HOMEWORK PROBLEMS 1

1) Invent a mnemonic (i.e., something that aids memory) so that you can easily remember the header required for the `main` method (see Fig. 1.3).

2) Which of the following identifiers are the names of Java classes: `main`, `Random`, `public`, `String`, `int`, and `Integer`.

3) What are the binary equivalents for the decimal numbers 0 through 15?

4) What is wrong with the following command,

```
javac Program1
```

and the following Java statement:

```
System.out.println[2 + 2]
```

5) Write a single statement that creates a `String` reference variable `s`, creates a `String` object that contains the string "`dog`", and assigns `s` the reference to the object created.

6) Compile and run the program in Fig. 1.3 by entering

```
javac Program1.java
java Program1
```

Now try running the program again by entering (with a lowercase `p`)

```
java program1
```

What happens?

7) Compile and run the program in `C1h7.java`. Verify that the program compiles and runs correctly. Change `args` to `goofy`. Verify that the program still compiles and runs correctly. Java does not require the name `args` for the name of the parameter in the `main` method, although everybody uses it.

8) Compile and run the program in `C1h8.java` in the software package. Verify that the program compiles and runs correctly. Remove the semicolon at the end of the first `println` statement. Compile the program and inspect the compile-time error that results. Does the error message correctly indicate the location of the error in the source program?

9) What's wrong with the following program:

```
Class C1h9
{
    public static void Main(String args)
    {
        system.out.println("C1h9");
     }
}
```

Compile and run it to check your answer. Fix the program so that it runs correctly.

10) Compile the following program to verify it has four syntax errors. For each error, what is the error message produced by the compiler?

```
clas s C1h10
{
    public static void main(String[] args)
    {
       System.out.println("Good
       bye");
       System.out.println("Go od by e");
       System.out.println("all done);"
    )
}
```

Fix the program so it runs correctly.

11) Verify that the following program runs correctly:

```
class
 C1h11
 { public static
   void main(String[] args) {System.out.println("hello");
 }}
```

Reformat the program so that it is easier to read. You should never write a program with sloppy formatting because it makes the program difficult to read. Always format your programs the way the programs in this textbook (except for this one) are formatted. When you indent, always indent the same number of columns (three columns is a good choice) from the start of the previous line. Don't indent fewer than three columns (it will make it difficult for your eye to pick up the indentation). *Indentation is essential! Always indent properly*.

12) Does the program below work? Note that `public` follows `static` in the method header.

```
class C1h12
{
   static public void main(String[] args)
   {
      System.out.println("hello");
   }
}
```

Now switch the positions of `void` and `public`. Does the program still work? The return type of a method must immediately precede the method name. Thus, `void` (which is the return type of `main`) must immediately precede `main` and follow `public` and `static`. `public` can either precede or follow `static`, although it is customary to put `public` first.

13) Insert the statement below in an otherwise correct program. What happens when you run the program?

```
System.out.println(5/0);
```

14) Write a program that displays your name. Use a file named `C1h14.java`. Use the class name `C1h14`. Compile and run.

15) Compile and run the following program:

```
1 // escape sequences example
2 class C1h15
3 {
4   public static void main(String[] args)
5   {
6      System.out.println("hel\"lo"); // escape sequence
7   }
8 }
```

The initial quote on line 6 is a *special* quote. It is special in the sense that it starts the string constant. The third quote on line 6 is also a *special* quote. It is special in the sense that it ends the string constant. The middle quote is an *ordinary* quote in the sense it does not act in a special way (it does not end the string constant). This quote is made ordinary by the preceding backslash character. Rule: *a backslash that precedes a character that is normally special makes that character ordinary*. We call sequences that start with a backslash **escape sequences**. When the program above runs, it displays:

```
hel"lo
```

The backslash-quote sequence is displayed as a single quote. Now delete the backslash character in the `println` statement on line 6 to get

```
System.out.println("hel"lo"); // escape sequence
```

Because the middle quote is now not backslashed, it is a special quote. That is, it ends the string constant, making the characters to its right look like garbage. Verify that the program no longer compiles correctly.

Like the quote, a backslash can be either a special character (if is not preceded by a special backslash) or an ordinary character (if is preceded by a special backslash). Change line 6 to

```
System.out.println("hel\\\"lo"); // escape sequence
```

Compile and run. We now have two consecutive escape sequences: `\\` followed by `\"`. The first backslash makes the second backslash an ordinary character. The third backslash makes the middle quote an ordinary character. Note that in sequence

```
\\\"
```

the third backslash is not an ordinary backslash even though it is preceded by a backslash because the middle backslash is an ordinary backslash by virtual of the first backslash. Thus, this sequence represents an ordinary backslash followed by an ordinary quote. This `println` statement displays the following seven characters:

```
hel\"lo
```

16) Write a program that displays the following sequence of eight characters (see homework problem 15):

```
/\/\"/\"
```

17) Can a blank line be inserted anywhere in a Java program? Run a test program to determine the effect of blank lines.

18) Determine the effect of

```
System.out.print("hello");
System.out.print("hello");
```

How does it differ from

```
System.out.println("hello");
System.out.println("hello");
```

and from

```
System.out.println("hello"); System.out.println("hello");
```

Hint: The "`ln`" in `println` means "go to the beginning of the next line."

19) Run a test program to determine the effect, if any, of

```
System.out.println();
```

20) Is the following a statement legal:

```
System.    out  .   println("hello");
```

Compile a program that contains it to check your answer.

21) Write a program that computes and displays the sum of 1, 2, 3, 4, and 5.

22) What does the following statement display:

```
System.out.println(1 + 2 + "3");
```

Why does it display what it does? *Hint*: The plus operations are performed left to right. The plus sign means concatenation (i.e., joining a string and something else together to get a longer string) if it is operating on a string.

23) Write a program that displays

```
H
 e
  l
   l
    o
```

24) Write a program that displays your name in a box, like so:

```
+-------------------------+
|                         |
|      your name          |
|                         |
+-------------------------+
```

25) Write a program that displays a plus sign formed with asterisks. The vertical and horizontal strokes of the plus sign should each be formed with 11 asterisks.

26) Write a program that displays the following sequence of characters (see homework problem 15):

```
\""\\"""\\\"""\\\\
```

27) Write a program that *computes* and *displays* the sum of 5 and -20. Your program should produce a display that looks like this:

```
5 + -20 = -15
```

Hint: use a `print` statement to display "`5 + -20 =  `" (see homework problem 18). Then use a `println` statement to compute the sum of 5 and -20 and display the result.

28) What is displayed when the following program is run?

```
class C1h28
{
   public static void main(String[] args)
   {
      System.out.println("hello");
      System.out.print("goodluck");
      System.out.print("havefun");
      System.out.println();
      System.out.println("alldone");
   }
```

29) Decimal, Binary, and Hexadecimal

Decimal is a positional number system. It is so called because in a decimal number the contribution of each digit to the value of the number depends not only on the digit but its position in the number. For example, consider the three-digit decimal number 157:

1	5	7	
100	10	1	weights

Each position has a weight. In a whole number, weights start with 1 and increase from right to left by a factor of 10 from each position to the next. The value of the number is given by the sum of each digit times its weight. Thus, the value of 157 is

$1\times100 + 5\times10 + 7\times1$

The 1 digit contributes $1\times100 = 100$ to the value of the number; the 5 digit contributes $5\times10 = 50$ to the value of the number, the 7 digit contributes $7\times1 = 7$ to the value of the number. Although the 7 digit is greater than the 5 digit, the 5 digit contributes more to the value of the number than the 7 digit because its weight is ten times that of the 7 digit. We call decimal the *base-10* number system because it uses 10 distinct symbols and because weights increase by a factor of 10 from each position to the next.

Binary is the *base-2* number system. It uses two distinct symbols (0 and 1), called *bits*. Position weights increase by a factor of 2 from each position to the next. For example, consider the five-bit binary number 01011:

```
 0  1  0  1  1
16  8  4  2  1  weights
```

Its value is given by the sum of each bit times its weight:

$0\times16 + 1\times8 + 0\times4 + 1\times2 + 1\times1 = 11$ decimal

It is easy to determine the decimal value of a binary number: Simply add up the weights corresponding to the 1 bits. In the binary number above, the weights corresponding to 1 bits are 8, 2, and 1. Thus, the value of the number is 8 + 2 + 1 = 11 decimal.

We call a sequence of eight bits a *byte*. For example, 1111000010101010 consists of two bytes: 11110000 and 10101010. To *complement* a bit means to flip it. That is, change a 1 bit to 0 and a 0 bit to 1.

Hexadecimal (or hex for short) is the base-16 number system. It uses 16 symbols: 0 to 9 and a, b, c, d, e, and f or equivalently A, B, C, D, E, and F (the lowercase forms are more convenient for keyboard input because they do not require the shift key). The values of a, b, c, d, e, and f and their corresponding uppercase forms equal decimal 10, 11, 12, 13, 14, and 15, respectively.
Weights in a hexadecimal number increase by a factor of 16. For example, consider the three-digit hex number 2c5:

```
 2   c   5
256  16  1 weights (in decimal)
```

Its value is given by

$2\times256 + c\times16 + 5\times1$

The hex digit `c` is 12 in decimal so the expression above using only decimal is equal to

$2\times256 + 12\times16 + 5\times1 = 512 + 192 + 5 = 709$

Here are the decimal numbers from 0 to 15 and their binary and hex equivalents.

Decimal	Binary	Hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	a (or A)

```
11      1011      b  (or B)
12      1100      c  (or C)
13      1101      d  (or D)
14      1110      e  (or E)
15      1111      f  (or F)

=======================================================
```

The text file `hex.txt` that contains

```
A B
a b
01
```

On the command line (when positioned on the software package), enter:

`see hex.txt` (on Raspberry Pi, use `seerpi`; on Linux, `seelnx`)

The `see` program will display the `hex.txt` file (both the hex in each byte and the corresponding characters):

```
see Version 6.2             Sun Oct 22 14:07:21 2023

       0:  4120 420D 0A61 2062 0D0A 3031 0D0A        A B..a b..01..

Input file = hex.txt
List file  = hex.see
```

On the left is the contents of each byte in hex. On the right are the corresponding characters. A period is displayed for any code that does not correspond to a character. For example, the code `0D` and `0A` are for carriage return and newline, respectively, neither of which correspond to characters. Thus, on the right, `see` displays periods for them. The carriage return and newline mark the end of each line. It triggers a text editor displaying the file to continue the display at the beginning of the next line. *Note*: Non-Windows systems mark the end of each line in a text file with only the newline character (`0A` instead of `0D 0A`).

On the far left followed by a colon, it the address (in hex) relative to the beginning of the file of the start of each line. The first byte in the hex display on the left is `41`, the code for the letter `A`. The first byte in text display on the right is `A`, the character that corresponds to the code `41` hex.

What is `41` hex in decimal? What is the hex code for the letter `a`? What separates each line from the next (on Windows, it will be two bytes; on other systems, it will be only one byte)? What is the hex code for the digit `0`? What is the hex code for the letter `C`? What is the hex code for `%`. In binary, how do the codes for the letter `A` and the letter `a` differ? In binary, how do the codes for the letter `B` and the letter `b` differ? What must you add to the code for the

letter `A` to get the letter `a`? How to you get the code for the letter `Z` from the code for the letter `z`?

Using Microsoft Word or similar word processing program, create a file that contains

```
A B
a b
01
```

Save the file using the word processor's standard format. Using the `see` program, display the contents of this file. Is it different from the contents of the text file `hex.txt`?

Compile `Program1.java`, if not already compiled. Use a text editor, such as `notepad` or `nano`, to display the file `Program1.class`. Because `Program1.class` is not a text file (i.e., each byte does not contain the code for some character), a text editor displays gibberish. However, `see` displays in hex the contents of each byte. On the right is displays the character for any byte that has a code for a character, and a period otherwise.

Use `see` to display the contents of any large file. What happens? See `see.txt` for documentation on the `see` program.

30) What the decimal equivalents of the following hex numbers:

10, A0, 103

31) What the decimal equivalents of the following binary numbers:

011111, 010101, 010001

32) What the decimal equivalents of the following octal numbers (base 8):

123, 10, 70

33) What is the next number is the following series:

10, 11, 12, 13, 14, 15, 16, 17, 20, 22, 24, 31, 100, 121, ???

34) In decimal, what does the following base 1 number equal: 0101010101