

# **@PaniniJ: Generating Capsule Systems from Annotated Java**

Dec15-12: Trey Erenberger, Dalton Mills, and David Johnston

# Overview

- Project Foundations: Capsule Oriented Programming
- Goals of @PaniniJ
- How It Works
- Usability and Maintainability Improvements

# Project Goal

Make Capsule Oriented Programming more  
accessible to Java programmers

# Concurrent Programming

Concurrent Programming in Java is *Hard*

# Capsule Oriented Programming

One can think of it like a design pattern.

- *Capsule*: Like an object with a thread inside.
- *System of Capsules*: A collection of capsules sending *requests* to each other.
- Write sequential code; asynchronous code.
- `String s = fooCapsule.bar("Hello, world!");`

# PaniniJ: The Existing Solution

- PaniniJ is a capsule-oriented language.
- Language is similar to Java.
- Modified Java compiler (panc) compiles PaniniJ code.
- Auto-generate boilerplate concurrent capsule code.
- Correct by construction concurrency.

```

43
44 capsule Console () implements Stream { //Capsule declaration
45     void write(String s) { //Capsule procedure
46         System.out.println(s);
47     }
48 }
49
50 capsule Greeter (Stream s) { //Requires an instance of Stream to work
51     String message = "Hello World!"; // State declaration
52     void greet(){ //Capsule procedure
53         s.write("Panini: " + message); //Inter-capsule procedure call
54         long time = System.currentTimeMillis();
55         s.write("Time is now: " + time);
56     }
57 }
58
59 capsule HelloWorld() {
60     design { //Design declaration
61         Console c; //Capsule instance declaration
62         Greeter g; //Another capsule instance declaration
63         a(c): //Wiring connecting capsule instance a to c

```

## The Problem With PaniniJ: Few Development Tools

# Initial Project Specification

Build an Eclipse Plugin for PaniniJ to:

- Fix red squiggles
- Provide useful compilation errors & warnings
- Enable code completion & IDE features



# Eclipse Plugin

**Pro:** It would work.

**Con:** IDE lock in

**Con:** Maintainability hurdles

**Con:** Usability hurdles

# Client Goals

Capsule Oriented Programming shall be:

- *More usable* by Java programmers.
- *More compatible* with existing Java tools.
- *Less complex* to use within Java projects.

# Alternative: Compiler Plugin

**Pro:** Core Java Feature (Annotation Processor)

**Pro:** IDE Independent

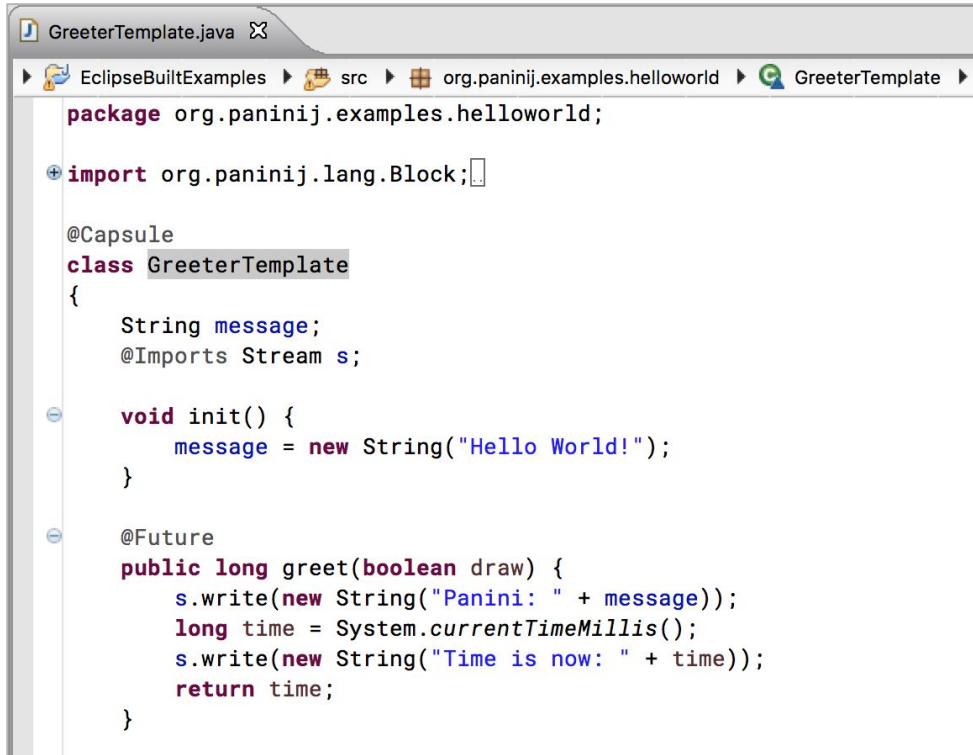
**Con:** Required Reimplementation Functionality of PaniniJ

# Deciding Factor: Use Standard Tools

Standard compiler plugin strategy met all 3 goals.

*Bring panini to tools rather than tools to panini.*

# @PaniniJ: Our Solution



```
GreeterTemplate.java
EclipseBuiltExamples > src > org.paninij.examples.helloworld > GreeterTemplate
package org.paninij.examples.helloworld;

import org.paninij.lang.Block;

@Capsule
class GreeterTemplate
{
    String message;
    @Imports Stream s;

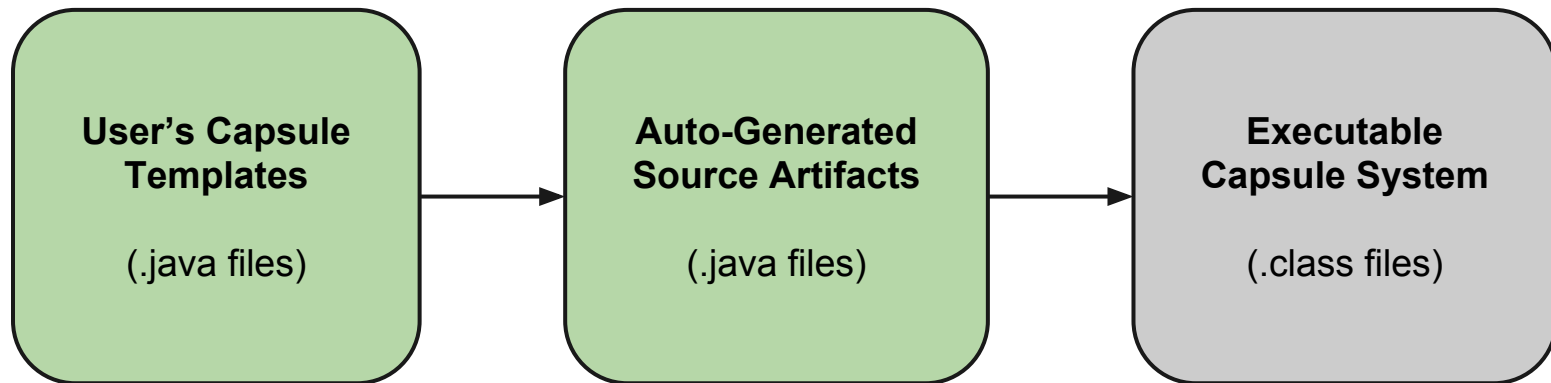
    void init() {
        message = new String("Hello World!");
    }

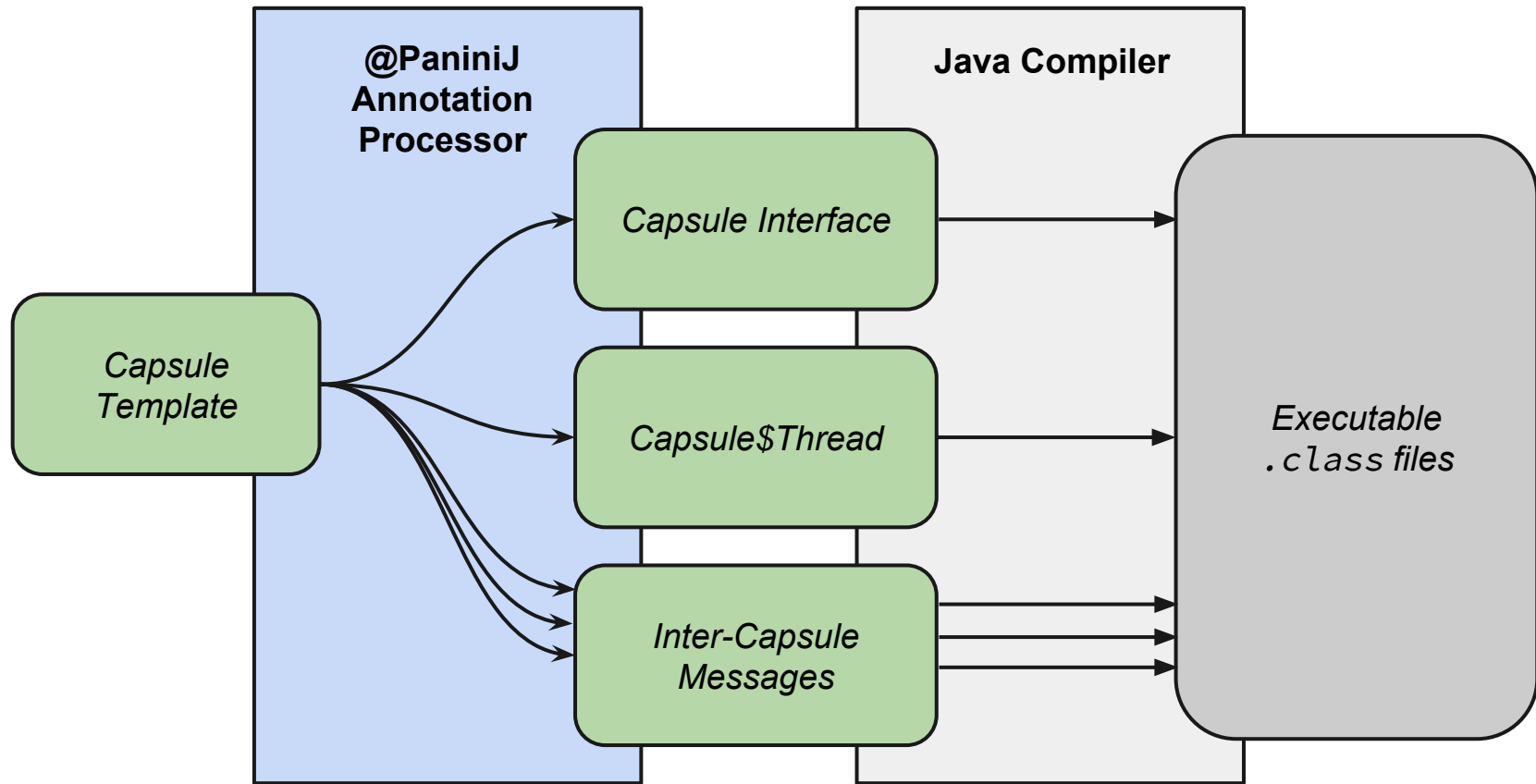
    @Future
    public long greet(boolean draw) {
        s.write(new String("Panini: " + message));
        long time = System.currentTimeMillis();
        s.write(new String("Time is now: " + time));
        return time;
    }
}
```

- The user defines a set of *capsule templates* as Java classes.
- Each template describes properties and behavior of the desired capsule.
- @PaniniJ generates the concurrent Java code required for such a capsule.

# Code Generation

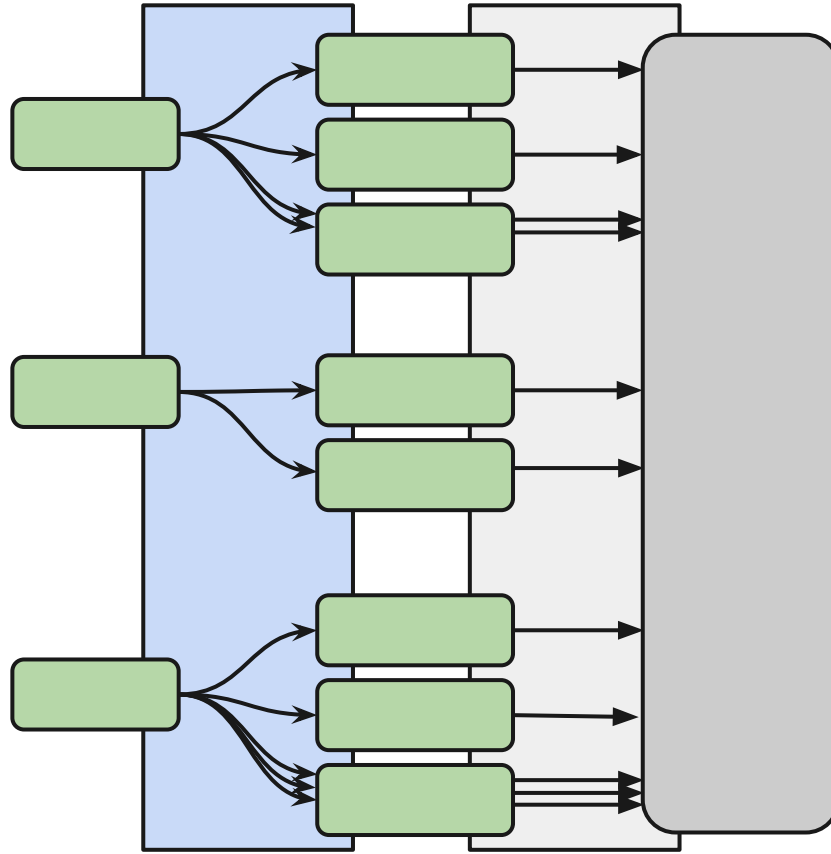
# Annotation Processing Pipeline





**Many Artifacts are Generated From One Capsule Template**





**Capsule System Includes Many Artifacts From Many Capsule Templates**

```

package org.paninij.examples.pi;

import java.util.Random;

/**
 * Each Worker capsule computes a fraction of the total number of samples.
 */
@Capsule
public class WorkerTemplate
{
    Random prng;

    public void init() {
        this.prng = new Random();
    }

    public Number compute(double num) {
        Number _circleCount = new Number();
        for (double j = 0; j < num; j++) {
            double x = this.prng.nextDouble();
            double y = this.prng.nextDouble();
            if ((x * x + y * y) < 1) _circleCount.incr();
        }
        return _circleCount;
    }
}

```

## User's Capsule Template

```
package org.paninij.examples.pi;

import java.util.Random;

/**
 * Each Worker capsule computes a fraction of the total number of samples.
 */
@Capsule
public class WorkerTemplate
{
    Random prng;

    public void init() {
        this.prng = new Random();
    }

    public Number compute(double num) {
        Number _circleCount = new Number();
        for (double j = 0; j < num; j++) {
            double x = this.prng.nextDouble();
            double y = this.prng.nextDouble();
            if ((x * x + y * y) < 1) _circleCount.incr();
        }
        return _circleCount;
    }
}
```

*Capsule Interface*

*Capsule\$Thread*

*Inter-Capsule  
Messages*

## User's Capsule Template

```
1 package org.paninij.examples.pi;
2
3 import javax.annotation.Generated;
4 import java.util.concurrent.Future;
5 import org.paninij.lang.CapsuleInterface;
6 import org.paninij.runtime.futures.org_paninij_examples_pi_Number$Future$dbl;
7 import java.lang.Object;
8 import java.util.Random;
9 import org.paninij.runtime.Panini$Capsule;
10 import org.paninij.examples.pi.Number;
11 import org.paninij.runtime.Panini$Capsule$Root;
12
13 @Generated(value = "org.paninij.proc.factory.CapsuleInterfaceFactory", date = "2015-12-08T19:05+0000")
14 @SuppressWarnings("unused")
15 @CapsuleInterface
16 public interface Worker extends Panini$Capsule
17 {
18     public org.paninij.examples.pi.Number compute(double num);
19 }
20
21
22
```

## Generated Capsule Interface

```
package org.paninij.examples.pi;

import java.util.Random;

/**
 * Each Worker capsule computes a fraction of the total number of samples.
 */
@Capsule
public class WorkerTemplate
{
    Random prng;

    public void init() {
        this.prng = new Random();
    }

    public Number compute(double num) {
        Number _circleCount = new Number();
        for (double j = 0; j < num; j++) {
            double x = this.prng.nextDouble();
            double y = this.prng.nextDouble();
            if ((x * x + y * y) < 1) _circleCount.incr();
        }
        return _circleCount;
    }
}
```

*Capsule Interface*

*Capsule\$Thread*

*Inter-Capsule  
Messages*

## User's Capsule Template

```

19
20 @Generated(value = "org.paninij.proc.factory.CapsuleThreadFactory", date = "2015-12-08T19:05+0000")
21 @SuppressWarnings("unused")
22 @CapsuleThread
23 public class Worker$Thread extends Capsule$Thread implements Worker
24 {
25     private org.paninij.examples.pi.WorkerTemplate panini$encapsulated = new org.paninij.examples.pi
26     public static final int panini$proc$compute$double = 0;
27
28     @Override
29     public org.paninij.examples.pi.Number compute(double num)
30     {
31         org_paninij_examples_pi_Number$Future$dbl panini$message = null;
32         panini$message = new org_paninij_examples_pi_Number$Future$dbl(panini$proc$compute$double, r
33         ;
34         panini$push(panini$message);
35         return panini$message.get();
36     }
37
38     @Override
39     protected void panini$initState() {
40         panini$encapsulated.init();
41     }
42

```

## Generated Multithreaded Wrapper

```

package org.paninij.examples.pi;

import java.util.Random;

/**
 * Each Worker capsule computes a fraction of the total number of samples.
 */
@Capsule
public class WorkerTemplate
{
    Random prng;

    public void init() {
        this.prng = new Random();
    }

    public Number compute(double num) {
        Number _circleCount = new Number();
        for (double j = 0; j < num; j++) {
            double x = this.prng.nextDouble();
            double y = this.prng.nextDouble();
            if ((x * x + y * y) < 1) _circleCount.incr();
        }
        return _circleCount;
    }
}

```

*Capsule Interface*

*Capsule\$Thread*

*Inter-Capsule  
Messages*

## User's Capsule Template



```

13 @Generated(value = "org.paninij.proc.factory.FutureMessageFactory", date = "2015-12-08T19:05+0000")
14 @SuppressWarnings("all")
15 public class org_paninij_examples_pi_Number$Future$dbl implements Panini$Message, Panini$Future<org
16 {
17     public final int panini$procID;
18     private org.paninij.examples.pi.Number panini$result = null;
19     protected boolean panini$isResolved = false;
20
21     public double panini$arg0;
22
23     public org_paninij_examples_pi_Number$Future$dbl(int procID, double arg0){}
24
25     public int panini$msgID() {}
26
27     @Override
28     public void panini$resolve(org.paninij.examples.pi.Number result) {
29         synchronized (this) {
30             panini$result = result;
31             panini$isResolved = true;
32             this.notifyAll();
33         }
34     }
35
36     @Override
37     public org.paninij.examples.pi.Number panini$get() {
38         while (panini$isResolved == false) {
39             try {
40                 synchronized (this) {
41                     while (panini$isResolved == false) this.wait();
42                 }
43             } catch (InterruptedException e) { /* try waiting again */ }
44         }
45     }
46 }

```

## Generated Message Wrapper



```
package org.paninij.examples.pi;
```

```
import java.util.Random;[]
```

```
/**
```

```
 * Each Worker capsule computes a fraction of the total number of samples.
```

```
 */
```

```
@Capsule
```

```
public class WorkerTemplate
```

```
{
```

```
    Random prng;
```

```
    public void init() {
```

```
        this.prng = new Random();
```

```
    }
```

```
    public Number compute(double num) {
```

```
        Number _circleCount = new Number();
```

```
        for (double j = 0; j < num; j++) {
```

```
            double x = this.prng.nextDouble();
```

```
            double y = this.prng.nextDouble();
```

```
            if ((x * x + y * y) < 1) _circleCount.incr();
```

```
        }
```

```
        return _circleCount;
```

```
    }
```

```
}
```

*Capsule Interface*

*Capsule\$Thread*

*Inter-Capsule  
Messages*

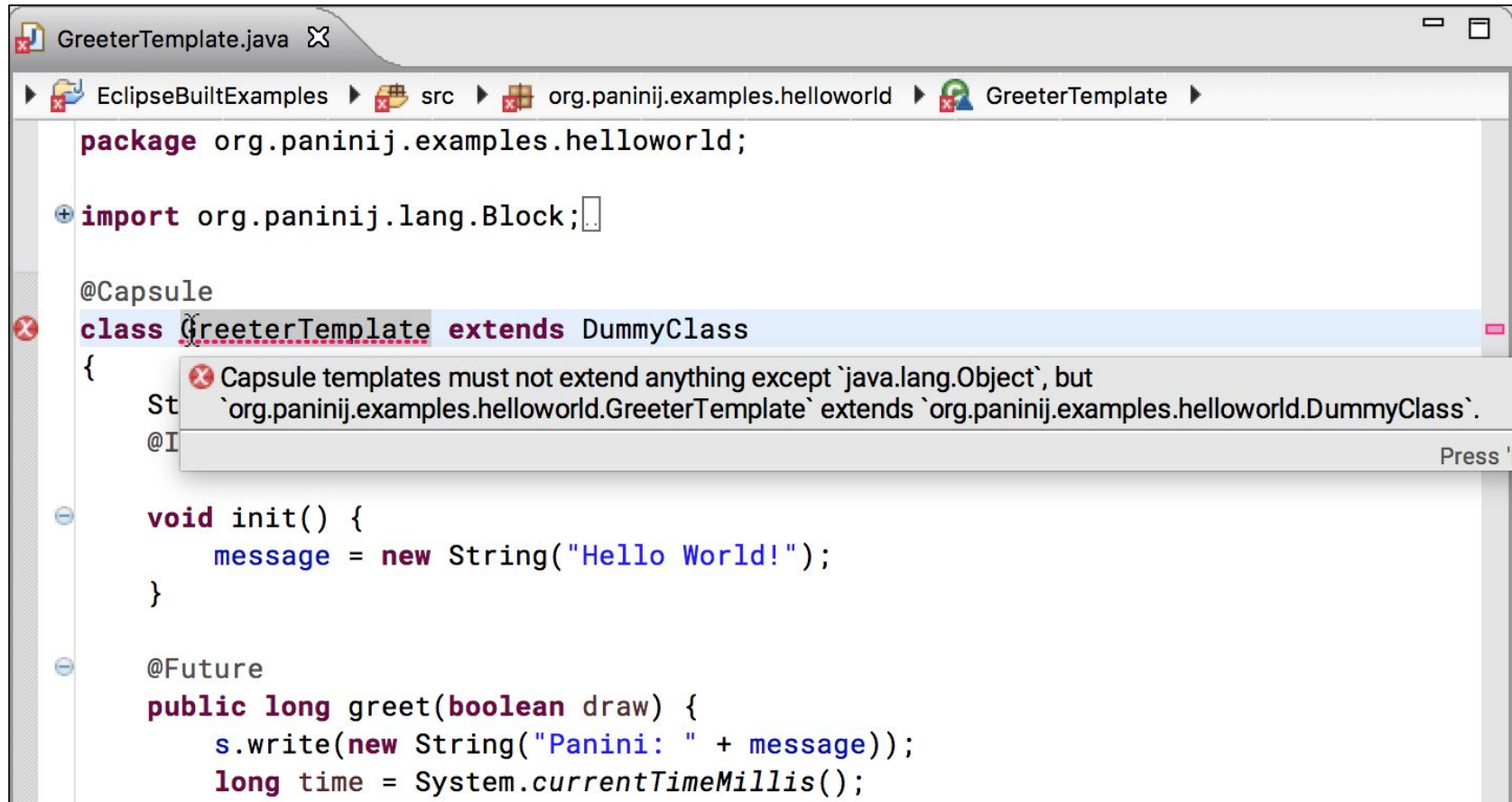
# User's Capsule Template

# Static Checks

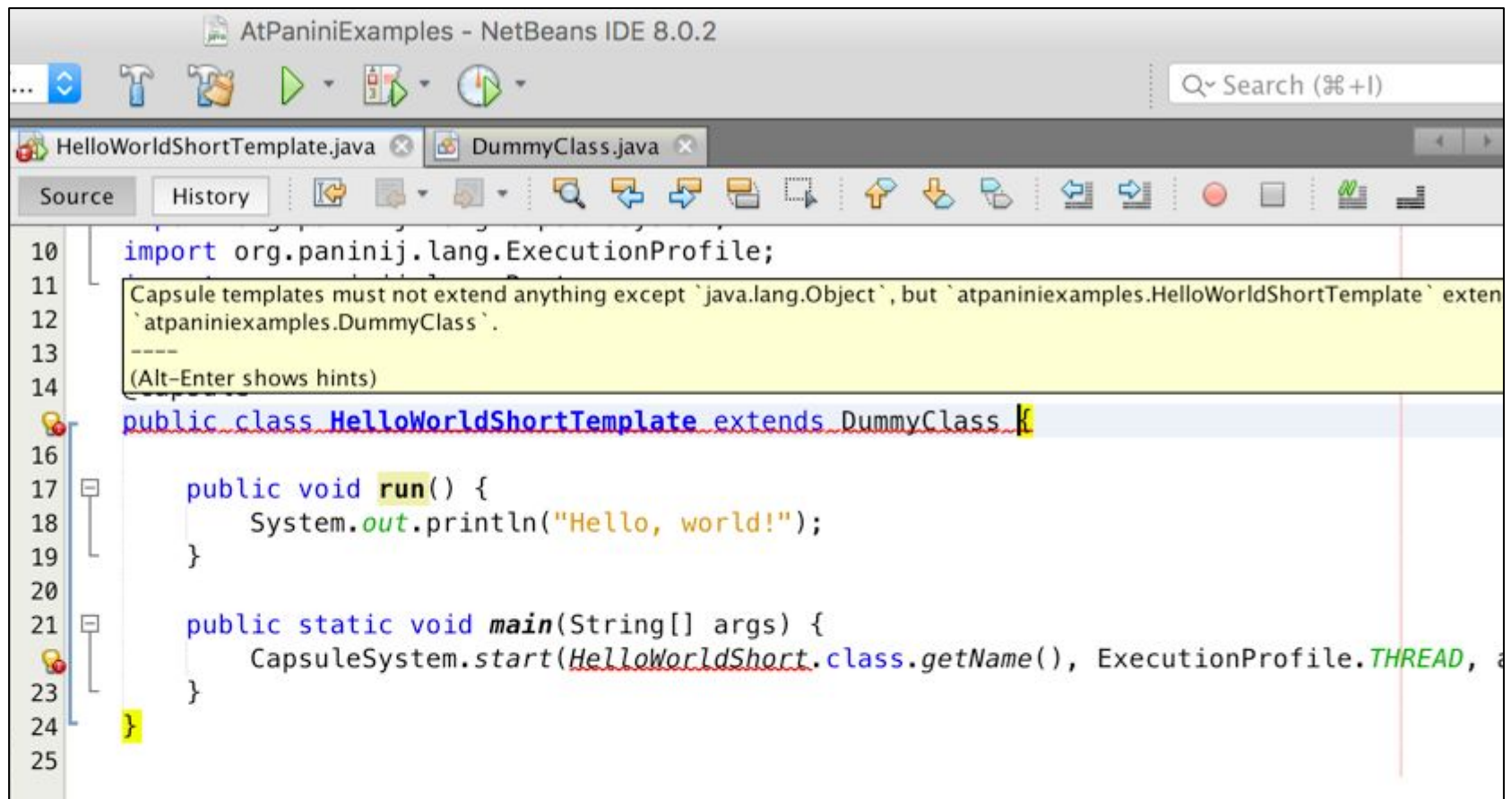
# Static Checking

## Exposing Panini Model via IDE

- Rules identified and implemented as checks.
- Reported from annotation processor
- Violations displayed in context
- 45 checks implemented



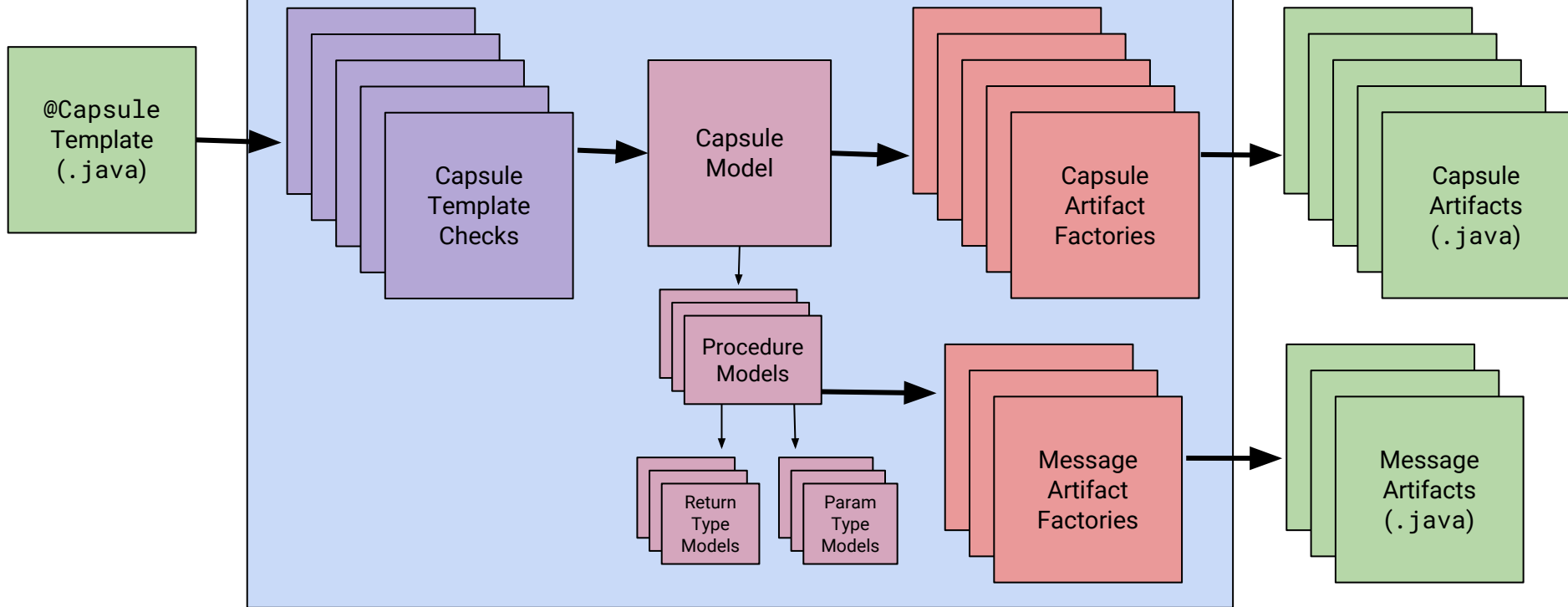
# Static Checking in Eclipse



## Static Checking in NetBeans

# Improving Maintainability and Usability

## Annotation Processor



**PaniniProcessor: Refactored Capsule Processing Dataflow**

# Improving Testing

## Testing Methods

- Invoke compiler with Maven
- Programmatically invoke compiler with `javax.tools`
- Unit testing with Google's `compile-testing`



# @PaniniJ

Getting Started Manual

[Introduction](#) [Installation](#) [Example](#) [More](#)

---

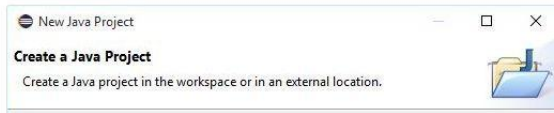
## Installation Overview

These are the overall steps to setting up an @PaniniJ project in Eclipse:

1. Create an Eclipse Project
2. Download the @PaniniJ jar
3. Enable annotation processing
4. Add at-paninij annotation processor to your build
5. Add the at-paninij jar as a referenced library

### 1. Setup project to use JRE 1.7 or Greater

When you create a new project, be sure to choose JRE 1.7 or greater, this is necessary for the annotation processing to work correctly.



# Getting Started Website

## Annotation Type Root

---

```
public @interface Root
```

Used to designate a capsule template as the root capsule.

### ***Purpose***

The purpose of this annotation is to designate a java class to act as a root capsule.

### ***Details***

A root capsule is a capsule from which a capsule system can be started (often also an active capsule.) A root capsule cannot have any fields annotated with `@Imports` (i.e. it cannot have any dependencies). Root capsules are only allowed to send outgoing messages. Therefore it is common for a root capsule to have `@Local` fields.

There is only one root capsule per capsule system. To start the capsule system, use the `CapsuleSystem` class.

### ***Exceptions***

A class annotated with `@Root` must also be annotated with `@Capsule`.

A class annotated with `@Root` must not contain any fields annotated with `@Imports`.

# Client Goals

v0.1.0

No due date ⌚ Last updated less than a minute ago

100% complete 0 open 42 closed

[Edit](#) [Close](#) [Delete](#)

Capsule Oriented Programming shall be:

- *More usable* by Java programmers.
- *More compatible* with existing Java tools.
- *Less complex* to use within Java projects.



Questions?

Questions?

# Spring Semester

- Background: Capsules and PaniniJ
- Identify (specific) client goals
- Reformulated project as @PaniniJ
- Built working prototype

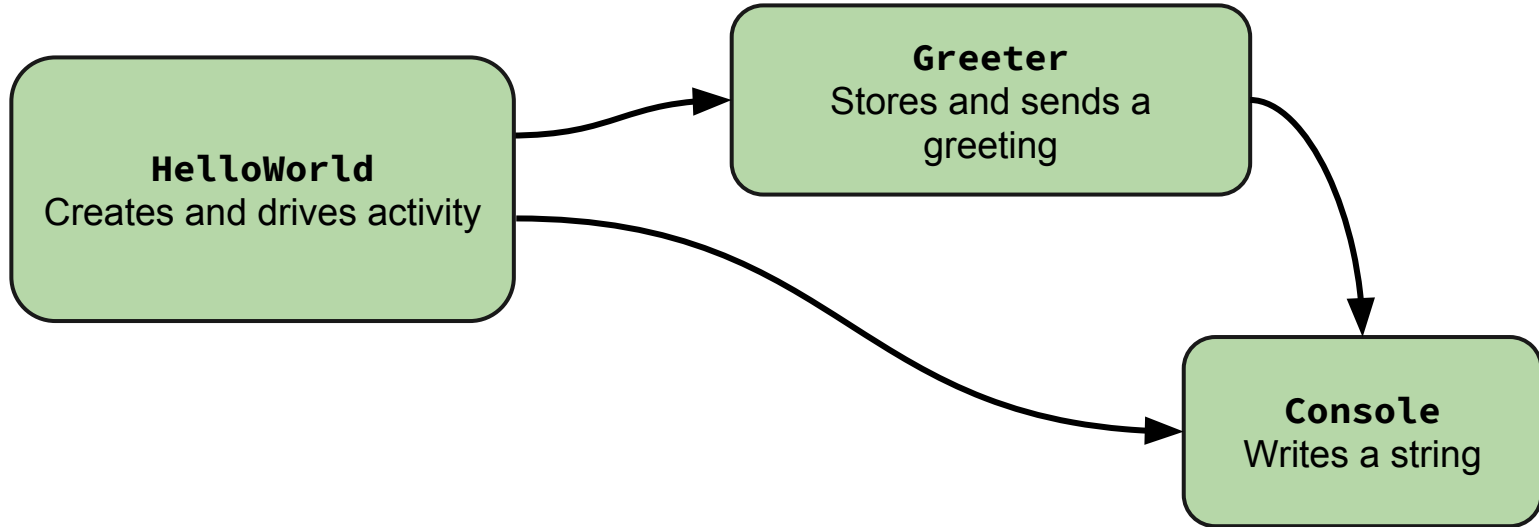
# Fall Semester

- Refine Prototype into Product
  - Processor Refactor
  - Unit & Integration Tests
- Documentation
- Usability Enhancements
  - Static Checks
  - Setup/Compile/Run Ease
- v0.1.0 release

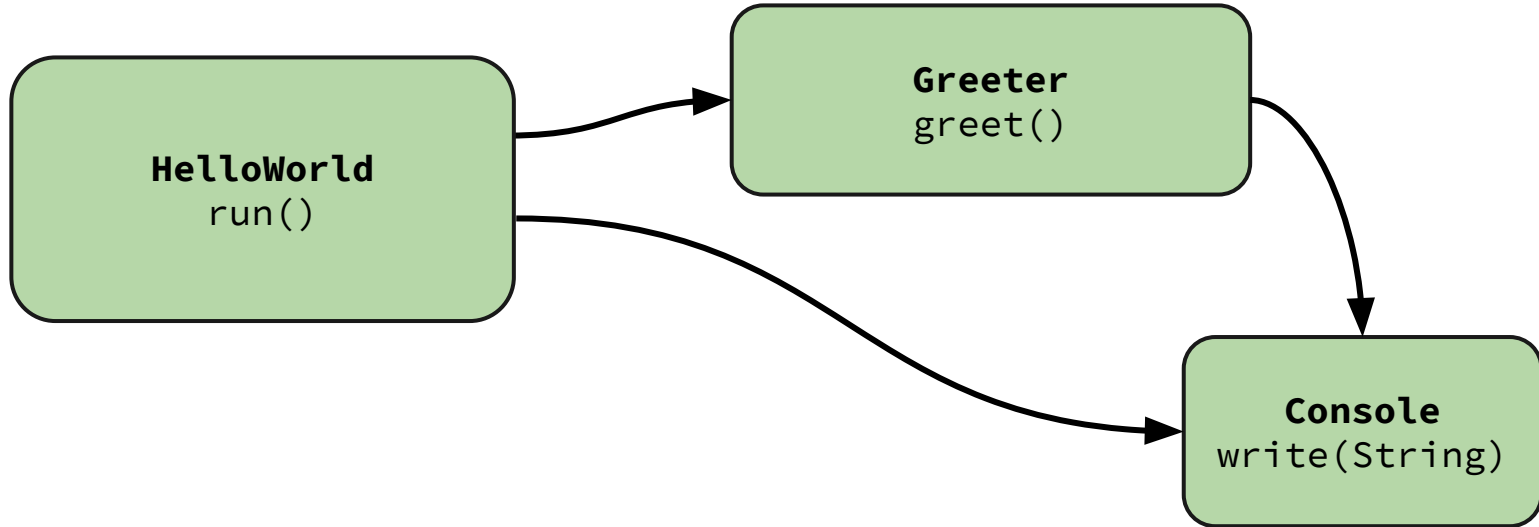
Example Program:  
“Hello, World!”



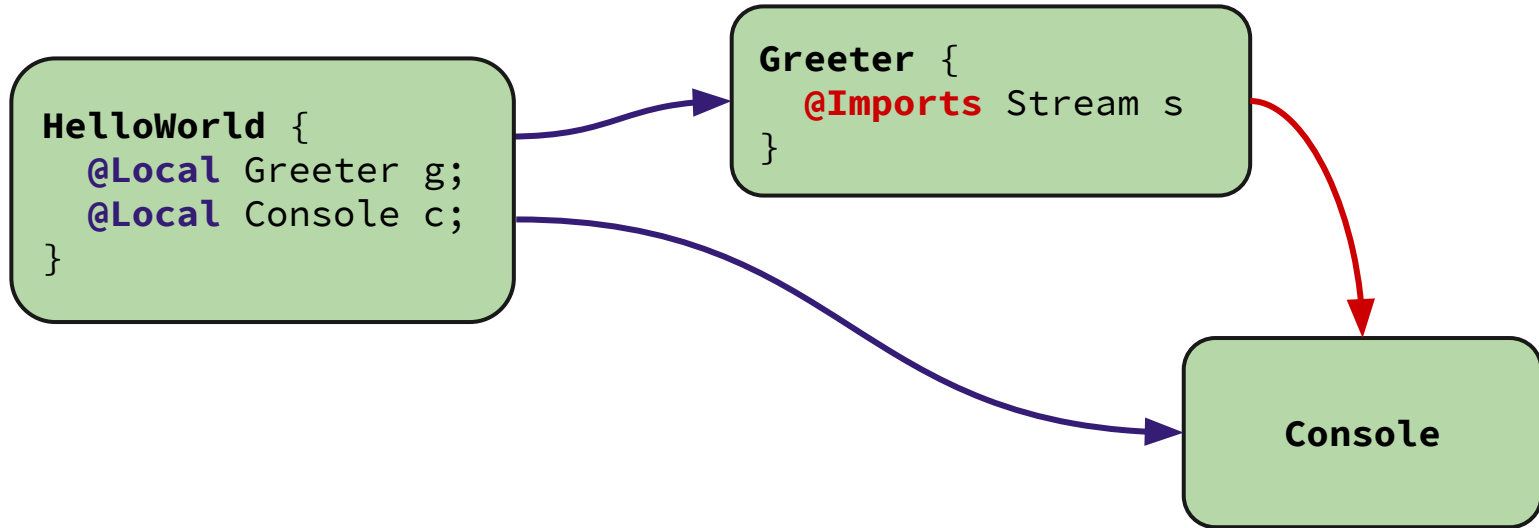
# “Hello World” Example



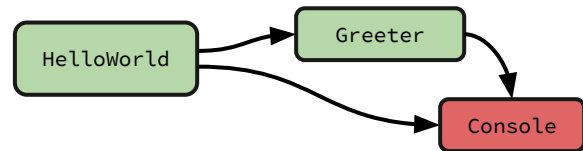
# “Hello World” Example



# “Hello World” Example



```
1 package helloworld;
2
3 import org.paninij.lang.Capsule;
4
5 @Capsule class ConsoleTemplate implements Stream {
6
7     @Override
8     public void write(String s) {
9         System.out.println(s);
10    }
11 }
```

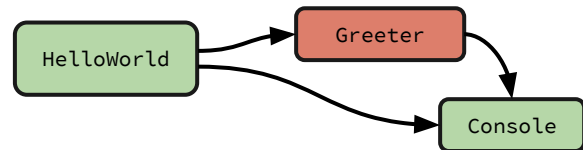


- @Capsule
- User-defined procedure

## “Hello World”: Capsule Template Syntax

## TODO: Change annotations!

```
2
3+import org.panini.lang.Capsule;
5
6 @Capsule class GreeterTemplate {
7
8     String greeting;
9     @Wired Stream s;
10
11-    void init() {
12         greeting = "Hello World!";
13     }
14
15-    public void greet() {
16         s.write("Panini: " + greeting);
17         long time = System.currentTimeMillis();
18         s.write("Time is now: " + time);
19     }
20 }
```



- @Wired
- init()
- User-defined procedure

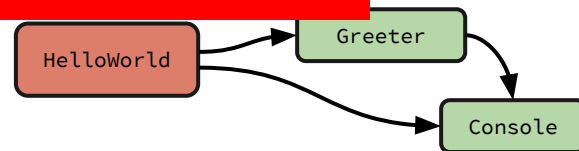
## “Hello World”: Capsule Template Syntax

```

1 package helloworld;
2
3+ import org.paninij.lang.Capsule;
4
5
6 @Capsule class HelloWorldTemplate {
7
8     @Child Console c;
9     @Child Greeter g;
10
11- void design(HelloWorld self) {
12     g.wire(c);
13 }
14
15- void run() {
16     g.greet();
17 }
18 }

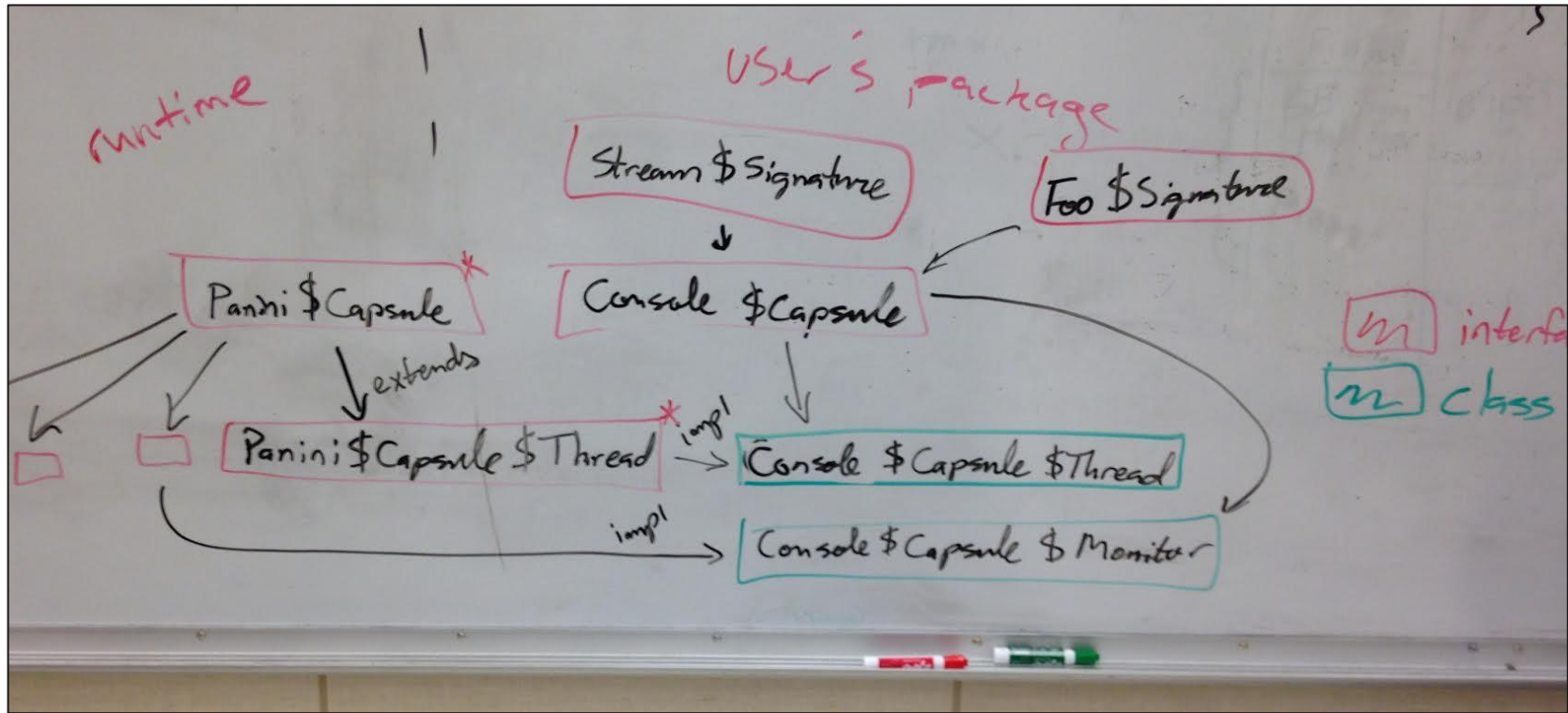
```

**TODO: Change annotations!**



- @Local
- design()
- run()
- imports()

## “Hello World”: Capsule Template Syntax



## Designing Capsule Artifact Inheritance





# Background: Our Client

- ISU Laboratory For Software Design
- Advisor: Dr. Hridesh Rajan
- Research:
  - Software Engineering
  - Programming Language Design
- Collaborators: Panini Project Grad Students

# Background: Panini Project Vision

Make efficient programming abstractions which increase productivity and decrease maintenance costs by making concurrent programming less error-prone.

*Make concurrency less complicated.*

# Development Process

- Rapid Application Development
  - Many prototypes which tackle small problems
  - Documentation along the way
- Tools Used
  - `git`, GitHub, GitHub Issues, GitHub wiki
  - Eclipse, Maven