CPSC 411, Spring 2005 – Quiz 1

Name: _____

Q1:	Q4:
Q2:	Q5:
Q3:	Q6:
TOTAL:	

Problem 1 [20 points]

Draw a diagram showing the standard structure of a compiler. Then take each of the following class names, and write it next to the diagram element that it helps to implement.

`IfExpr` `BindNames` `MJScanner` `MJParser`

`SymbolTable` `CheckTypes` `GenerateCode` `ClassDecl`

Problem 2 [15 points]

The following method definition comes from one particular class in the MiniJava compiler.

```
public void visitLocalVarDecl(LocalVarDecl ld) {
    super.visitLocalVarDecl(ld);
    currentScope.put(ld.nameAsString(), ld);
}
```

Which class?

What happens if we swap the two lines of the method body? If you believe this would change the semantics of the language then show a concrete example of MiniJava code that would be affected.

Problem 3 [15 points]

Sally is writing an Asteroids game using AspectJ. One part of her design involves a `Ship` class, that has methods called `fire()`, `turn()` and `thrusters()`.Ships also have a `getFuel()` method that returns an `int` saying the level of fuel in the tank.

Sally needs to implement functionality that prevents the above methods from actually doing anything if the fuel in the tank has reached empty. The idea is that a ship cannot fire, turn or use thrusters if it is out of fuel.

Show the AspectJ code Sally could write to do this:

Problem 4 [20 points]

In this problem you will show the abstract syntax tree (AST) and the AST class hierarchy for the pointcut sub-language of your AspectJ extension to the Mini-Java compiler. Consider the part of the pointcut language described by the following grammar:

```
<pointcut> ::= call(<method signature>)
               execution(<method signature>)
               get(<field signature>)
               set(<field signature>)
               within(<type pattern>)
               <pointcut> && <pointcut>
               <pointcut> || <pointcut>
               ! <pointcut>
```

Show the AST class hierarchy you would use to support the above grammar. Include enough classes in your hierarchy so that you get all the way down to strings or other primitive tokens.

Now show the AST structure that would result from parsing the following pointcut. Use a notation like the one that the `toLongString` method returns.

```
( call(void Point.setX(int)) || call(void Point.setY(int)) )
  && !within(Shape+)
```

Problem 5 [20 Points]

The following code fragment contains shadows for several AspectJ dynamic join points.

```
void foo(int x) {
    this.x = bar(x);
}
```

How many shadows of AspectJ dynamic join points does it contain, and which ones?

Now re-write the code so that the shadows are in a canonical form. You can use any canonical form you want, but it must be consistent in that all the shadows must be re-written into a single canonical form. Explain the canonical form you are using, and show all the re-written code.

Problem 6 [10 points]

Consider the following simplification of an AspectJ program.

```
class Client {
   Service s = ...;

   void itAllStartsHere() {
      s.call();
   }
}
class Service {
    Result call(Request request) {
        return lookupWorker(request).doWork(request);
    }
    Worker lookupWorker(Request request) { return ...; }
}
class Worker {
    Result doWork(Request request) {
        ...
    }
}

aspect Billing {

   before(Client c, Worker w): ____________________________________ {
      billForWork(c, w);
   }
   void billForWork(Client c, Worker w) {
      ...does the actual billing of the client based on the worker...
   }
}
```

Show the pointcut, using cflow, that fills in the blank, so that when `itAllStartsHere` runs, and results in a call to `Work`, the `Client` that starts running, and the actual `Worker` doing the work are passed to `billForWork`.