

Section 1. The software component code of the CIIP system

```
1 //CIIP System Code
2 import java.util.Random;
3 abstract class CIIPState {
4     final static byte dontCare = -1;
5     static byte currentState;
6     public String name, sugar;
7     public static int degreeSugar, dose, doseAvailable;
8     public static byte sampleAttribute, timeAttribute,
        sugarAttribute, doseAttribute, volumeAttribute;
9     Random rand;
10    abstract boolean action();
11 }
12
```

Fig. 1- The software component code (the Abstract class)

Section 1. The software component code of the CIIP system, continue

```
13 class Inactive extends CIIPState {
14     Inactive() {
15         name = "Inactive";
16     }
17     public void initAttributes() {
18         sampleAttribute = dontCare;
19         timeAttribute = dontCare;
20         sugarAttribute = dontCare;
21         doseAttribute = dontCare;
22         volumeAttribute = dontCare;
23     }
24
25     public boolean action() {
26         System.out.print("\n In " + name + " Mode... ");
27         currentState = 1;
28         initAttributes();
29         sampleAttribute = 0;
30         timeAttribute = 0;
31         try {
32             Thread.sleep(600); // Wait for 10 Minutes
33         } catch (InterruptedException e) { // Early Sampling Happened
34             sampleAttribute = 1;
35         }
36         if (sampleAttribute == 0) {
37             timeAttribute = 1; // Wait Time Expired
38             try {
39                 Thread.sleep(60); // Wait for Sampling for 1 Minute
40             } catch (InterruptedException e) {
41                 sampleAttribute = 1; // Timely Sampling Happened
42             }
43         }
44         sampleAttribute = 1;
45         if (sampleAttribute == 1 && timeAttribute == 1)
46             return true; // Report to StateManager to Go Next State
47         else return false;
48     }
49 }
```

Fig. 2- The software component code (the Inactive class for the Inactive operation mode)

Section 1. The software component code of the CIIP system, continue

```
50
51 class Sample extends CIIPState {
52     Sample() {
53         name = "Sample";
54     }
55     public boolean action() {
56         currentState = 2;
57         System.out.print("\\n In " + name + " Mode... ");
58         timeAttribute = dontCare;
59         rand = new Random();
60         degreeSugar = rand.nextInt(20); // Simulation of Blood
                                         Sugar Determination
61
62         if (degreeSugar < 5) {
63             sugarAttribute = 1;
64             sugar = "low";
65             System.out.println("!Blood Sugar : " + sugar);
66             return false;
67         } else
68         if (degreeSugar <= 10) {
69             sugarAttribute = 2;
70             sugar = "normal";
71             System.out.print("(Blood Sugar = " + sugar + ")");
72             return true;
73         } else {
74             sugarAttribute = 3;
75             sugar = "high";
76             System.out.print("(Blood Sugar = " + sugar + ")");
77             return true;
78         }
79     }
80 }
81
```

Fig. 3- The software component code (the Sample class for the Sampling operation mode)

Section 1. The software component code of the CIIP system, continue

```
81 class Compute extends CIIPState {
82     Compute() {
83         name = "Compute";
84     }
85     public boolean action() {
86         System.out.print("\n In " + name + " Mode... ");
87         currentState = 3;
88         sampleAttribute = dontCare;
89         timeAttribute = dontCare;
90         doseAttribute = dontCare;
91         rand = new Random();
92         dose = rand.nextInt(8);
93         // Simulation of Dose Determination
94         rand = new Random();
95
96         if (dose == 0) doseAttribute = 1; // Dose is Zero
97         else if (dose > 0 && dose <=5) doseAttribute = 2; //
98                                     Dose is Normal
99         else {doseAttribute = 3; sugarAttribute = dontCare;}
100                                     //Overdose
101
102         if (doseAttribute == 2) {
103             doseAvailable = rand.nextInt(50);
104             // Simulation of Available Dose Determination
105             if (doseAvailable >= dose) volumeAttribute = 1;
106             // Available Dose is Sufficient
107             else volumeAttribute = 0;
108             // Available Dose is Insufficient
109         }
110         System.out.print("(Dose = " + dose + " Available
111                             dose:" + doseAvailable + ")"); //Dose is Normal
112         if (sugarAttribute == 3 && doseAttribute == 2 &&
113             doseAvailable ==1)
114             return true;
115         else if (sugarAttribute == 2 && doseAttribute == 1) {
116             doseAttribute = dontCare; // Dose is Zero
117             return true;
118         } else return false;
119     }
120 }
```

Fig. 4- The software component code (the Compute class for the Computing operation mode)

Section 1. The software component code of the CIIP system, continue

```
114 class Deliver extends CIIPState {
115     Deliver() {
116         name = "Deliver";
117     }
118     public boolean action() {
119         System.out.println("\n In " + name + "Mode... ");
120         currentState = 4;
121         try {
122             Thread.sleep(60);
123             // Simulation of Insulin Delivering Time
124         } catch (InterruptedException e) {}
125         return true;
126     }
127 }
128 class StateManager {
129     long start, end;
130     public CIIPState currentState;
131     Inactive iState;
132     Sample sState;
133     Compute cState;
134     Deliver dState;
135     boolean stateResult;
136     StateManager() {
137         iState = new Inactive();
138         sState = new Sample();
139         cState = new Compute();
140         dState = new Deliver();
141     }
142     public void Manager() {
143         currentState = iState;
144         stateResult = currentState.action();
145         if (stateResult) {
146             // Normal Inactive State Termination
147             currentState = sState;
148             stateResult = currentState.action();
149             if (stateResult) {
150                 // Normal Sampling State Termination
151                 currentState = cState;
152                 stateResult = currentState.action();
153                 if (stateResult) {
154                     // Normal Computing State Termination
155                     currentState = dState;
156                     stateResult = currentState.action();
157                 }
158             }
159         }
160     }
161 }
```

Fig. 5- The software component code (the Deliver class for the Deliver operation mode and the Manager class for management of switching between operation modes

## Section 2. The monitor code of the CIIP system (Implemented in Java)

```
1 public aspect Verification {
2
3     final static byte dontCare = -1;
4     final static int attributeNos = 5;
5     final static int lowerInactiveIndex = 0, upperInactiveIndex = 1;
6     final static int lowerSampleIndex = 2, upperSampleIndex = 2;
7     final static int lowerComputeIndex = 3, upperComputeIndex = 6;
8     static int patternNos = 7;
9     static int counter, checkedConstraints = 0;
10    static String abnormalPatterns[] = new String [patternNos];
11    static String currentStringPattern, msg;
12    static byte currentPattern[] = new byte [attributeNos];
13    static long start, end, monitorTime = 0;
14
15    public static byte Abnormals[][] = {
16 // Inactive State Abnormals
17         {1, 0, dontCare, dontCare, dontCare},
18         {0, 1, dontCare, dontCare, dontCare},
19 //Sampling State Abnormals
20         {1, dontCare, 1, dontCare, dontCare},
21 //Computing State Abnormals
22         {dontCare, dontCare, dontCare, 3, dontCare},
23         {dontCare, dontCare, 2, 2, dontCare},
24         {dontCare, dontCare, 3, 1, dontCare},
25         {dontCare, dontCare, 3, 2, 0}
26    };
27
28    public static String msgs [] = {
29        "Sampling is Early", "Sampling is Late",
30        "Blood Sugare Dropped",
31        "Dose is Over", "Dose Must Be Zero", "Dose Must Not Be
32        Zero", "Insufficient Available Dose"
33    };
34 }
```

Fig. 1- Safety aspect definition

Section 2. The monitor code of the CIIP system, continue

```
39 public void CIIPState.searchAbnormal() {
40     currentPattern[0] = CIIPState.sampleAttribute;
41     currentPattern[1] = CIIPState.timeAttribute;
42     currentPattern[2] = CIIPState.sugarAttribute;
43     currentPattern[3] = CIIPState.doseAttribute;
44     currentPattern[4] = CIIPState.volumeAttribute;
45     currentStringPattern = new String(currentPattern);
46     msg = "";
47     switch (CIIPState.currentState) {
48         case 1:
49             for (counter = lowerInactiveIndex; counter <=
50                 upperInactiveIndex; counter++) {
51                 checkedConstraints++;
52                 if (currentStringPattern.equals
53                     (abnormalPatterns[counter])) { msg =
54                         msgs[counter]; break; }
55             }
56             return;
57         case 2:
58             for (counter = lowerSampleIndex; counter <=
59                 upperSampleIndex; counter++) {
60                 checkedConstraints++;
61                 if (currentStringPattern.equals
62                     (abnormalPatterns[counter])) {msg =
63                         msgs[counter]; break; }
64             }
65             return;
66         case 3:
67             for (counter = lowerComputeIndex; counter <=
68                 upperComputeIndex; counter++) {
69                 checkedConstraints++;
70                 if (currentStringPattern.equals
71                     (abnormalPatterns[counter])) {msg =
72                         msgs[counter]; break; }
73             }
74             return;
75     }
76 }
```

Fig. 2- The main code

Section 2. The monitor code of the CIIP system, continue

```
71     pointcut verifyState(CIIPState c) :  
72         call(* action()) && target(c);  
73         after(CIIPState c) : verifyState(c){  
74             start = System.nanoTime();  
75             c.searchAbnormal();  
76             end = System.nanoTime();  
77             monitorTime += end - start;  
78             System.out.println(" Already Monitored Constrains:  
79                 " + checkedConstraints +  
80                 " Already Monitoring Time: " +  
81                 monitorTime / 1000000.0);  
82             if (msg.length() > 0) System.out.println("Bad Event-->"  
                 + msg);  
81         }  
82     }
```

Fig. 3- The weaving method,  
for directing AspectJ compiler

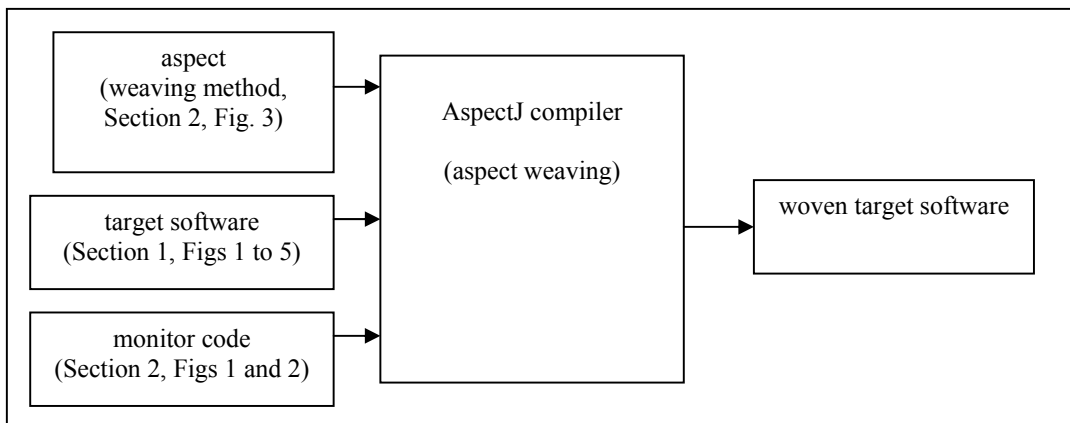


Fig. 4- Automatic instrumentation of target software using the AspectJ compiler

### Section 3. The instrumented target software and its execution

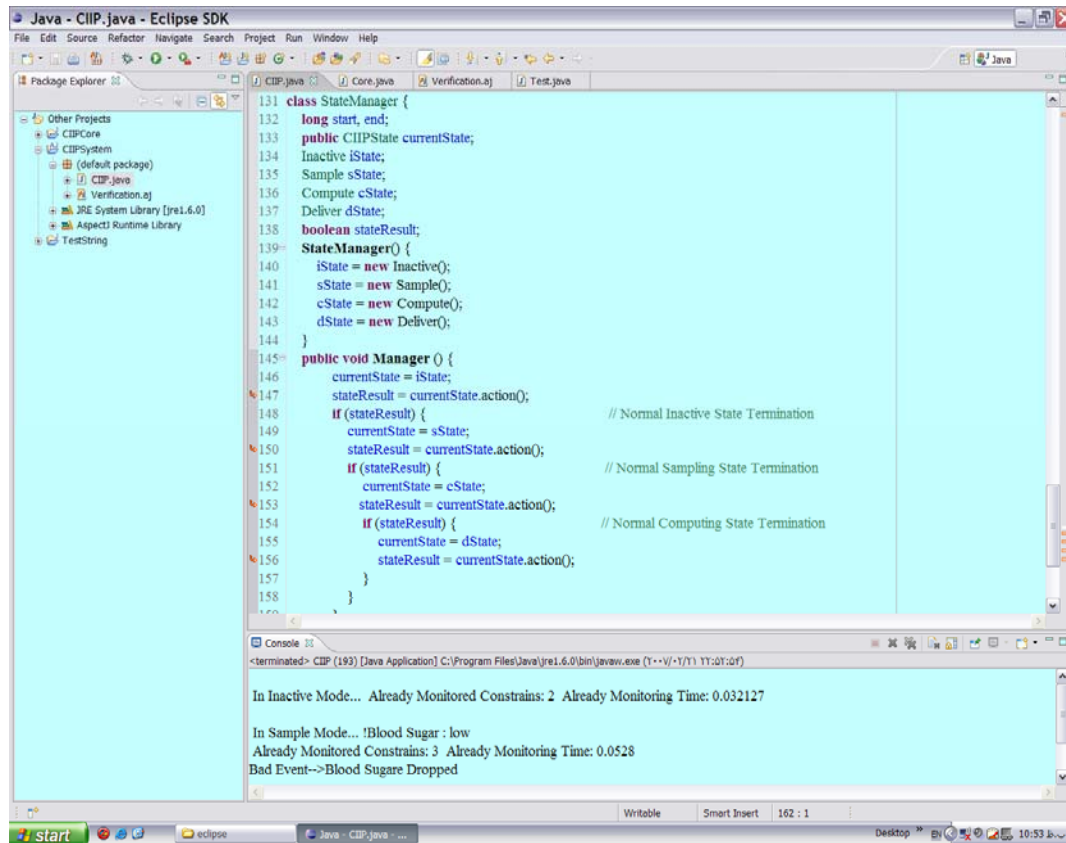


Fig. 1- A segment of instrumented target software (the woven code has been identified by red arrows) and The warning message issued by the monitor code (at the foot of the figure)