Write a program to simulate the duel using this strategy. Your program should use random numbers and the probabilities given in the problem to determine whether a shooter hits the target. Create a class named `Duelist` that contains the dueler's name and shooting accuracy, a Boolean indicating whether the dueler is still alive, and a method `ShootAtTarget(Duelist target)` that sets the target to dead if the dueler hits his target (using a random number and the shooting accuracy) and does nothing otherwise.

Once you can simulate a single duel, add a loop to your program that simulates 10,000 duels. Count the number of times that each contestant wins and print the probability of winning for each contestant (e.g., for Aaron your program might output "Aaron won 3,595/10,000 duels or 35.95%").

An alternate strategy is for Aaron to intentionally miss on his first shot. Modify the program to accommodate this new strategy and output the probability of winning for each contestant. What strategy is better for Aaron, to intentionally miss on the first shot or to try and hit the best shooter? Who has the best chance of winning, the best shooter or the worst shooter?

Note: To generate a random number x, where $0 <= x \leq 1$, use `x = Math.random();`. This generates a random number between 0 and 1.

4. You are interested in keeping track of the team members and competition information for your school's annual entries in computer science programming competitions. Each team consists of exactly four team members. Every year your team competes in two competitions. As an initial start for your database, create a class named `Team` that has the following data members:

```
// Name for the team
String teamName;
// Names for each team members.
String name1, name2, name3, name4;
// Info on each competition
Competition competition1, competition2;
```

Note that there is a much better way to represent the team members and competitions using arrays; this is covered in a subsequent chapter. The class should also have a method that outputs all the names of all team members and the competition information to the console.

The `Competition` class contains variables to track the following:

String: Name of the competition, Name of the winning team, Name of the runner-up

Integer: Year of the competition

Implement the `Team` and `Competition` classes with appropriate constructor, accessor, and mutator methods. In entering data for past competitions, you note that an entry is usually very similar to the previous year's entry. To help with the data entry, create a deep copy constructor for the `Team` class. Test your copy constructor by creating a copy of an existing team object, changing the competition information for the copy, and outputting the data for the original and the copy. The original object should be unchanged if your deep copy constructor is working properly.

myCodeMate

5. Define a class named `Money` whose objects represent amounts of U.S. money. The class should have two instance variables of type `int` for the dollars and cents in the amount of money. Include a constructor with two parameters of type `int` for the dollars and cents, one with one constructor of type `int` for an amount of dollars with zero cents, and a no-argument constructor. Include the methods `add` for addition and `minus` for subtraction of amounts of money. These methods should be static methods and should each have two parameters of type `Money` and return a value of type `Money`. Include a reasonable set of accessor and mutator methods as well as the methods `equals` and `toString`. Write a test program for your class.

   Part Two: Add a second version of the methods for addition and subtraction. These methods should have the same names as the static version but should use a calling object and a single argument. For example, this version of the `add` method (for addition) has a calling object and one argument. So `m1.add(m2)` returns the result of adding the `Money` objects `m1` and `m2`. Note that your class should have all these methods; for example, there should be two methods named `add`.

   Alternate Part Two (If you want to do both Part Two and Alternate Part Two, they must be two classes. You cannot include the methods from both Part Two and Alternate Part Two in a single class. Do you know why?): Add a second version of the methods for addition and subtraction. These methods should have the same names as the static version but should use a calling object and a single argument. The methods should be `void` methods. The result should be given as the changed value of the calling object. For example, this version of the `add` method (for addition) has a calling object and one argument. Therefore,

```
m1.add(m2);
```

   changes the values of the instance variables of `m1` so they represent the result of adding `m2` to the original version of `m1`. Note that your class should have all these methods; for example, there should be two methods named `add`.

myCodeMate

6. Define a class for rational numbers. A rational number is a number that can be represented as the quotient of two integers. For example, 1/2, 3/4, 64/2, and so forth are all rational numbers. (By 1/2 and so forth, we mean the everyday meaning of the fraction, not the integer division this expression would produce in a Java program.) Represent rational numbers as two values of type `int`, one for the numerator and one for the denominator. Your class should have two instance variables of type `int`. Call the class `Rational`. Include a constructor with two

arguments that can be used to set the instance variables of an object to any values. Also include a constructor that has only a single parameter of type `int`; call this single parameter `wholeNumber` and define the constructor so that the object will be initialized to the rational number `wholeNumber/1`. Also include a no-argument constructor that initializes an object to 0 (that is, to 0/1). Note that the numerator, the denominator, or both may contain a minus sign. Define methods for addition, subtraction, multiplication, and division of objects of your class `Rational`. These methods should be static methods that each have two parameters of type `Rational` and return a value of type `Rational`. For example, `Rational.add(r1, r2)` will return the result of adding the two rational numbers (two objects of the class `Rational`, `r1` and `r2`). Define accessor and mutator methods as well as the methods `equals` and `toString`. You should include a method to normalize the sign of the rational number so that the denominator is positive and the numerator is either positive or negative. For example, after normalization, 4/–8 would be represented the same as –4/8. Also write a test program to test your class.

Hints: Two rational numbers *a*/*b* and *c*/*d* are equal if *a***d* equals *c***b*.

Part Two: Add a second version of the methods for addition, subtraction, multiplication, and division. These methods should have the same names as the static version but should use a calling object and a single argument. For example, this version of the `add` method (for addition) has a calling object and one argument. So `r1.add(r2)` returns the result of adding the rationals `r1` and `r2`. Note that your class should have all these methods; for example, there should be two methods named `add`.

Alternate Part Two (If you want to do both Part Two and Alternate Part Two, they must be two classes. You cannot include the methods from both Part Two and Alternate Part Two in a single class. Do you know why?): Add a second version of the methods for addition, subtraction, multiplication, and division. These methods should have the same names as the static version but should use a calling object and a single argument. The methods should be `void` methods. The result is given as the changed value of the calling object. For example, this version of the `add` method (for addition) has a calling object and one argument. Therefore,

```
r1.add(r2);
```

changes the values of the instance variables of `r1` so they represent the result of adding `r2` to the original version of `r1`. Note that your class should have all these methods; for example, there should be two methods named `add`.

myCodeMate

7. Define a class for complex numbers. A complex number is a number of the form

```
a + b*i
```

where, for our purposes, *a* and *b* are numbers of type `double`, and *i* is a number that represents the quantity $\sqrt{-1}$. Represent a complex number as two values of type `double`. Name the instance variables `real` and `imaginary`. (The instance variable for the number that is multiplied by *i* is the one called `imaginary`.) Call the class `Complex`. Include a constructor with two parameters of type `double` that can be used to set the instance variables of an object to any values. Also include a constructor that has only a single parameter of type `double`; call this parameter `realPart` and define the constructor so that the object will be initialized to `realPart + 0*i`. Also include a no-argument constructor that initializes an object to `0` (that is, to `0 + 0*i`). Define accessor and mutator methods as well as the methods `equals` and `toString`. Define static methods for addition, subtraction, and multiplication of objects of your class `Complex`. These methods should be static and should each have two parameters of type `Complex` and return a value of type `Complex`. For example, `Complex.add(c1, c2)` will return the result of adding the two complex numbers (two objects of the class `Complex`) `c1` and `c2`. Also write a test program to test your class.

Hints: To add or subtract two complex numbers, you add or subtract the two instance variables of type `double`. The product of two complex numbers is given by the following formula:

```
(a + b*i)*(c + d*i) = (a*c - b*d) + (a*d + b*c)*i
```

Part Two: Add a second version of the methods for addition, subtraction, and multiplication. These methods should have the same names as the static version but should use a calling object and a single argument. For example, this version of the `add` method (for addition) has a calling object and one argument. So `c1.add(c2)` returns the result of adding the complex numbers `c1` and `c2`. Note that your class should have all these methods; for example, there should be two methods named `add`.

Alternate Part Two (If you want to do both Part Two and Alternate Part Two, they must be two classes. You cannot include the methods from both Part Two and Alternate Part Two in a single class. Do you know why?): Add a second version of the methods for addition, subtraction, and multiplication. These methods should have the same names as the static version but should use a calling object and a single argument. The methods will be `void` methods. The result is given as the changed value of the calling object. For example, this version of the `add` method (for addition) has a calling object and one argument. Therefore,

```
c1.add(c2);
```

changes the values of the instance variables of `c1` so they represent the result of adding `c2` to the original version of `c1`. Note that your class should have all these methods; for example, there should be two methods named `add`.