HOMEWORK PROBLEMS 5

1) Is it legal to call

```
public static void rabbit(double x)
{
   System.out.println("hello");
}
```

with

```
rabbit(5);
```

Is it legal to call

```
public static void hare(int x)
{
   System.out.println("hello");
}
```

with

```
hare(5.0);
```

Run a test program to check your answers.

2) Write a program with a `main` method and a `max` method. `max` should return the largest of the three values it is passed. `main` should call `max` three times. On the first call, it should pass `max` 1, 2, and 3; on the second call, it should pass 2, 3, and 1; on the third call, it should pass 3, 1, and 2. For each call, `main` should display the value returned, appropriately labeled.

3) Can you call `goose` with

```
goose(5.0f);
```

if you have overloaded `goose` as follows:

```
public static void goose(int x)
{
   ...
}
public static void goose(double x)
{
   ...
}
```

Run a test program to check your answer. What happens?

The parameter `5.0f` is a `float` constant. Neither parameter in the two `goose` methods matches this type. When the compiler translates a method call, it generates code that calls the method whose parameter list is the best compatible match with the argument list. An exact match is not required.

4) Is this a legal overloading of the method name `g`:

```
public void pig()
{
   System.out.println("hello");
}
public int pig()
{
   return 5;
}
```

Run a test program to check your answer.

5) We call the operators `||` and `|` in the program below **logical operators** because they operate on true/false values and yield results that are true/false.

```
class C5h5
{
   public static void main(String[] args)
   {
      if (f() || g())                // short circuited OR
         System.out.println("first if");
      if (f() | g())                 // not short circuited
         System.out.println("second if");
   }
   public static boolean f()
   {
      System.out.println("in f");
      return true;
   }
   public static boolean g()
   {
      System.out.println("in g");
      return true;
   }
}
```

Java has two logical OR operators: `||` (with two vertical bars) and `|` (with one vertical bar). `||` is the **short-circuit** or **lazy** logical OR. That is, if the value of the operation can be determined from the left operand alone, the right operand is not evaluated. `|` is not short-circuited—that is, both operands are always evaluated. Run the program above. Explain why the two `if` statements in the code above behave differently?

Java also has two logical AND operators: `&&` (short circuited) and `&` (not short circuited). When the operators `|`, `||`, `&`, and `&&` have true/false operands, they function as **logical operators**. That is, they operate on true/false values and they yield true/false results. However, `|` and `&` (but not `||` and `&&`) can also have operands of an integer type, in which case they function as **bitwise operators**. For more information on the bitwise operators, see homework problem 6.

6) The | and & operators are overloaded. If their operands have true/false values, they function as logical operators (i.e., they operate on true/false values and they yield results that are true/false). But if their operands are an integer type (i.e., `byte`, `short`, `int`, or `long`), they function as **bitwise operators**. In a bitwise operation, the corresponding bits in the two operands are operated on, yielding the values given by the following tables:

bit 1	bit 2	|
0	0	0
0	1	1
1	0	1
1	1	1

bit 1	bit 2	&
0	0	0
0	1	0
1	0	0
1	1	1

Include the following code in a program and run:

```
int x = 12, y = 10, z1, z2;
System.out.println("x  = " + Integer.toBinaryString(x)); // display x
System.out.println("y  = " + Integer.toBinaryString(y)); // display y
z1 = x | y;    // bitwise OR
System.out.println("z1 = " + Integer.toBinaryString(z1));// display z1
z2 = x & y;    // bitwise AND
System.out.println("z2 = " + Integer.toBinaryString(z2));// display z2
```

Examine the values of `x`, `y`, `z1`, and `z2` that are displayed. Confirm that the bitwise OR and AND operators work as specified by the tables above.

7) Does the program below compile without errors? If so, does it run without errors? What is unusual about this program?

```
class C5h7
{
   public static void main(String[] args)
   {
      f();   // what about the value returned by f?
   }
   public static int f()
   {
      return 1;
   }
}
```

8) Does the program below compile without errors? If so, does it run without errors? What is unusual about this program?

```
class C5h8
{
   public static void main(String[] args)
   {
      int x;
      x = stingy();      // stingy does not return a value
   }
   public static void stingy()
   {
      System.out.println("hello");
   }
}
```

9) Change line 13 `C5h9.java` (a copy of Fig. 5.7) to

```
public static int x;
```

Compile and run the program. From the output displayed, deduce the initial value of `x`. Change line 17 to

```
int y;
```

Compile. What happens? Why? Reset line 17 to

```
int y = 2;
```

Then change `public` on line 15 to `private`. Compile. What happens? Why?

10) Compile the following program to determine what is wrong with it:

```
class C5h10
{
   public static void main(String[] args)
   {
      static int x = 3;
      System.out.println(x);
   }
}
```

11) Write a program in which `main` prompts for and reads in from the keyboard an integer. It should then call `f` passing it the value read in. `f` should display the letter `f` and then call `g`, passing it one less that the value in its parameter. On return from `g`, `f` should display `hello`. If the value passed to `g` is negative, `g` should immediately return to its caller. Otherwise, `g` should display the letter `g` and then call `f`, passing it one less than the value in its parameter. Before you run your program, predict what its output will be when the integer entered is 2.

12) Is it legal in a method to return a value of type `int` if the return type in the method's header is `double`. Is it legal in a method to return a value of type `double` if the return type in the method's header is `int`? Run a test program to check your answers.

13) Write a program in which `main` prompts for and reads in three integers, and then calls a `sort` method, passing `sort` the three integers read in. `sort` should display the three values in ascending order. Test your program with several different triplets of integers to make sure it works for all cases.

14) Write a program in which `main` calls `Math.cos` *and* `myCos` twice, once passing them `Math.PI/3.0` and once passing them `Math.PI/6.0`. `Math.cos` and `Math.PI` are both available in the predefined `Math` class. `myCos` is a method you should write. `main` should display the values returned by both methods. `myCos` should compute and return the value of

$$1.0 - x^2/(2!) + x^4/(4!) - x^6/(6!) + \ldots + x^{96}/(96!) - x^{98}/(98!)$$

where `x` is the `double` parameter that receives the value passed to `myCos`. Compute the value of each term by multiplying the value of the previous term by some appropriate factor (see homework problems 18 and 19 in Chapter 4

15) Write a program that has three static `f` methods (overloaded). Each `f` method should compute and display the average of the double values it is passed. One `f` method should have 2 parameters, the second 5 parameters, and the third 10 parameters. Call your `f` methods from `main` using

```
f(2.0, 99.5);
f(1.0, 2.0, 3.0, 4.0, 5.0);
f(1, 1, 1, 1, 1, 1, 1, 1, 1, 100.0);
```

16) Write a program that generates and displays 30 random numbers between 0 and 1. Your program should contain a `main` method, a `myRand` method, and a class variable `seed`. Each time `myRand` is called, it should return a random `double` value between 0 and 1. `main` should call `myRand` 30 times, each time displaying the number that `myRand` returns. `myRand` should use the following **algorithm** (i.e., step-by-step procedure) to generate each random number:

    1) Square the sum of `Math.PI` and `seed`. Put the result back into `seed`.
    2) Assign `seed` to an `int` variable `x` (you will need a cast).
    3) Subtract `x` from `seed`. Put the result back into `seed`.
    4) Return the value in `seed`.

    Initialize `seed` to 0.123456789. What is the effect of steps 2 and 3 on `seed`? How well does your `myRand` method work? Do the numbers it generates look like true random numbers? Compute the average of 10,000 numbers generated by your `myRand` method. Is it close to 0.5? Generate 10,000 numbers and count the number of numbers between 0.1 and 0.2 and between 0.4 and 0.5. Are the two counts close to 1000? Why can we not use a local variable within the `myRand` method for `seed`?

17) Suppose you want to determine the square root of `x`. Set `root` to 1.0. If `root` is less than the square root, of `x`, then `x/root` has to be greater than the square root of `x`. If `root` is greater than the square root of `x`, then `x/root` has to be less than the square root of `x`. Thus, regardless of the value of `root`, `root` and `x/root` bound the square root of `x`. That is, the square root of `x` is somewhere between `root` and `x/root`. By averaging `root` and `x/root`, we can get a better estimate of the square root of `x` than is in `root`. We can then assign this better estimate to `root`. If we repeat this process many times in a loop, the value in `root` will converge on the square root of `x`. Using this technique, implement a method that *returns* the square root of the value it is passed. Call your method `mySqrt`. Use your method *and* the `Math.sqrt` method to determine the square roots of 5.0E-100, 5.0, 500.0, 50000.0, 5000000.0, 500000000.0, and 5.0E300. When you program starts, it should prompt for and read in from the keyboard the number of times the loop that computes the square root should iterate. The larger the number entered (up to some limit), the more accurate the calculated square root. `Math.sqrt` is a static method in the predefined `Math` class. How many times does your method have to iterate to get an accurate value of the square root. Is there some way to get a starting value for `root` better than 1.0?

    At the machine language level, the computer can perform only the basic math operations, such add, subtract, multiply, and divide. This problem and homework problem 14 illustrate how a computer can be programmed to perform more sophisticated mathematical operations, such as square root and cosine.

18) Suppose a dart board is a square with dimensions 1 by 1, on which we draw a quarter circle with radius 1 and with the origin located at the lower left corner of the dart board. The area on the dart board that is inside the quarter circle is the "hit" area. The area on the dart board outside the quarter circle is the "miss" area. If we throw randomly at the dart board, the ratio of hits to throws will be roughly equal to the ratio of the hit area ($\pi r^2/4 = \pi 1^2/4 = \pi / 4$) to the total area ($1 \times 1 = 1$). Write a program that simulates randomly throwing a dart 1,000,000 times at this dart board. Estimate a value for π from the hits-to-throws ratio. *Hint*: To "throw" a dart, generate two random double values, each between 0.0 and 1.0. Treat these values as the (x, y) coordinates of the point on the dart board on which the dart lands. To determine if a throw is a hit, check if the distance of the hit from the origin is less than or equal to 1. If it is, then the throw is a hit; otherwise, it is a miss. This problem is particularly interesting because we are using a random process to determine the universal constant π.

To generate a random number, use the predefined `Random` class. Place the following statement at the beginning of your program:

```
import java.util.Random;
```

To create a `Random` object, use

```
Random r = new Random();
```

To generate a a random double value between 0.0 and 1.0 and assign it to x, use

```
x = r.nextDouble();
```