

INSTRUCTORS MANUAL

The following stuff are bits and pieces for instructors (e.g. quizzes and tests). All of this stuff is in incredibly raw form, so consider yourself forewarned and forearmed.

A lot of these sections are simply handouts you can hand your kids on an as-needed basis. Some are quizzes for the entire class. It is a bit touch-and-go.

INSTRUCTORS MANUAL	1
SETTING UP THE BOT SYSTEM—TO-DO LIST	2
IDEAS FOR STRENGTHENING METHODS WITH BOTS	3
SNIFFERBOT CODE	4
HOMEWORK: METHODS AND BOTS	5
THE EXTRA BOARDS	6
SOME BOT REINFORCEMENT	7
COMPUTERS ONE HOMEWORK	8
QUIZ: READING CODE	10
HW: METHOD DESIGN	12
QUIZ: CODE-READING	14
QUIZ: READING CODE	17
HW: MORE METHOD PRACTICE	20
COMPUTERS HOMEWORK: BOTS AND METHODS (9/16/04)	21
COMPUTERS HOMEWORK: GENERAL VARIABLES (9/16/04)	23
HW: A SMALL HOMEWORK ASSIGNMENT	25
COMPUTER HOMEWORK: MORE RANDOM METHODS	27
COMPUTER 1 HOMEWORK ASSIGNMENT	29
HELP WITH SMARTBOT AND WALL FOLLOWING	31
HW: METHOD DESIGN	33
COMPUTERS HOMEWORK: BOTS AND METHODS #3	35
QUIZ: BOUNCE PROGRAMMING QUIZ	36
QUIZ: MELTING PATH PROGRAMMING QUIZ	37
QUIZ—THE MELTING PATH	ERROR! BOOKMARK NOT DEFINED.
QUIZ: BULLSEYE PROGRAMMING QUIZ	38
HW: BOTS AND METHODS #4	39
HW: BOT METHOD #5	40
HW: AN INVESTIGATION OF OBJECTS AND INHERITANCE	41

SETTING UP THE BOT SYSTEM—TO-DO LIST

Here are some things to remember when setting up your year:

- 1) Make sure the bot package can be seen on the computers the students will be using. Also make sure the local directory is in the classpath (or seen) so that wherever the students put their files, there will be no problem.
- 2) Remove the bot.ps package, or remove all the java files from the bot package so the students can't get at the solution code.
- 3) Make sure that the text files for Haiku and Jokes are either in the bot/gen folder, or in the student problem set folder. The former is easier to handle.
- 4) Make the non-generator boards. You need to make the board files for the problem sets that do not have board generators. These are:
 - a. ToddleBot
 - b. DancerBot

In bot/boards is a Makefile that will generate all these boards. The bot system will look for these files in that location, but you can also put them on the server and have the students copy the files over directly—it might teach them more about files and information that way.

IDEAS FOR STRENGTHENING METHODS WITH BOTS

Methods seem to elude students, or some students at any rate. Here are some scattered thoughts on trying to address this.

Where I have seen methods in bots:

Taking multiple steps
steps(5)

feelBomb()

Should I teach "return;" without a value?

For: it appeals to making code writing easier for students

Against: bad style – but is it?

Yes, because then you can bail out easily? Or you end up with weird while loops and messiness, or you have to explain while(true)

assignment ideas

Write a bot that does blah, blah, blah.

Give an algorithm and have them implement it in code

Or: give out algorithms in lag time — so they get the coding experience but can move faster.

Does this undercut the students? Or is it doing parallel advancement—working on conceptual with folks who 'know' programming language, but work on programming language for those who need it—

Because now they are being asked to do too much.

SNIFFERBOT CODE

Stuck on SnifferBot? Here is the algorithm in English—can you translate it into Java?

```
// two variables are needed: the "old smell" and the "new smell",
// so we can compare the two with each step.

// set up the while loop's variables
Sniff, stash in "old smell" variable

// find clear space to take step to start cycle
While feel wall
    Right
Step

// do the while loop
While not done:

    Sniff

    If what you just smelled is 1 then
        Spin until key is felt, then grab it. (a method)

    Otherwise if what you just smelled is weaker than what you
    smelled before
        Turn right

    Otherwise if you don't feel a wall
        Step

    Otherwise
        Turn right

    Stash what you just smelled in the old smell variable
```

HOMEWORK: METHODS AND BOTS

Write a bot with the following methods. Headers are provided:

- a) `public void turnaround() // spins bot 180 degrees`

- b) `public boolean feelBomb()`

- c) `public void clearFront() // turns until nothing is in front of bot
// warning: make sure it doesn't loop forever if surrounded!`

- d) `public boolean stringIsUppercase(String what)
// returns true if what is all uppercase`

- e) `public void randomSearch()
// turn a random direction and go a random distance from 1 – 5 steps
// stopping if a key is hit and not accidentally hitting a wallsteps
// stopping if a key is hit and not accidentally hitting a wall`

- f) `public boolean maintainMarker()
// Check if there is a marker in front of bot, and if there is, grab it,
// and place a new marker with a tag 1 greater than it. Then return
// true. If there isn't a marker, place a marker with a tag of 0 and
// return false.`

THE EXTRA BOARDS

All the board generators are listed in the index of the bot book (and are used either in the in-text examples or the problem sets), except for the following ones.

These are in ascending order of topic, sort-of.

Plus – bot is in center of plus. Key is in one leg. Extra pain are longer legs

Winder – A mix of loops and IFE structures are needed for this one.

PlusLoop – Four branches with little loops on the end so left-hand-wall doesn't work. Simple logic needed.

Whiskers – another logic one that defeats left-hand-wall follower.

Islands – using orbs to jump around, it is a problem solving maze with left-hand-wall following solution

MarkerChain – Using a bunch of markers, you pull numbers off them and take that many steps, and have to skip the bad keys.

Used as Quizzes

Bounce – back and forth needed to solve. Has extra pain with random key placement. Used on a quiz.

BullsEye – A bullseye, with extra pain of acid. Used as a quiz (harder than Bounce).

Miscellaneous

Plain – this is a large empty board with the bot in the middle.

CritterRoom – this is a work in progress. A board with moving creatures. Doesn't quite work yet.

DirectoryGenerator – this is if you want to run through all the boards in a given directory. It uses the local directory, pulls all the brd files and goes from there.

SOME BOT REINFORCEMENT

If you are having some difficulty with loops, try doing the following to get more of a grasp of it.

bot.gen.UBoard and bot.gen.RandomHook

With the “extra Pain” option, the key is placed randomly. Can you modify your original UBoard and RandomHook bots to get the key and stop?

bot.gen.XBoard

First try doing this with just if-then-else statements. Once you do this that way, try doing it with loops. Then try doing the “extra pain” option, which makes the legs longer.

bot.gen.BounceBoard

Hint: why is it called a bounce board?

COMPUTERS ONE HOMEWORK

Here are some problems, the sort of which you might expect on some kind of pop quiz, or similar exercise. These should not take you too long—certainly do not spend more than half an hour or so. You should not use a computer when answering them.

0) What does the following code do?

```
void dance( int sc ) {
    while( sc > 0 ) {
        left();
        if ( sc > 3 )
            right();
        sc = sc - 1;
    }
}

public void compute() {
    int val = 4;
    dance( val );
    dance( val );
}
```

1) What possible errors can the following code generate, and how?

```
Thing t = grab();
Marker m = (Marker)t;
if ( m.tag() == Heading.NORTH ) {
    step();
}
```

2) Find and correct the five syntax errors in this code:

```
while( feelOrb() = true ) {
    step();
    If ( feelKey )
        grab;
        done = true;
    else
        right();
}
```

3) What is wrong with the code below and how would you fix it? Describe what it does now that you have fixed it.

```
void takeOrbAction() {
    if( feelOrb() ) {
        int count;
        while( !feelWall() ) {
            count = count + 1;
            step();
        }
    }
    right();
    while( count > 0 ) {
        step();
        count--;
    }
}
```

4) Write, by hand, code to do the following (you should not need a computer):

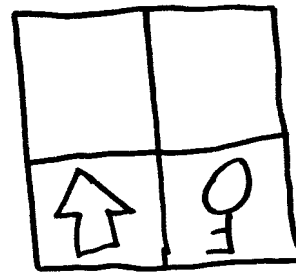
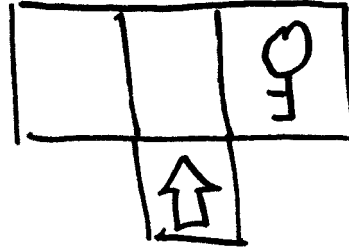
Walk until you hit a wall, turn right, grab something if it is in front of you, turn right again, and then walk until you are back at start.

QUIZ: READING CODE

#0) Where does the bot end up if it is placed as shown on the first board? How many times does it say "yah! "? Do the same for the second board.

```
void boogie() {
    left();
    right();
    say( "yah!" );
    left();
    while( feelWall()
) {
    right();
    }
    step();
}

public void compute() {
    while( !feelKey()
) {
    say( "rockin!"
);
    boogie();
    }
    grab();
}
```



#1) Fix the following code's syntax so it compiles. (There are multiple solutions, any of them is fine.)

```
public void compute() {

    while( feelkey == false )

        step

            if ( feelWall == false )
            }
            right();

            if ( feel wall == true ) && ( feel key =
false ) {
                left
```

```
        }  
    }  
}
```

#2) What is a method?

#3) Turn the following into code:

```
key while not at a wall, do the following:  
    take a step, turn right, and if in front of a  
    grab it, otherwise turn back.
```

HW: METHOD DESIGN

This assignment is to work on simple method design, parameters, and return values. You should turn in your work as a collection of printouts.

0) Consider the following code:

```
public class Fiddle {
    int pump( int v ) {
        v = v + 13;
        if ( v > 5 )
            return 3;
        else if ( v < 0 )
            return 6;
        else
            return 1;
    }
    public void doIt() {
        int v = 12;
        int s = 15;
        int r = pump( v );
        s = pump( r );
        Body.say( "v = " + v );
        Body.say( "s = " + s );
        Body.say( "r = " + r );
    }
    public static void main( String[] args ){
        Fiddle f = new Fiddle();
        f.doIt();
    }
}
```

- a) What are all the methods in this class?
- b) What is the name of the first parameter in the pump() method?
- c) What is the type of the first parameter in the pump() method?
- d) For each method, list the parameters and return types.
- e) What does the code do? (Do not use a computer for this.)

1) Write the headers (not the bodies) for methods that meet the following criterion:

- a) A method called blather that takes two strings, one which is some question, and one which is the name of the asker, and returns a

true / false value where true means the question is to pass, and false means it is not.

- b) A method called tank which takes an array of ints and returns the average of all those ints.
- c) A method called deal that takes an array of Card objects and a Player object and returns a Hand object.

For 2-5, you must also write a little program that demonstrates that the method works.

- 2) Write a method that takes two numbers and returns the average of them, rounded down.
- 3) Write a method that takes three numbers and returns true or false on whether some two of the numbers have an average equaling the third number. Use (2) for this.
- 4) Write a method that returns a String that the user types in. The method should make sure the user does not hit return without typing something in.
- 5) Write a method that takes two Strings and returns the average number of characters in them.

QUIZ: CODE-READING

0) What is a NullPointerException? Explain in language you understand.

1) What is a variable? What is the difference between a primitive variable and a non primitive variable?

2) What is "scope"? Illustrate with an example.

3) What is wrong with the following code:

```
int[] arr = new int[ 15 ];
for ( int i = 1; i <= 15; i++ ) {
    arr[i] = i * i * i;
}
```

4) Fix the following code's syntax so it compiles. (There are multiple solutions, any of them is fine.)

```
public void compute() {
    {
        while feel key == false
            step
                if ( feelWall == false ) }
                    right();
        if ( feel wall == true) && ( feel key =
false ) {
            left
            }
        }
    }
```

- 5) Where does the bot running the following code end up if it is placed as shown on the board given? How many times does it say "grumble"?

```
public void compute() {
    boolean gumpy;
    gumpy = false;
    while( gumpy == false ) {
        say( "grumble." );
        if ( feelKey() ) {
            gumpy = false;
            grab();
        }
        if ( feelWall() ) {
            right();
        }
        else {
            step();
        }
    }
}
```

- 6) Turn the following into code:

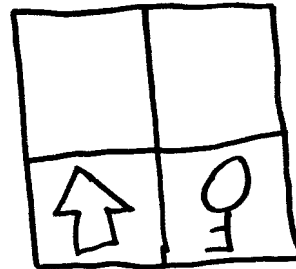
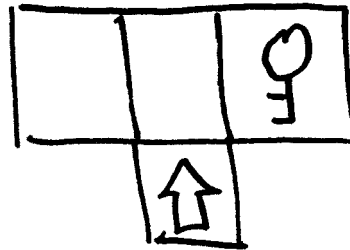
```
while not at a wall, do the following:
    take a step, and if you feel a marker,
    turn left
```

QUIZ: READING CODE

#0) Where does the bot running the following code end up if it is placed as shown on the first board? How many times does it say "yah! "? Do the same for the second board.

```
void boogie() {  
    left();  
    say( "yah!" );  
    while( feelWall()  
    ) {  
        right();  
    }  
    step();  
    right();  
}
```

```
public void compute() {  
    while( !feelKey()  
    ) {  
        say( "rockin!"  
    );  
        boogie();  
    }  
    grab();  
}
```



#1) What is a method? Demonstrate this concept by writing a short example of one, as well as code using it.

#2) Fix the following code's syntax so it compiles. (There are multiple solutions, any of them is fine.)

```
public void compute {  
    while( feelkey == false )  
        step  
            if ( feelWall == false )  
                }  
                right();  
            if ( feel wall == true ) || ( feel key =  
false ) {  
                left  
            }  
        }  
    }
```

#3) When will the bot step in the following? Be specific. Say “always” if it will always step and “never” if it will never step.

a)

```
if ( feelWall() == !feelWall() ) {  
    step();  
}
```

b)

```
if ( true && ( !false == (true || false) ) ) {  
    step();  
}
```

c)

```
if ( feelWall() != feelKey() ) {  
    step();  
}
```

#4) Turn the following into code:

```
while not at a wall, do the following:
```

take two steps, turn left, and if in front of
a key grab it, otherwise turn 180 degrees.

HW: MORE METHOD PRACTICE

Start with a class like this:

```
import console.*;

public class HW {
    public static void main( String[] args ) {
        HW h = new HW();
        h.run();
    }

    public void run() {
        Body.say( "hi!" );
    }
}
```

Get this going and make sure it works. Now start working on the stuff below. Make sure that each piece works before going on to the next one.

1) Write a method that asks for a number from 1-5 and returns that number. It should keep asking as long as the user does not enter a valid response. Here is the header:

```
int askForNumber()
```

You can test this method in your run() method like this:

```
int v = askForNumber();
Body.say( "You typed: " + v );
```

2) Make a method that takes a value and prints out that value times 5. Here is the method header:

```
public void printCode( int v )
```

3) Now make a method that sets a variable to a random number from 1 to 5 and then asks the user for a number from 1-5. It should then compare these two numbers and tell the user that they won if their number is equal to the random number.

COMPUTERS HOMEWORK: BOTS AND METHODS (9/16/04)

Each piece (but the last two) of this assignment, below, asks you to write a method that does a particular task. Write that method, compile it, and make sure it works. Your compute method should test all your code. Headers for the first two are provided. The second-to-last piece asks you to use the methods to solve a maze—it should be easier than before once your methods are in order.

This assignment is to get you used to making small methods for small tasks, and then using them to do larger tasks. Tell me if it works.

1)

```
/**
 * spins bot 180 degrees */
public void turnAround()
```

2)

```
/**
 * Goes to a wall and stops. */
public void goToWall()
```

So at this point you would have a compute that looks something like this:

```
public void compute() {
    turnAround();
    goToWall();
    say( "I should be at the bottom wall." );
}
```

3) Goes to either a wall or a key, and stops.

Are you compiling and running your bot after each one to make sure it works?

4) Takes ten steps.

For the next few, give your bot a stepCount field (i.e. an int variable at the top of your Bot code called "stepCount". The following methods will change that variable.

5) Go to the wall, setting stepCount to the total number of steps taken.

Make sure the above method works if you use it twice. Test it by making your bot say the value of stepCount:

```
say( "Stepcount = " + stepCount );
```

Now test it by having your bot say the value of stepCount, turning right, going to another wall, and saying stepCount again. Does your method work multiple times in a row, or does stepCount not get reset to 0?

6) Take steps, subtracting one from stepCount, until stepCount equals zero.

Test this method by running the following code:

```
goToWallAndCount();  
right();  
right();  
takeStepCountSteps();
```

7) Make a new Bot and copy over all the methods you made above. Now, with your new Bot, and using some of the above methods, solve bot.gen.Whiskers. Even if you have already solved Whiskers, do it again with the above. Note that you will not need all of the above methods, so don't get confused if you can't figure out how to use one or another of them.

8) Did having methods help you with #7? Did doing this homework help you understand how to make methods, or help you learn how to code in any way?

COMPUTERS HOMEWORK: GENERAL VARIABLES (9/16/04)

This homework does some basic work on using general variables and writing methods. It is a mixture of written work and programming. You should be sure to compile and check all the programming parts thoroughly. Your written parts should similarly be edited for clarity—remove all English syntax errors as well as Java ones!

When you do this homework, it should all be in a single Bot file. At the top of the file you should have a long comment (start the comment with “/*” and end it with “*/” that has all the answers to the written questions in it (unless otherwise noted in the text). You are going to print out and turn this combination of Bot code and written work.

Recall that feel() gives you back whatever is in front of the Bot. If you put what the bot feels into a variable, you can check to see if the bot felt various things. For example:

```
Thing t;
t = feel();
if ( t == null ) {
    say( "I feel nothing!" );
} else if ( t instanceof Wall ) {
    say( "I feel a wall!" );
} else {
    grab();
}
```

1) Put the above code in a method (give the method an appropriate name) and then call that method inside compute(). *Be sure to compile and run your Bot in a variety of situations to make sure this method works!* Try using a board such as bot.gen.LitteredMaze to test your method.

2) Describe what the method does in a comment right above the method. So your method would look like:

```
/**
 * This method blah, blah, blah...
 */
public void methodName() {
    ... etc ...
}
```

3) Write a method that feels in front of the Bot, and spins around in happiness if there is a Key, and says, “PANIC!” if there is a Bomb. Use bot.gen.LitteredSpiral to test your method.

4) Write a method that walks forward 4 steps, counting the number of things it feels on the way. Note that for this, your Bot will have two integer variables. The first is to count the steps, and the second is to count the number of things it feels. The Bot should feel in front of it after each step.

When checking the above, use `bot.gen.LitteredRoom` for a good debugging environment.

If you get stuck with 4, or are just feeling ambitious, try making a method that walks up to the next wall and stops, counting the number of non-wall things it feels along the way.

5) Write a method that makes your Bot walk up to the next non-Wall Thing in front of it and then stops. If it reaches a wall, it should turn right and keep going. When doing this method, start with the sample code in the book in the “General Variables” section

6) If you haven't, read up to Problem Set 10 (CollectorBot). Be sure to do the mini-problem sets. In fact, put the methods you write for these right in your Bot.

7) What does `while(tentativeStep()) {}` do? Put your answer in a comment at the top of your Bot code.

8) Write a method that feels in front of the bot, and if there is anything there, examines it, using the `examine()` method, and then says what it discovered.

9) Write a `handleThing()` method that checks in front of the Bot, and if there is a non-bomb, non-wall object, makes the Bot grab it.

10) Put a comment at the top of your code telling me if this assignment helped you at all, what you learned, whether it was too frustrating, etc. This description should be at least a few coherent sentences.

HW: A SMALL HOMEWORK ASSIGNMENT

1) In the following code, what would the bot do? (If the bot would do different things depending on where it was, explain.)

a)

```
if ( 5 == 5 ) {  
    step();  
} else {  
    right();  
}  
step();
```

b)

```
if ( feelWall() || feelPit() ) {  
    right();  
}
```

c)

```
if ( !( feelWall() || (feelWall() && feelPit()) ) ) {  
    step();  
}
```

d)

```
if ((feelWall() == false || ( feelPit() == !feelWall() ))) {  
    step();  
}
```

e)

```
if ( true || (false || (!false && (false == false))) ) {  
    right();  
}
```

- 2) Fix the syntax of the following code:

```
import bot

public class MyBot EXTENDS Bot {

    Public Void Compute() {

        while( feelWall ) {
            right

        if ( feelKey() ) {
            grab();
        }
    }
}
```

- 3) Write a method, `checkFour()`, which makes the bot face a key, if there is one, and face its original direction if there is no key next to it.

- 4) Is this valid code? How would you fix it, if it is not. What does the code do?

```
while( !feelKey() ) {
    step();
} else {
    grab();
}
```

- 5) What does the following method do?

```
while( !feelKey() ) {
    if ( feelWall() ) {
        right();
    } else {
        step();
    }
}
```

COMPUTER HOMEWORK: MORE RANDOM METHODS

This is just hammering away at methods, but this time we have parameters and return values

Turn in your work as a collection of printouts. One should be all the written stuff, and one should be your Bot. If you like, you can have the written stuff as a big comment at the top of your Bot code. Either way, your name should be on all pages.

0) Consider the following code:

```
public class MathBot extends Bot    {

    int tot;

    boolean isIt( int v, int q ) {
        if ( v * v == q ) {
            return true;
        } else {
            return false;
        }
    }

    public int countToWall() {
        steps = 0;
        while( !feelWall() ) {
            step();
            steps++;
        }
        return steps;
    }

    public void compute() {
        int cntr;
        cntr = goToWall();
        right();
        tot = cntr + goToWall();
        if ( isIt( cntr, tot ) ) {
            say( "IT IS TRUE!" );
        }
    }
}
```

- a) What are all the methods in this class? For each method, list the parameters and return types.
- b) What do the first two methods do?
- c) What does the code do? (Do not use a computer for this.)

1) Write the headers (not the bodies) for methods that meet the following criterion:

a) A method called `useFiveTimes` that takes a `Thing` object and uses it five times.

b) A method called `blather` that takes two `Strings`, one that is a question, and one that is the name of the asker, and returns a `true/false` value where `true` means the question is good, and `false` means it is not.

c) A method called `avg` that takes three `ints` and returns the average of all those `ints`.

For 2-5, you must write an actual Bot on your computer that demonstrates that all these methods work. So your Bot should have a `compute()` method that calls all these methods.

2) Write a method that take a number and makes your Bot turn right that number of times. Here is the header:

```
public void rightTurn( int num ) {
```

3) Write a method that takes a `Thing` and returns `true` if the `Thing` is a `Stick` or an `Orb`

```
public boolean isStickOrOrb( Thing t ) {
```

4) Write a method counts the number of `Walls` next to the bot, and returns that number.

```
public int numWalls() {
```

5) Write a method that takes a `String` and says it twice.

```
public void sayWithEcho( String s ) {
```

COMPUTER 1 HOMEWORK ASSIGNMENT

This assignment is to work on simple method design, parameters, and return values. You should turn in your work as a collection of printouts. Your name should be on all pages.

0) Consider the following code:

```
public class FiddleBot extends Bot {
    int pump( int v ) {
        v = v + 13;
        if ( v > 5 ){
            return 3;
        } else if ( v < 0 ) {
            return 6;
        } else {
            return 1;
        }
    }
    public void compute() {
        int v = 12;
        int s = 15;
        int r = pump( v );
        s = pump( r );
        say( "v = " + v );
        say( "s = " + s );
        say( "r = " + r );
    }
}
```

- a) What are all the methods in this class?
- b) For each method, list the parameters and return types.
- c) What does the code do? (Do not use a computer for this.)

- 1) Write the headers (not the bodies) for methods that meet the following criterion:
 - a) A method called blather that takes two strings, one which is some question, and one which is the name of the asker, and returns a true/false value where true means the question is to pass, and false means it is not.
 - b) A method called tank which takes an array of ints and returns the average of all those ints.
 - c) A method called deal that takes an array of Card objects and a Player object and returns a Hand object.

For 2-4, you must also write a little Bot that demonstrates that all these methods work.

- 2) Write a method that takes two numbers and returns the average of them, rounded down.
- 3) Write a method that takes three numbers and returns true or false on whether some two of the numbers have an average equaling the third number. Use (2) for this.
- 4) Write a method that takes two Strings and returns the average number of characters in them.

HELP WITH SMARTBOT AND WALL FOLLOWING

This does some work on methods, and gives you a structure to run through SmartBot

SmartBot is supposed to follow the left-hand wall. While a human can easily describe this, e.g., “stick out your left hand, and keep it on the wall to your left as you walk through the maze”, it can become a bit tricky when trying to codify it in Java code.

1) The most basic way to cut this problem up into smaller pieces is to realize that the Bot will be essentially doing the same thing over and over again, until it reaches a key. This means we have a “while not done” loop with a method in it called something like, “takeOneStep”.

2) Now we need to look at this “takeOneStep” method. How do we do this as a Bot? Well, first let us write the code in *pseudocode* (a mix of Java and English). It might be something like:

```
If there is a wall to your left:
    If there is no wall right in front of you:
        take a step
    But if there is:
        turn right
    ... etc ...
```

And so on.

Once it is in English, it should be possible to turn it into actual code. What can one do about such sentences as “if there is a wall to your left”? Well, we can usually make methods out of them! For the above we could invent a method called “feelWallOnLeft()” (see the book for a hint on this) and then have a nice and simple line of code such as:

```
if ( feelWallOnLeft() ) {
    ... etc ...
```

So write up all this. Don’t worry about keys at all. Just worry about getting the bot to follow the left hand wall.

3) At this point, you should be able to compile your code and test it. Does your bot follow the left hand wall?

4) Now worry about the key. Make a new method, “keyCheck”, that checks in front of the Bot for a key and grabs it if there is one there. keyCheck should also set done to true if the key is grabbed. Now your bot is fully functional!

Still Stuck?

Even if you are stuck, you should be able to get the following methods working:

```
void keyCheck()
boolean feelWallOnLeft()
```

Once you get those working (as in you have compiled them and tested them in a variety of situations), fiddle around with getting your Bot to walk along a corridor until it reaches an exit to the left. Make that a method. What happens if the Bot reaches a dead end before finding a left-hand exit? Fix it so the bot turns in such a circumstance. Make all this a method. What happens when you call this method over and over? The key here is to *play* when you get stuck. Write methods that sort of do what you want, and then see if you can improve them. **COMPILE AND RUN YOUR CODE FREQUENTLY!**

HW: METHOD DESIGN

This assignment is to work on simple method design, parameters, and return values. You should turn in your work as a collection of printouts.

0) Consider the following code:

```
1      public class FiddleBot extends Bot {
2
3          int fork;
4
5          int fubble( int al, int bel ) {
6              int tot;
7              tot = al * bel + 1;
8              return tot;
9          }
10
11         int pump( int v ) {
12             if ( v > fubble(3, 5) ) {
13                 return 3;
14             } else if ( v < 0 ) {
15                 return v;
16             } else {
17                 return fork;
18             }
19         }
20
21         public void doIt( String msg ) {
22             fork = 12;
23             int s = 15;
24             int r = pump( s );
25             s = pump( r );
26             say( msg );
27             say( "fork = " + fork );
28             say( "s = " + s );
29             say( "r = " + r );
30         }
31
32         public void compute( ) {
33             doIt( "The Way!" );
34         }
35     }
```

- List all the methods defined in this class. For each method, list the parameters and return types.
- List all places (give line numbers) where a variable is declared.
- List all places (give line numbers) where a variable is assigned.
- List all the fields in this class.
- What does the code do? (Do not use a computer for this.)

- 1) Write the headers (not the bodies) for methods that meet the following criterion:
 - d) A method called `tryAndFindIt` that takes a page number, and returns a boolean on whether it found a book with that page number.
 - e) A method called `wallDetect` that takes no parameters, and returns the number of empty spaces next to the bot.
 - f) A method that follows the left hand wall for a given number of steps or until it finds something, and then returns what it found (null if nothing found).

For 2-6, you must also write a Bot that demonstrates that the method works. Turn in a printout of the bot. You can have the answers to 0 & 1 in a comment at the top of your bot.

- 2) Write a method that, in `Room` or `BigRoom`, goes to the corner and stops.
- 3) Write a method that says a given String in all uppercase with a bunch of exclamation points after it.
- 4) Write a method, `ask()`, that says a passed question, then asks for response from the human, returning that response in all lowercase.
- 5) Write a method that takes a given number of steps.
- 6) Write a method that takes two numbers, an `x` and a `y`, and goes north `y` steps and goes east `x` steps. For extra credit, make it go west if `x` is negative, and go south if `y` is negative. Use the method from #5 in writing this method.

COMPUTERS HOMEWORK: BOTS AND METHODS #3

Write a bot with the following methods. Be sure to have a compute that tests all your code. Headers are provided.

- 6) /**
* spins bot 180 degrees */
public void turnAround()
- 7) /**
* returns true if bot feels a bomb, false for anything else */
public boolean feelBomb()
- 8) /**
* turns until nothing is in front of bot
* warning: make sure it doesn't loop forever if surrounded! */
public void clearFront()
- 9) /**
* returns true if what is all uppercase */
public boolean isStringUppercase(String what)
- 10) /**
* turn a random direction and go a random distance from 1 – 5 steps
* stopping if a key is hit and not accidentally hitting a wall. */
public void randomSearch()
- 11) /**
* Check if there is a marker in front of bot, and if there is, grab it, and
* and place a new marker with a tag 1 greater than it. Then return true
* If there isn't a marker, place a marker with a tag of 0 and return false. */
public boolean maintainMarker()

QUIZ: BOUNCE PROGRAMMING QUIZ

A) Do the written quiz, which you should now have in your hands.

B) Run `bot.Runner bot.ps.WallBot bot.gen.Bounce`

You should see a long, horizontal hallway with acid at each end. The key is at one end or the other.

C) Turn off your airport so you are not using the Internet.

D) Write a bot to solve this maze.

The key is always 6 spaces to one side or the other.

You are allowed to look at any code you have written, but are not allowed to look at anyone else's code, or talk to anyone else. This is just like any other test. You are allowed to use your bot book.

E) Extend your bot to solve the "extra Pain" option for the maze.

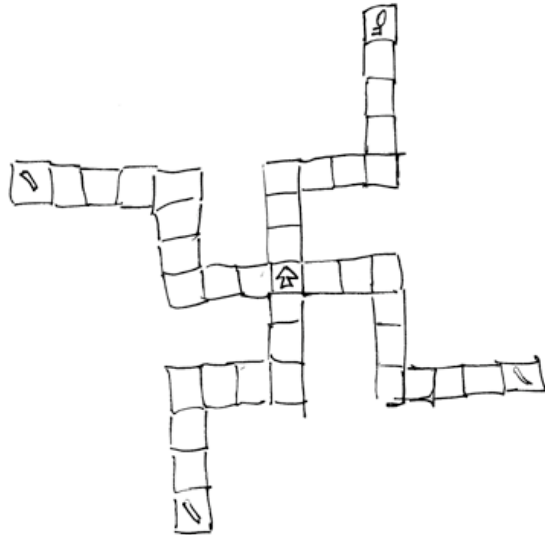
Save a backup so if you can't do this part, you still get credit for the first part!

In this maze, the corridor is a random length, and the key is randomly placed. However, the bot is guaranteed to be in the exact center of the corridor. Your bot should not use the Sniffer, if at all possible.

F) When finished, put a comment at the top of the bot with your name and how long it took to do the problem set (this will *not* affect your grade). Print out a hardcopy of your bot, and turn it in.

QUIZ: MELTING PATH PROGRAMMING QUIZ

Your Bot is in the center of a big swirly plus. Three of the endpoints have simple sticks, and one has the key. Your Bot needs to make the trip to go get the key, and then it must return to the center, starting square.



- 1) Download the latest version of bot.jar from the CompSci server. Make sure the bot.gen.MeltingPath maze works by running a working Bot on it. (You will see only one of the four branches pictured above—don't panic, that is what you should see). Once you have verified you have all the working files, turn off your internet connection by going to the little radar picture in the upper-right.
- 2) Now, write a step() method that takes a number and takes the passed number of steps. Right now, the above maze is a fixed number of steps for each branch (3, to be exact).
- 3) Now solve the maze. The simple maze has only one branch to the north (so ignore all other exits. Remember that after the bot gets the key, it needs to return back to the starting square!
- 4) Now solve the "extra pain" maze, which has all four branches. Think *methods* when writing your code!
- 5) Now make it do any difficulty level. (Change the difficulty with the slider bar under the file menu.) The difficulty controls how long each segment of each branch is. Your bot is going to need to count the number of steps in the branch, and then remember this number in order to get back to start.

QUIZ: BULLSEYE PROGRAMMING QUIZ

A) Take a big breath.

B) Run `bot.Runner bot.ps.WallBot bot.gen.BullsEye`

You should see a two-ring bull's-eye with a vertical corridor. If not, get help with part (A).

C) Turn off your airport so you are not using the Internet. (And so no one can spy on you!)

D) Write a bot to solve this maze.

The key is somewhere in the maze. The bot starts in the center, and has a marker under it with the radius of the bull's-eye, which you may use if you choose to. For this level, however, the maze's dimensions are completely fixed and unchanging.

You are allowed to look at any code you have written, but are not allowed to look at anyone else's code, or talk to anyone else. This is just like any other test. You are allowed to use your bot book.

Note: you can do E or F in any order. I recommend E first.

E) Extend your bot to solve various maze difficulties.

Save a backup so if you can't do this part, you still get credit for the first part!

These are all various-radius bull's-eyes (which you can get by adjusting the difficulty slider-bar. You can set the difficulty, from the command line, with `"-d [1-100]"` as one of your command-line arguments.

F) Extend your bot to solve the "Extra Pain" option for the maze.

Save a backup so if you can't do this part, you still get credit for the first part!

In this maze, the corridor's inner rings are all acid, so you need to remember where you are in the bull's-eye. You can start with "extra pain" on by running the board with the `"-p"` option on the command line. Your bot should not cheat, btw.

G) When finished, put a comment at the top of the bot with your name and how long it took to do the problem set (this will *not* affect your grade). Print out a hardcopy of your bot, and turn it in!

HW: BOTS AND METHODS #4

Write a bot with the following methods. Be sure to have a compute that tests all your code.

12) A method that puts the bot one space to the left, facing the same direction.

```
13) /**
    * returns true if bot feels a bomb, false for anything else
    */
    public boolean feelBomb()
```

14) A method that returns the number of bombs next to the bot.

15) A method that takes two numbers and turns the first number and goes forward the second number.

16) A method that takes a String as a parameter, asks it of the human, and then returns the human's typed in response.

```
17) /**
    * turn a random direction and go a random distance from 1 – 5 steps
    * stopping if a key is hit and not accidentally hitting a wall. */
    public void randomSearch()
```

18) A method, remoteWallDetect, that makes an orb, teleports N spaces in the direction it is facing, counts the number of walls next to it, teleports back, and then returns the result found.

HW: BOTS AND METHODS #5

Write a bot with the following methods. Be sure to have a compute that tests all your code.

- 19) A method that puts the bot one space to the left, facing the same direction.
- 20) A method that returns the number of bombs next to the bot.
- 21) A method that takes two numbers and turns the first number and goes forward the second number.
- 22) A method that takes a String as a parameter, asks it of the human, and then returns the human's typed in response.
- 23) A method that follows the left-hand wall for N steps, where N is a passed parameter.
- 24) A method, remoteWallDetect, that makes an orb, teleports N spaces in the direction it is facing, counts the number of walls next to it, teleports back, and then returns the result found.

HW: AN INVESTIGATION OF OBJECTS AND INHERITANCE

This assignment is to reinforce understanding of objects, fields, and inheritance. We are going to make a mathematical function calculator that allows a user to evaluate different functions at different points.

1) First make a Function interface. It should have a method:

```
double eval( double x )
```

This method returns the value of the function at a given point x.

2) Make two new classes SqrtFunction and SqrFunction that implement Function. These two classes should return the square root and the square of x, respectively.

3) Make a test main method that makes a SqrtFunction and a SqrFunction and evaluates the function at different points. Ask the user for the points to square or take the square root of.

4) Now make a QuadraticFunction class which implements the Function interface. Make the QuadraticFunction have three fields (a, b, and c) that correspond to the coefficients to the function:

$$f(x) = a x^2 + b x + c$$

The QuadraticFunction should have a constructor that takes three doubles and sets a, b, and c to them.

5) Make a little test main method in QuadraticFunction that asks the user for three ints (which will be cast to doubles) and makes a QuadraticFunction using those ints. It should then ask the user where she wants to evaluate the function and print out the result.

6) Make an add() method that takes a QuadraticFunction and returns a QuadraticFunction which is the sum of the called on function and the passed function. (Remember that adding a quadratic function to another is just adding the coefficients.)

7) Make a new class ComposedFunction, that takes two Functions f and g and, for its eval, first calls eval on f, passing it x, and then calls eval on g, passing it the result of the eval on f. This class should implement Function as well.

8) Make a test program that allows the user to make two functions and then composes them, and then prints out the results. Check to make sure the composed function of sqrt and sqr always returns x when you put in x.