A few examples of integer division using
    // / and %
    System.out.println("8 / 3 = " + 8 / 3 );
    System.out.println("8 % 3 = " + 8 % 3 );

    System.out.println("2 / 3 = " + 2 / 3 );
    System.out.println("2 % 3 = " + 2 % 3 );

    System.out.println("49 / 3 = " + 49 / 3 );
    System.out.println("49 % 3 = " + 49 % 3 );

    System.out.println("49 / 7 = " + 49 / 7 );
    System.out.println("49 % 7 = " + 49 % 7 );

    System.out.println("-8 / 3 = " + -8 / 3 );
    System.out.println("-8 % 3 = " + -8 % 3 );

    System.out.println("-2 / 3 = " + -2 / 3 );
    System.out.println("-2 % 3 = " + -2 % 3 );

    System.out.println("-2 / -3 = " + -2 / -3 );
    System.out.println("-2 % -3 = " + -2 % -3 );

    System.out.println("2 / -3 = " + 2 / -3 );
    System.out.println("2 % -3 = " + 2 % -3 );

    System.out.println("-49 / 3 = " + -49 / 3 );
    System.out.println("-49 % 3 = " + -49 % 3 );

    System.out.println("-49 / -3 = " + -49 / -3 );
    System.out.println("-49 % -3 = " + -49 % -3 );

    System.out.println("49 / -3 = " + 49 / -3 );
    System.out.println("49 % -3 = " + 49 % -3 );
```

```
    System.out.println("-49 / 7 = " + -49 / 7 );
    System.out.println("-49 % 7 = " + -49 % 7 );

    System.out.println("-49 / -7 = " + -49 / -7 );
    System.out.println("-49 % -7 = " + -49 % -7 );

    System.out.println("49 / -7 = " + 49 / -7 );
    System.out.println("49 % -7 = " + 49 % -7 );
  }
}
```

Integer Division Results

8 / 3 = 2	8 % 3 = 2
2 / 3 = 0	2 % 3 = 2
49 / 3 = 16	49 % 3 = 1
49 / 7 = 7	49 % 7 = 0
-8 / 3 = -2	-8 % 3 = -2
-2 / 3 = 0	-2 % 3 = -2
-2 / -3 = 0	-2 % -3 = -2
2 / -3 = 0	2 %-3 = 2
-49 / 3 = -16	-49 % 3 = -1

Integer Division Results (continued)

-49 / -3 = 16	**-49 % -3 = -1**
49 / -3 = -16	**49 % -3 = 1**
-49 / 7 = -7	**-49 % 7 = 0**

Final Compound Interest Program

```
public class Interest3  {
  // 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 = 2005;
    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;

      // Print the principle for every 20th year
      if (year % 20 == 5)
        System.out.printf
          ("year = %4d\tprinciple = $%13.2f\n",
                        year, principle);
    }
    // Print te values for the last year
    System.out.printf
         ("year = %4d\tprinciple = $%13.2f\n",
                        year, principle);
  }
}
```

A program to calculate Grade Point Average

Example – Professor Smith gives n tests during the term and uses a grading system, where each test is 1/n of the course grade. Assuming that that the average of the test grades translate into a letter grade as follows:

Test Average	Letter Grade
90.0+	A
80-89.9	B
70-79.9	C
60-69.9	D
below 60.0	F

write a program that will calculate a student's grade.

A Program To Calculate Test Average

Input - Number of tests and the student's test grades
Output – Test average and course grade
Other information
A 90+ average is an “A”.
A 80-90 average is a “B”.
A 70-80 average is a “C”.
A 60-70 average is a “D”
An average below 60 is an “F”.

Test average = Sum of the test grade/ Number of tests

Our first step is to write out our initial algorithm:

1. Find the number of tests
2. Find the average of *n* tests
3. Find the corresponding letter grade and print it out.

Our program

```
public static void main(String[] args)  {
  Scanner keyb = new Scanner(System.in);
  int thisTest, numTests, total, thisGrade;
  float testAverage;
  char  courseGrade;

  // Find out the number of classes
  System.out.println
            ("How many tests did you take ?");
  numTests = keyb.nextInt();

  for (thisTest = 0;  thisTest < numTests;
                                    thisTest++) {
      System.out.println
       ("What grade did you get on this test  ?");
      thisGrade = keyb.nextInt();
```

```
    // Make sure that the grades are valid
    // percentages
    total = total + thisGrade;
  }
  // Find the average
  testAverage = total/numTests;

  // Find the letter grade corresponding to the
  // average
  if (testAverage >= 90)
    courseGrade = 'A';
  else if (testAverage >= 80)
    courseGrade = 'B';
  else if (testAverage >= 70)
    courseGrade = 'C';
  else if (testAverage >= 60)
    courseGrade = 'D';
  else
    courseGrade = 'F';









    // Print the results.
    System.out.println
         ("Your test average is " + testAverage);
    System.out.println
           ("Your grade will be " + courseGrade);
  }
}
```