EXAMPLE GROUP MEETING

Exercise 1: Drawing Pictures

A Java program involves creating and manipulating *objects*, each of which provides some *operations*. An operation can either perform a task (like printing something on the computer screen), or it can do a computation and tell you the answer. Some operations require that you provide values to be used in their task/computation.

For this exercise, we'll assume that we have an *Artist* object named *picasso* that provides the following operations:

drawLineDown(int length)
    Draw a vertical line of the given length (in inches), starting from the current position and going straight down. The current position is changed to be at the bottom end of the line.

drawLineUp(int length)
    Draw a vertical line of the given length, starting from the current position and going straight up. The current position is changed to be at the top end of the line.

drawLineRight(int length)
    Draw a horizontal line of the given length, starting from the current position and going straight to the right. The current position is changed to be at the right end of the line.

drawLineLeft(int length)
    Draw a horizontal line of the given length, starting from the current position and going straight to the left. The current position is changed to be at the left end of the line.

moveRight(int d)
    Move the current position d inches to the right.

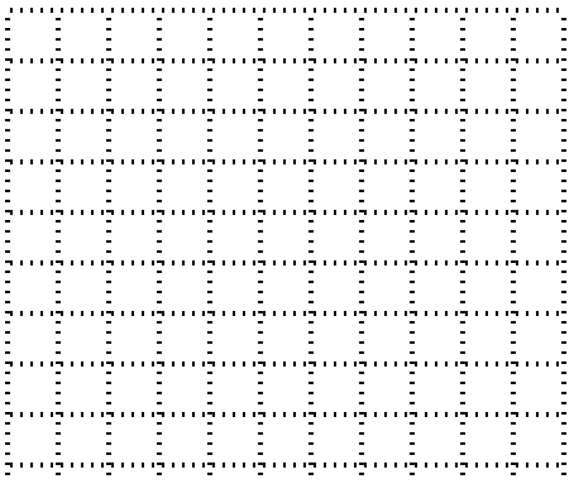
moveLeft(int d)
    Move the current position d inches to the left.

moveUp(int d)
    Move the current position d inches up.

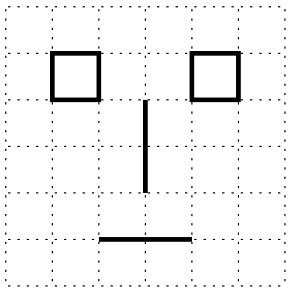
moveDown(int d)
    Move the current position d inches down.

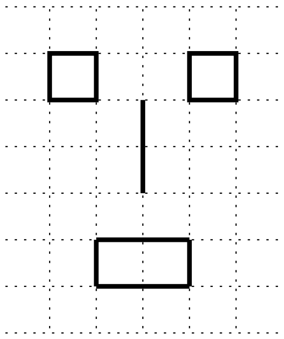
Part (a). What is drawn when the following code executes? (Use the grid to do the drawing; assume that the current position starts in the top left corner and that the squares in the grid are 1 inch high and 1 inch wide.)

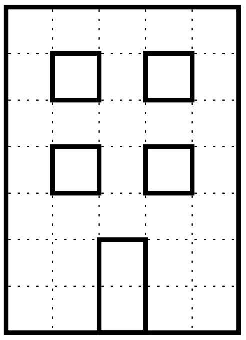
```
picasso.drawLineDown( 7 );
picasso.moveUp( 4 );
picasso.drawLineRight( 2 );
picasso.moveUp( 3 );
picasso.drawLineDown( 7 );
picasso.moveRight( 2 );
picasso.drawLineRight( 2 );
picasso.moveLeft( 1 );
picasso.drawLineUp( 7 );
picasso.moveLeft( 1 );
picasso.drawLineRight( 2 );
```



Part (b). Divide into groups of two. Each group choose one of the pictures shown below and write Java code that would make *picasso* draw the picture. (The dotted lines are not part of the pictures; they're there to show you how many inches you need to move or draw. Each dotted box is one inch on each side. Assume that the current position starts in the top left corner of the grid.)







Part (c). What other *Artist* methods would have made it easier to draw the pictures?

Part (d). Most programming languages, including Java, let you write *loops*. For example, in Java you can essentially say "repeat the following commands *n* times", Can you simplify any of the code you wrote for part (b) by using loops?

Exercise 2: The String class

The `String` class has many useful methods including the following:

`int length()`

Returns the number of characters in this string.

`String substring(int beginIndex, int pastEnd)`

Returns a new string that is the substring of this string that starts at position `beginIndex` and ends at position `pastEnd - 1` (the position of the first character is 0). Error if `beginIndex` is negative, or `pastEnd` is greater than the length of this string.

`int indexOf(int ch)`

Returns the position within this string of the first occurrence of `ch`. If no such character occurs in this string, then -1 is returned.

`int compareTo(String anotherString)`

Returns 0 if this string is the same as `anotherString`; returns a negative integer if this string comes before `anotherString` in lexicographic order (dictionary order); returns a positive integer if this string comes after `anotherString` in lexicographic order.

`boolean equals(Object anObject)`

Returns true if `anObject` is a `String` that represents the same sequence of characters as this string; otherwise returns false.

Part (a): Getting familiar with the `String` class

To get some practice using the `String` methods, play the following game (in pairs).

- Each person thinks of a word, 3 to 5 letters long. Whoever guesses their opponent's word first wins!

- Each person takes 2 cards.

- If you ever have more then one of the same card, you may trade one of them in for a new card if you wish. You may also trade in a card that is exactly the same as one you already played.

- Alternate turns. When it's your turn, pick up one new card, then play one of your 3 cards. If it has a blank (e.g., `s.indexOf(__)`), you get to fill in the blank with whatever you want. Your opponent tells you what value their word, `s`, would return if the method call on your card were made. For example, if your opponent is thinking of the word "hat" and you play `s.substring(1, 2)`, your opponent must say "a"; if you play `s.substring(2, 4)`, your opponent must say "error" (because "hat" has only 3 letters).

- You win if you play an `s.equals` card and your opponent says "true", or if you play an `s.compareTo` card and your opponent says 0 (i.e., you guessed their word).

Part (b): Algorithms

An *algorithm* is a clear, step-by-step description of how to perform a task. For example, here's an algorithm for determining whether a positive integer N is prime:

```
Step 1:  Let $k$ equal 2.
Step 2:  If $k$ is greater than $N/2$, stop: $N$ is prime.
Step 3:  If $N$ is evenly divisible by $k$, stop: $N$ is not prime.
Step 4:  Let $k$ be $k+1$.
Step 5:  Go to Step 2.
```

Assume that you have a `String` variable called `word`. Write an algorithm to determine whether the sequence of characters in `word` is a palindrome (i.e., is the same forward and backward). Start by writing your algorithm using "Step 1", "Step 2", etc, like the one given above (be sure you include a "stop" step). Then try writing actual Java code.

You might find it helpful to use the following `String` method (in addition to the ones given on the first page):

`char charAt(int index)`
    Returns the character at the specified index.

Exercise 3: Logical Thinking

Here are two logic puzzles. Try solving them now. If you don't finish, you can keep working on them during the week if you want to. We'll talk about the answers next time.

1. What are the ages?

   Two mathematicians are sitting together in a building. They don't have anything to do at the moment, so one says to the other, "Try guessing the ages of my three children. I'll give you a hint - the product of their ages is 72." The other mathematician says, "That's not enough information. Tell me more." The first mathematician says, "Their ages add to be the number of this building." The other mathematician goes outside, looks at the building number, comes back, and says, "That still isn't enough information. Tell me more." The mathematician says, "My youngest child's name is Anne."

   What are the ages of the children?

2. How many handshakes?

   A woman and her husband attend a party with four other couples. Some people shake hands with other people. Of course, no one shakes their own hand or the hand of the person they came with. When the woman asks the other (9) people present how many different people's hands they shook they all gave a different answer. Question (this is *not* a trick!): How many different people's hands did the woman's husband shake?

Exercise 4: More Logical Thinking

Assume that you have 8 coins, and you know that 7 are OK but one is bad. You know that the bad coin has a different weight than the good coins, but you don't know whether it's heavier or lighter.

Figure out how, using only a balance scale, you can find out which is the bad coin using just 3 weighings. Hint: Find a way to determine that half of the coins are OK with just 1 weighing.

Now figure out which is the bad coin assuming that you have *nine* coins, one of which is bad. (Still use just 3 weighings to find the bad coin.)

And now for a real challenge, do the same thing assuming that you have 13 coins.

Exercise 5: The Car Class

This exercise will help you to understand what happens when objects are declared and created, and when methods are called.

It will also help you to understand the difference between copying from one variable to another when the variable is an object, and when it is a primitive type (`int`, `double`, `boolean`, etc), as well as the difference between changing the values of variables that are objects vs primitive types.

First, take a look at the `Car` class defined on the next page.

Now execute the following code fragment; let one person play the role of each variable (`myCar, yourCar, anotherCar, oldSpeed`, and `currSpeed`). When a variable is assigned to, or one of its methods is called, the person playing the role of that variable should act out effects of the assignment or call.

```
Car myCar, yourCar, anotherCar;
int oldSpeed, currSpeed;

myCar = new Car("beep");
yourCar = new Car("honk");
anotherCar = myCar;

currSpeed = myCar.getCurrSpeed();
yourCar.changeSpeed(7);
anotherCar.changeSpeed(20);
currSpeed = myCar.getCurrSpeed();

myCar.blowHorn(2);
yourCar.blowHorn(3);
anotherCar.blowHorn(4);

oldSpeed = currSpeed;
myCar = yourCar;
currSpeed = myCar.getCurrSpeed();

myCar.changeSound("ooga");
myCar.blowHorn(currSpeed/5);
yourCar.blowHorn( yourCar.getCurrSpeed()/2 );
anotherCar.blowHorn( myCar.getCurrSpeed()/10 );
anotherCar = myCar;
```

```
class Car {
    /*******************
     * data members
     *******************/
    private int currSpeed;
    private String hornSound;

    /*******************
     * public methods
     *******************/
    /* constructor */
    public Car(String sound) {
        currSpeed = 0;
        hornSound = sound;
    }

    /* changeSound: change the horn sound */
    public void changeSound(String newSound) {
        hornSound = newSound;
    }

    /* blowHorn: blow the horn;
     * parameter numTimes tells you how many times
     */
    public void blowHorn(int numTimes) {
      while (numTimes > 0) {
          System.out.println(hornSound);
          numTimes--;
      }
    }

    /* changeSpeed: change speed */
    public void changeSpeed(int milesPerHour) {
        currSpeed = currSpeed + milesPerHour;
    }

    /* getCurrSpeed: return the current speed */
    public int getCurrSpeed() {
        return currSpeed;
    }
}
```