Class structure and syntax

- Which part goes where – specifications or implementation source code files.
  - Everything we type goes in the ClassName.java file for the class. There is no separate specification file. The java compiler reads the ".java" file and produces the ".class" file. The ".class" file has the same information in it that the original source file did, except that it has been converted to a binary format instead of the human-readable text format of the source file.
- Basic structure. Constructors, instance variables, etc.
  - Outer envelope is the public class ClassName { … } statement. Inside the curly brackets are the declarations of the instance variables, the comments and code for the constructors, and the comments and code for the methods. The instance variables can come before or after the constructors and methods. The constructors and methods can be in any order, although it is customary to put the constructors first.
- Reading and writing the syntax for different parts of a source code. For example, constructors, methods (queries, commands), variables, etc.
  - I've tried to address this in class by showing how you can easily draw an outline of the code before knowing exactly what it does. Find the curly brackets that surround the entire class. Then within that there may be some instance variable declarations. Also, there may be some constructors and some methods.
- General syntax. Specifically how the Java language can be interpreted in C terms.
  - The statements and basic syntax are very similar to C. The big difference is that C does not have the ability to link together methods with a block of data. In C you generally create a struct of some sort and pass around a pointer to it to the various methods that operate on it. In Java, you describe a class (containing instance variables) and then create an instance of it. This is similar to allocating memory for a struct. However, in Java, the methods of the class are automatically associated with the data object and so the two are more closely coupled and hopefully work better.
- Expressions vs statements. When to use which.
  - A statement describes a complete action for the computer to perform. An expression is an action that computes a value. One or more expressions are usually part of a complete statement. For example, the assignment statement `total=sum+delta;` computes the value of the expression "`sum+delta`" then stores that in variable "`total`".
- What is the common convention used when you have a really long expression that you'd like to split over two lines of code? Can you just do a soft return?
  - Yes, a plain return wherever you want the line to break is okay. There is no special "line continuation" character. Don't break a line in the middle of a literal string (between quotation marks) but otherwise it's just up to you as far as what you think will be the most readable.
- Your lectures of July 16 and 18 were good places to start breaking down the code. Need more repetition or iteration for some of us beginners.
  - Don't hesitate to ask me questions in class when I put code up on the screen. That's why I do examples. Even if the question is not about the specific topic of the code, ask anyway. Hopefully the answer will clear it up for you, and maybe for some other people who were fuzzy about the same issue. It's okay to be confused by all

this – there is a tremendous amount of new information. Even expert programmers are unclear about many areas of the systems they work with, but they have well developed information finding skills, both by reading and by asking, and they don't hesitate to seek out the information they need.

- Implementation
  - Develop a plan, outline the code and the approach, then write it one part at a time, testing as you go.

Constructors

- Constructors. How do they work? Why do we need them? Where do they go in the code and why?
  - I've talked about this in a couple of the lectures, so you might want to go back and reread some of them. The constructor runs when some piece of code uses the "new" operator to create a new object based on a known class. So when we say "new Acrobat()", the Acrobat constructor is called after the memory is allocated for the new object. The constructor's job is to initialize the instance variables and do any other preparation needed before the object is ready for use. They are defined within the body of the class, just like the methods of the class.

Methods

- How to decide what method to use in order to achieve the command.
  - You need to first determine what the overall structure of the program is, decide what classes you will be able to use from existing libraries and what classes will have to be written from scratch. You need to understand what is available, and you figure that out by reading whatever documentation seems relevant. For example, if you know that you will be using Rectangles from the UWCSE java library, then it would make sense to look at the documentation for that class and at least skim the method descriptions so that you know what a Rectangle can do. Once you know the overall structure and the capabilities of the existing code, then you take a pass at designing the new classes that your program will need. This requires that you think about what the responsibilities of those classes will be, and how you might implement them. This should get you well on the way to having a design to start coding from.
- How do you determine if a method needs parameters and what parameters it needs?
  - For existing methods you look it up in the documentation from Sun or whoever provided the library containing the class you are using. For new designs, you need to think about what the basic responsibilities of the class are, and how you would want the caller to be able to influence those responsibilities. Each responsibility is a method, and each influence is a parameter. So for example, our Acrobats clap, twirl, and count. Each of those is a method. We want the caller to be able to say how many claps and twirls, so both of those methods have an integer parameter to specify the number.
- The implementation of methods. Where to start, under what circumstances are local variables supposed to be used?
  - Where to start is by deciding on the responsibilities needed as described above. The influences from the caller are defined and provided to a method as parameters. Local variables are created if you need to store intermediate results while you are

implementing the responsibility. For example, you might keep a reference to a Road in a local variable while you decide whether or not the Road is important for future movement of your Car.

- Modularity, method calling within methods. How do you go about doing it – syntax, technique.
- The specifics of invoking methods, commands etc in the procedure.
  - Invoking a method just requires that you provide the name of the object for which the method is defined, followed by a ".", followed by the name of the method, followed by the parameters you are passing to the method enclosed in a set of parentheses. So to call the twirl method of a Dog object that implements the Acrobat interface (as I did in the lecture about interfaces) you just put `actC.twirl(7);` Assuming that you have properly initialized `actC` with a reference to a Dog object, this statement will transfer control to the twirl method. When the twirl method is complete, control will transfer back to the statement following `actC.twirl(7);`
- How to reference another method with a method.
- How to recall a method in a class.
- How to refer back to methods in other methods within the same class.
  - When you want to make a call from one method in a class to another method in the same class, you can just use the unqualified name of the other method. So for example, in example set 11, the DigitalLock constructor is:

```
public DigitalLock(int theCombination) {
    setCombination(theCombination);
    clearEntries();
  }
```

    The body of this constructor consists of two statements. The first one calls the `setCombination` method and the second one calls the `clearEntries` method.
- Also I am a bit confused about passing parameters in between methods.
- How to use the method parameters when actually writing the methods.
  - Generally when you are creating a new method you first decide what the parameters should be before you start coding. The idea is that the interface to the method (the things that a caller can tell the method) should be well understood and relevant to the caller. Then you write down the method header (the statement that gives the name of the method and its parameters). The names and types of the parameters are specified in the method header. Then you use the parameters in the body of the method using the names that you provided in the header. For example, the enter method in class DigitalLock (example 11) looks like this:

```
public void enter (int digit) {
    if (digit >= 0 && digit <= 9 ) {
      entered1 = entered2;
      entered2 = entered3;
      entered3 = digit;
      if (isValidCombination(entered1, entered2, entered3)) {
        open = true;
      }
    }
  }
```

    There is one parameter "digit" of type int. The caller supplies a value for the parameter each time it calls the enter method. The method refers to the parameter using the name "digit".

- The return function
  - The return statement is used to indicate that the method is done and control should return to the caller. If the method returns a value, then the value must be specified in the return statement. If the method does not return a value, then just the word return is used. If the method does not return a value, then just reaching the end of the method body is equivalent to executing a return statement as the last statement of the method. (In other words, if the method just ends without a return statement, the compiler assumes that you have nothing more to do and returns to the caller.)
- Difference between "argument" and "parameter"
  - I don't usually make a big distinction between the two words. The actual difference is that a "parameter" is the formal definition of what is required by a method (as described in the method header) whereas an "argument" is the value that is actually provided at run time (in the method call). So for example with the `enter` method given above, the method header is `public void enter (int digit).` "`digit`" is the parameter (or "formal parameter") for this method. When the method is called using a statement like `lock.enter(k)`, then k is the argument (or "actual parameter") whose value is provided to the method and can be accessed in the body of the method using the name "`digit`".
- The functions of things like void, stati
  c, public static void main(String arg[])
- Why do you have to have public static void main(String[] arg) no matter what?
  - Because the program has to start somewhere. The designers of Java chose "main" as the name to use. It could have been anything (start, entrypoint, run, ...) but main is what they picked. It has to be public because it wouldn't make sense to have an invisible starting point. It has to be static because static members are available after the class definition is read into memory and before creating any objects. This way the java virtual machine can just read your class in memory and jump right to the beginning. The String array is the required argument so that java can pass the user command line arguments to your program. The *name* of the argument can be anything you want, but it must refer to a String[].
- I am a bit hazy about why we always have to create a class, even if it is to just create a method.
  - Methods are the implementation of responsibilities. Without a class there would not be any logical place to put the implementation. Note that you can get away with a minimum of one class – one gigantic class that contains all the methods you need and all the variables. This can be done, but it rapidly gets too big and too complicated to understand.
- Construction of a method, a constructor.
- How the syntax of writing a method works. What is the return returning to? When would void be used?
  - The method header defines the access (public or private), the type of value the method returns (int, String, double, Road, void, etc), the name of the method (twirl, count, enter, etc), and the list of parameters and their types. The header is followed by the body of the method enclosed in curly brackets. When a method is called (eg, `lock.enter(k)` calls the `enter` method) the statements in the body are executed one after another until the end of the body is reached or a return statement is executed. The return tells java to return control to the statement following the call, in this case

the statement that follows `lock.enter(k)`. Void is used to indicate that a method does not return a value. The method executes, and it does something useful, but it does not return a value to the caller.

- If you are in some class say Card class can you call methods of that class without having something like player.addToDeck() or this.addToDeck() from another method in that same class.
  - Yes, you can call the methods of a class from the other methods of the class. You can either use the keyword this (for example `this.addToDeck(...)`) or you can leave it off and just give the unqualified name of the method (for example, `addToDeck(...)`).
- When given a reference to an object that is passed in from somewhere, for example with: `public class Example(Director director, Car car) {}`, I don't understand exactly what is being passed in. What use is passing in an object? Do I get access to its functions, variables, what?
  - Yes you get access to its public methods and public instance variables (if any). The reason for passing a reference to an object is so that you can ask it questions or ask it to do something.
- The advance methods
  - The advance method in Car uses the moveBy method to move the rectangle representing a car a little bit every time it is called. The advance method in Sun uses the moveBy method to move the circle representing the sun a little bit every time it is called.

Variables and References

- When you declare a variable, what is its default value? Is it null, zero, or just nothing? Does it differ based on variable type?
  - Don't depend on the default values. Instance variables default to zero or null depending on their type, but local variables do not get automatically set to anything and so they have no default values. It's easier to just initialize everything yourself.
- Are we going to need to know about casting variables from one type to another in the tests?
  - You should be familiar with the idea and understand it when you read it.
- Local variables referencing other objects that go to another class method. Stack->object->different class.
  - A local variable can contain a reference to an object. For example, `Dog z = new Dog("J");` creates a new Dog object and stores a reference to it in variable z. Then if we want to call a Dog method, we can write `z.eat(10);`
- How to reference to a class.
  - If you want to create a new object based on a class definition, you use the name of the class with the new operator to call the constructor, as in the example just above this paragraph.
- When to put values in parentheses and when not to. For example, public Road(int x, int y), public int getActionCount(), Director bob = new Director()
  - The definition of parameters goes in parentheses as part of the constructor or method header. There are no values between the parentheses in the call to the Director constructor because the constructor was not defined with any parameters. Therefore, none are supplied by this caller.

- Why do we write something like this: `Counter counter = new Counter();` instead of: `counter = new Counter();` Page 124 talks about this question, but I don't quite understand it yet.
    - You specify the type of a variable (eg, Counter) if you are declaring it. For local variables it is quite common to declare the variable and initialize it in one statement as done here. However after the variable has been declared, you can use it without stating its type. So the second time you use the local variable counter, you would not name the type again. Similarly, if counter is an instance variable, you need to state the type when it is declared, but then you don't need to state the type when it is used because Java already knows about it.
- Why is it necessary to create so many variables all referring to similar things and then setting them equal to one another. It gets very, very confusing. Ie, Road – getCurrentRoad, roadToCheck, CarIsOnRoad(road)
    - I agree it gets confusing sometimes. The reason for doing it is that each method wants to keep track of the things it is working on. So you might need little spots to keep track of "the road I'm on", "the next road in the list of all roads", "the road I'm currently looking at". At times they may all point to the same road, but they are being kept for different reasons.
- Creating an instance variable for a parameter??
    - The reason you copy a parameter to an instance variable is so that the methods can use the value later. The parameter values of a constructor are only available while the body of the constructor is executing. If you need to use the value later on in one of the methods, you need to copy it to an instance variable so that it is still available after the constructor is done.
- "this" is it a reference to the initial object or a duplicate?
    - It refers to the current object itself, not a copy.
- Scope, the scope of different variables defined in different places.
    - Private instance variables are visible to any method defined for the object. Public instance variables (generally a bad idea) are visible to any method that has a reference to the object. Local variables are visible within the block where they are declared. Parameter variables are visible within the method for which they are declared. Take another look at the lecture about this. Ask me or the TAs specific questions.

Access Levels

- Access levels. I am fluent in C so I am unfamiliar with the notion of access levels. What exactly does it mean when you say a public variable is accessible to anyone? Do you mean that other people from other computers can hack my code?
    - No. "Public" means that a method or variable for object A is visible to any object B in the same program that has a reference to object A. "Private" means that the method or variable is not visible to other objects unless object B is the same class as object A. "Visible" means that your code can use the method or variable. In ex11 class `DigitalLock`, the method `enter` is public and so the Cracker object can call `enter`. The method `clearEntries` is private, and so the Cracker object cannot call `clearEntries`.
- When to use private and when to use public.

- Visibility specifiers – when exactly should public or private be used?
    - Use public for the methods that you want other objects to be able to access. Use private for the instance variables and methods that you do not want other objects to use.

Loops

- When would you use a for loop and when would you use a while loop?
    - Use a "for" loop when you want have a specific number of times that you want to go through the loop. You might want to exit the loop early, but with a for loop you generally have a maximum number of times that you want to execute the body of the loop and you know before hand what that number is. Use a "while" loop when you don't know how many times you want to run through the loop. You just want to keep going until something happens.
- How to write good loops. I can read them, but writing them is difficult.
    - Practice, practice, practice. They really are pretty simple once you get over learning the syntax. It will get easier after you have a better idea of how to express the condition that controls when you should stop looping.
- The body of while loops. Examples of while loops in context of classes and other methods etc.
    - The body of a while loop is just a set of statements that get executed over and over again. The body can contain anything you want to do. As with all code, the while loop must be contained within some executable block (the body of a constructor or method) but otherwise there is nothing special about it.
- In the digital lock / cracker code, why does the for loop stop as soon as the correct combination is determined? The condition of the for loop seems still applicable after the combo is returned.
    - The for loop stops because the return statement says "we are done in this method" and control passes back to the calling method. The return statement causes the pick method to abandon all future work even though the loops have not run all the way through. It's the same as if you were looking through all the possible combinations yourself in an orderly fashion but found a winner part way through. You'd stop right there and return the answer to the friend who asked for the combination even though you hadn't looked at every possible combination.
- Loops and embedded loops. How to write, where to begin.
    - Think about what is controlling the loop. Do you need to loop a certain number of times? If so, a for loop is probably the best approach. Do you need to loop until something happens (like the Car goes off the end of the Road)? Then you probably need a while loop. Embedded loops are just a way of saying that for each pass through the outer loop, you want to do some smaller task several times. This is similar to saying something like "for houses 1 through 10, do the following: for each window on the house, check that the window is locked.
- For loops. How does the rule about "one allowed method" work?
    - I don't know what this rule is. There are no limits that I can think of that would be described this way. Ask me again when we can discuss it.
- How to convert for loops to while loops
    - A for loop is written like this: for (initialize; condition; update) {body}. The equivalent while loop is written like this: initialize; while (condition) {body;update}.

- For example, assuming first and last are already defined, this *for* loop:

```
int sum = 0;
for (int i=first; i<=last; i++) {
    sum = sum + i;
}
```

  could be rewritten as this *while* loop:

```
sum = 0;
int i = first;
while (i<=last) {
    sum = sum + i;
    i++;
}
```

If, switch

- What's switch?
  - It's another statement used to control the flow of execution in the program. It can be used to select one of several blocks of code to execute, depending on the value of an integer variable. I will not ask any questions about it on any exam.
- When would one use an if-else method instead of nested if method?
  - If-else executes one or the other possible blocks of code. Nested-if executes the code block if the first condition *and* the second condition are both true. These are *not* the same.

```
double value;
double rate = 1.03;
if (sum == 0) {
    value = 0;
} else {
    value = sum * rate;
}

if (sum != 0) {
    if (rate != 0) {
        value = sum * rate;
    }
}
```

- Boolean
- The whole Boolean-logic and how that exactly works out. I don't quite get the logic table in the book and what we use it for.
- Boolean statements like the ones on homework 3. I got them right, but I was not comfortable with them.
  - Boolean logic is the same as our normal logic. We use boolean variables to hold the values of true and false, and we can combine them in expressions that are also true or false. We can use these expressions to control the flow of the program. So for example, in English we might say:

    We know when it is summer time. We know when it is raining. If it is summer and it is raining, then play video games.

Or, in Java:

```
boolean isSummer = true;
boolean isRaining = false;
if (isSummer && isRaining) {
    videoGame.play();
}
```

Comments and Documentation

- How to turn comments for a constructor into code that fulfills all of the @param statements.
  - The information in the @param tag describes what the method will use the given parameter for. So the code should just implement whatever use has been described for the parameter. The @param tag is a contract between the author of the method and the user of the method (another programmer) describing how the parameter will be used in the body of the method.
- The use of "@" in comments.
  - The "@" sign is used to signal to the javadoc program that a special little piece of comment is coming up and so different formatting should be applied. Javadoc is just another program. It reads your source code and filters out the special comments and then produces nice web pages containing the formatted information. It recognizes the "@" as identifying a special section.

Java libraries

- I'm having trouble finding methods from the java library, and even when I find the method, it is not easy to know how to use that method because the explanation isn't enough for beginners. Basically, java library is not explained well enough for beginners.
  - True, the descriptions are not intended to be tutorials in general. However, all the information needed to actually use the library classes is usually there. I have mentioned the Java tutorial in several of the lectures, and that is a good place to get more extended descriptions of the basic Java classes and how to use them.
- Difference between ArrayList and arrays, and how each of them work.
  - I'll talk about this next week. In the meantime: arrays are a way to keep a list of objects or primitive values that you can look at one by one. ArrayList is a class that manages an array for you and adds some fancy capabilities.

Exams

- Just talk about things that will be on the midterm, that should be fine.
- What are the specific codes that we must know by now if any?
  - That's what this review is for. Also, go back over the lectures.
- Drawing diagrams. Do we have to know how to draw diagrams in the mid-term.
  - No.

Miscellaneous

- Too abstract, and too complicated. Ie, I'm still not sure about relationship between director and classes.
  - It's confusing at first. If you work out the examples I give in class and the TAs give in section, the patterns will start to be visible.
- Your explanation could be understood by students who have experienced programming, but not by beginners. Also there is a huge gap between homework and lecture content-homework is a lot harder.
  - Ask questions in class or in section. The students in the class have a wide range of backgrounds but we are trying to provide a good basic introduction for everyone. If we skim over something too quickly, ask for an explanation. If you're confused about something it's very likely that other people are too.
- What are stacks? The explanation made sense, but I couldn't find any way that it related to actually putting one in.
  - A stack is a way of organizing data values so that only the most recently stored value (or set of values) is available at one time. Stacks are used in many different ways in programming. The stack I have mentioned in class is one that is maintained by Java for you and you don't need to do anything to set it up. In future programs for more complicated applications you may write code that manages a stack, but you do not need to do that for this class.
- Ways of approaching problems where comments are provided and one is required to fabricate code from basically nothing.
  - Remember the structure of a class, with instance variables, constructors, and methods. Given the comments, you need to decide how to accomplish the tasks described (ie, what methods are needed), decide what data your methods will need (ie, what instance variables) and decide how to initialize it all (ie, what constructors are needed) and then write it all down.
- How does it really work? Ie, how does the computer understand it?
  - When you say "compile all" in drJava, your source code (the Car.java file that you wrote) is compiled into a class file (Car.class). The class file contains the same information, but in a format that is more compact. When you say "java Director" the java program opens the Director.class file and starts reading it. Each instruction that you wrote is there, and so it performs the first instruction, then it reads the next one and performs that, and so on until the program ends.
- How do you make a dynamic system, meaning the user can input into it?
  - The arguments that you enter on the command line when you run the program are passed to the main method in the String[] arg. Take a look at the main method of class Mathie in example set 9.
- How do you make a small car drive down a predetermined road accelerating at each crosswalk? :-)
  - :-)
- HW4 written assignment #2, #4
  - Talk to the TAs or me with specific questions.