Exercise 1: What are joinpoints?
Consider the following example ClassLiLa program.

```
(run "class Point extends Object
    field x
    field y
    method initialize(ix,iy) begin
        set x = ix; set y = iy
    end
    method moveByX(dx) set x = +(x,dx)
    method moveByY(dy) set y = +(y,dy)
    method moveBy(dx,dy) begin
        send self moveByX(dx); send self moveByY(dy)
    end

    let pt = new Point(1,2)
    in begin
        send pt moveBy(1,1);
        send pt moveBy(1,1);
        pt
    end")
```

Questions:
1. How many joinpoints are there in this program?
2. What do the joinpoints representation for these joinpoints look like? (Draw a picture of some of the joinpoint representations on a piece of paper.)
3. How would you define a scheme procedure `same-joinpoint?` which takes two joinpoint representations as arguments and returns whether they represent the same joinpoint in the program.

Exercise 2: What are joinpoints?
See slide 25. Find out what language feature(s) are supported by the mysterious "args" field in the joinpoint representation and construct an example using this/these feature(s).

Exercise 3: More pointcuts and joinpoints
See slides 41-44. These slides suggest a number of ways in which AspectLiLa’s pointcut language could be extended to be more interesting and more like AspectJ’s. Think about these features and whether or not adding them would be a good idea:
- What is their semantics? Are their perhaps options to explore? How and where does the interpreter need to be changed to support them? How about the “weaver” implementation?
- What is the purpose of these features? Can you think of examples that show what they are good for?
- Are their reasons why you might not want to add such features?
**Exercise 4: Basic Understanding of “The Weaver”**

Consider again the program from Exercise 1 and the original (most basic) AspectLiLa language. Consider slide 97. For a given method in a program (e.g. the moveBy method from the example program).

1. How many times does the code above the dotted line get executed for this method.
2. How many times does the code below the dotted line get executed for this method.

**Exercise 5: What are joinpoint-shadows**

Consider again the program from Exercise 1 and the original (most basic) AspectLiLa language. Questions:

1. How many joinpoint-shadows are there in this program?
2. What do the representations for these joinpoint-shadows look like?
3. How do these shadows relate to the joinpoints from question 1?
4. Does solving exercise 3 affect the answers to these questions? If so how? If not, why not?