# Weekly Report

**Team Dec 15-12: PaniniJ Eclipse Plugin**

***Week 3:*** *Feb. 2nd - Feb. 9th*

---

| | | |
|---|---|---|
| **Advisor** | Dr. Rajan | |
| **Client** | Dr. Rajan | |
| **Team Members** | Dalton Mills | *Webmaster* |
| | David Johnston | *Team Lead* |
| | Kristin Clemens | *Communication Lead* |
| | Trey Erenberger | *Key Concept Holder* |

---

## Weekly Summary

---

**Project Goals:** The goal of this project is to make a system which makes the functionality and safety guarantees of capsules accessible to the programmer. As the project was originally described, our solution would come in the form of an Eclipse plugin for the PaniniJ language. However, after some discussions with Dr. Rajan and some experimentation, we have decided that our project deliverable should instead come in a different form, though it will still be designed to meet this larger goal.

**Changed Project Deliverable:** We started this week with several potential project routes. We narrowed it down to just one through research, discussion with our advisor, and some prototyping. We now intend to use Java 8's enhanced annotation features to automatically generate `.java` source code for capsules from Java class definitions.

One reason for the shift away from the previously proposed Xtext/Xtend solution is the relative immaturity and volatility of the Xtext/Xtend framework.

Furthermore, an annotation-based capsule framework has many potential advantages over the existing PaniniJ compiler. For example, our solution should be an easier way for a user to build systems with Capsules, because our proposed system will be built within the Java language and should seamlessly integrate into standard Java build environments (i.e. both the javac and Eclipse toolchains). Integrating PaniniJ capsules into larger systems is much less straightforward. An annotation solution may therefore be more likely to be used by Java programmers.

The end product will be a JAR file that will be included to projects in order to allow for the use of @Capsule annotations. Our JAR will hook into the standard oracle java compiler to produce extended thread-safe versions of the classes annotated with @Capsule.

**Current and Future Work:** Proof of concept work on the initial compiler hook (i.e. annotation processor) was very successful. A very basic working prototype is now able to automatically generate basic capsule functionality from a template class.

In the future, we expect that our project will become focused on integrating the safety-check algorithms developed by Dr. Rajan's research lab into a user-friendly annotation-based system. It will take time to find out which of these algorithms can be integrated into the annotation solution.

# Technical Progress

This week we explored several options for implementing our project's goals. David produced a proof of concept for the annotation approach that augments java classes with preliminary capsule functionality. This prototype demonstrates that it is possible to hook into the compilation process and produce intermediary java files that will then be compiled and usable elsewhere in a Java project.

**Example:** The first image below shows a `.java` file which includes a class annotated with `@Capsule`. The subsequent image shows the `.java` file generated from the original `@Capsule` class. This automatic source generation process is performed by an `@Capsule` annotation processor which we have implemented.

```java
package me.dwtj.capsules;

@Capsule
public class HelloWorld
{
    public void helloWorld() {
        System.out.println("Hello, world.");
    }

    public void hello(String name) {
        System.out.println("Hello, " + name + ".");
    }

    public void helloWorldRepeat(int repetitions) {
        for (int i = 0; i < repetitions; i++) {
            System.out.println("Hello, world.");
            try {
                Thread.sleep(1000);
            } catch (InterruptedException ex) {
                // TODO ???
            }
        }
    }
}

```

```java
package me.dwtj.capsules;

import java.util.concurrent.Callable;
import java.util.concurrent.Future;
import java.util.concurrent.FutureTask;
import java.util.concurrent.LinkedBlockingQueue;



/**
 * This capsule was auto-generated from `me.dwtj.capsules.HelloWorld`.
 */

public class HelloWorldCapsule extends HelloWorld implements Runnable
{
    LinkedBlockingQueue<Runnable> queue = new LinkedBlockingQueue<Runnable>();
    Thread thread;


    public void start()
    {
        thread = new Thread(this);
        thread.start();
    }


    public void run()
    {
        while(true) {
            try {
                queue.take().run();
            } catch (InterruptedException ex) {
                // TODO?
            }
        }
    }


    // TODO: build constructor

    public Future<Void> helloWorldProc()
    {
        FutureTask<Void> f = new FutureTask(
            new Callable<Void>() {
                public Void call() {
                    helloWorld(); return null;
                }
            }
        );

        try {
            queue.put(f);
        } catch (InterruptedException ex) {
            // TODO?
        }
        return f;
    }


    public Future<Void> helloProc(java.lang.String name)
    {
        FutureTask<Void> f = new FutureTask(
            new Callable<Void>() {
                public Void call() {
                    hello(name); return null;
                }
            }
        );

        try {
```

# Meetings

## Weekly Administrative Meeting

**Members Present**: All
**Additional Participants:** N/A
**Date & Location:** Tuesday 3 of February; Molecular Biology 1414
**Minutes**:

- Discussed prototyping and proof of concept tasks for coming week
- Disseminated research findings
- Agreement to set hard deadlines for project sprints
- Pluggable Type checker research needed
- We need more documentation

---

## Bi-Weekly Advisor Meeting

**Members Present**: All
**Additional Participants:** Dr. Rajan
**Date & Location:** Friday 6 of February; Atanasoff 101
**Minutes**:

- David explained his initial proof of concept work to group and Dr. Rajan.
- Dr. Rajan and group decided that we should shift in direction of annotation processor and pluggable type checker.

---

## Weekly Collaboration Meeting

**Date & Location:** Sunday 8 of February; Google Hangouts
**Members Present**: All
**Additional Participants:**
**Minutes**:

- Set up of David's Test project
    - Installing ant, maven
    - How to drop the product (capsule-generator) into a project
- David presents Test project for group
- Discussion of constructor implementations
    - Factory for each thread scheme

    - Factory.make() overloaded with the different signatures of 'template class' constructors
- Discussion of end product deliverable
    - How to handle dependencies for generator class jar
    - preventing jar bloat
    - not tied to certain build scheme (maven, ant, etc.)
- Future tasks: -Refactoring test project to cultivate best practices (David/Trey)
    - Research into basic validation for capsules, pluggable type checker (Kristin/Dalton)
    - Test code best practices exploration (Kristin)
    - Further development of SD website (Dalton)
- Concluded

---

## Individual Hourly Contributions

| | |
|---|---|
| **Trey Erenberger** | 6.5 Hours |
| **David Johnston** | 17 Hours |
| **Kristin Clemens** | 4 Hours |
| **Dalton Mills** | 6 Hours |

## Tentative Plans for Week 3

For the next week we have narrowed our body of research to topics within this annotation approach. We will continue to do proof of concept work with the capsule-generator project that David implemented this week. We will also be looking to define a specific set of features for our deliverable and then validate those during our meeting with Dr. Rajan. For next week we have set out specific goals for each member of the team. Dalton is going to look into pluggable type checkers. His goal is to get just one type check working in a branch of the current project. Trey is working on adding constructors to the generated java source files.