TP : Machine à bonbons

Compétences visées :

- Mise en pratique des connaissances acquises en modélisation UML.
- Mise en pratique des connaissances acquises en programmation Java.
- Mise en pratique du patron de conception (*design pattern*) *State*.
- Mise en pratique de l'utilisation d'un logger
- Mise en pratique des tests unitaires avec JUnit 5 et de la couverture de code.

Le code d'une machine à bonbons est donné dans la Figure 1. Après une analyse du code, vous devrez proposer une nouvelle modélisation de cette machine à bonbons en vous aidant du patron de conception *State*. Pour vous assurer que votre nouvelle modélisation n'altère pas les fonctionnalités de la machine à bonbons, vous devrez implanter et tester votre solution.

```
1  package fr.esisar.gumball;
2
3  import org.apache.logging.log4j.LogManager;
4  import org.apache.logging.log4j.Logger;
5
6  public class GumballMachine {
7
8        private static final Logger LOGGER =
9  LogManager.getLogger(GumballMachine.class);
10
11       static final int SOLD_OUT = 0;
12       static final int NO_QUARTER = 1;
13       static final int HAS_QUARTER = 2;
14       static final int SOLD = 3;
15
16       private int state = SOLD_OUT;
17       private int count = 0;
18
19       public GumballMachine(int count) {
20             super();
21             this.count = count;
22             if (count > 0) {
23                   state = NO_QUARTER;
24             }
25       }
26
27       public void insertQuarter() {
28             if (state == HAS_QUARTER) {
29                   LOGGER.info("You can't insert another quarter");
30             } else if (state == NO_QUARTER) {
31                   LOGGER.info("You inserted a quarter");
32                   state = HAS_QUARTER;
```

```
33              } else if (state == SOLD) {
34                    LOGGER.info("Please wait, we're already giving you a gumball");
35              } else if (state == SOLD_OUT) {
36                    LOGGER.info("You can't insert a quarter, the machine is sold
37  out");
38              }
39        }
40
41        public void ejectQuarter() {
42              if (state == HAS_QUARTER) {
43                    LOGGER.info("Quarter returned");
44                    state = NO_QUARTER;
45              } else if (state == NO_QUARTER) {
46                    LOGGER.info("You haven't inserted a quarter");
47              } else if (state == SOLD) {
48                    LOGGER.info("Sorry, you already turned the crank");
49              } else if (state == SOLD_OUT) {
50                    LOGGER.info("You can't eject, you haven't inserted a quarter
51  yet");
52              }
53        }
54
55        public void turnCrank() {
56              if (state == HAS_QUARTER) {
57                    LOGGER.info("You turned...");
58                    state = SOLD;
59                    dispense();
60              } else if (state == NO_QUARTER) {
61                    LOGGER.info("You turned but there's no quarter");
62              } else if (state == SOLD) {
63                    LOGGER.info("Turning twice doesn't get you another gumball!");
64              } else if (state == SOLD_OUT) {
65                    LOGGER.info("You turned, but there are no gumballs");
66              }
67        }
68
69        private void dispense() {
70              if (state == HAS_QUARTER) {
71                    LOGGER.info("No gumball dispensed");
72              } else if (state == NO_QUARTER) {
73                    LOGGER.info("You need to pay first");
74              } else if (state == SOLD) {
75                    LOGGER.info("A gumball comes rolling out the slot");
76                    if (count == 0) {
77                          LOGGER.info("Oops, out of gumballs!");
78                          state = SOLD_OUT;
79                    } else {
80                          count = count - 1;
81                          state = NO_QUARTER;
82                    }
83              } else if (state == SOLD_OUT) {
84                    LOGGER.info("No gumball dispensed");
85              }
86        }
87
88        @Override
89        public String toString() {
90              StringBuilder result = new StringBuilder();
91              result.append("\nMighty Gumball, Inc.");
```

```
92              result.append("\nJava-enabled Standing Gumball Model #2022\n");
93              result.append("Inventory: " + count + " gumball");
94              if (count <= 1) {
95                    result.append("s");
96              }
97              result.append("\nMachine is ");
98              if (state == HAS_QUARTER) {
99                    result.append("waiting for turn of crank");
100             } else if (state == NO_QUARTER) {
101                   result.append("waiting for quarter");
102             } else if (state == SOLD) {
103                   result.append("delivering a gumball");
104             } else if (state == SOLD_OUT) {
105                   result.append("sold out");
106             }
107             result.append("\n");
108             return result.toString();
109       }
110 }
```

Figure 1 : Code de la machine à bonbons.

Travail à réaliser

1. Analysez le code existant afin de mettre en évidence le fonctionnement de la machine à bonbons et d'identifier les limites de ce code. Cette analyse doit être argumentée, notamment, par des diagrammes UML. Pourquoi la visibilité de la méthode dispense est-elle différente de la visibilité des autres méthodes de la classe GumballMachine ?
2. Après avoir étudié les limites de cette version de la machine à bonbons, proposez une nouvelle modélisation (diagramme de classes) de cette machine en s'inspirant du patron de conception *State*.
   Attention : la méthode dispense() est privée dans la première version du code. Réfléchissez bien !
3. Implantez votre diagramme de classes. Votre projet Java doit respecter toutes les bonnes pratiques vues en cours de Génie Logiciel, comme :
   a. le respect des normes de développement Java,
   b. la structuration de votre projet,
   c. l'automatisation des tâches,
   d. la documentation de votre code.
4. Ajoutez à votre nouvelle version de la machine à bonbons une classe GumballMachineTestDrive, permettant d'exécuter votre code et d'obtenir la même trace d'exécution avec les deux versions de la machine à bonbons.
5. Réalisez 4 classes de tests de votre machine à bonbons dans un répertoire de type « Source Folder », à la racine de votre projet, nommé test. Vos tests devront tester le bon fonctionnement de la machine à bonbons ; c'est-à-dire que, pour un état donné de la machine à bonbons avec un nombre donné de bonbons, après une action (insertQuarter, ejectQuarter ou bien turnCrank), vous obtenez bien le bon nombre de bonbons et le bon message. Les tests seront faits avec une machine à bonbons vide ou contenant des bonbons. Les méthodes et les attributs privés ainsi que les traces des loggers ne doivent pas être testés

Pour vous aider :

- chaque classe de tests aura comme attribut une machine à bonbons et un état. La machine à bonbons sera initialisée dans une méthode dédiée avec le nombre de bonbons et son état (i.e., l'attribut état). Cette méthode devra être utilisée avant chaque test.
- chaque classe de tests devra avoir une méthode de tests par opérations possibles sur la machine à bonbons (insertQuarter, ejectQuarter ou bien turnCrank).
- chaque méthode de tests devra avoir deux assertions : l'une pour le nombre de bonbons et l'autre pour le message retourné par la machine à bonbons (chaîne de caractères retournée par la méthode toString()). Exemple :

```
Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2022
Inventory: 5 gumballs
Machine is waiting for turn of crank
```

- Pour utiliser une méthode d'initialisation différente selon si la machine à bonbons a des bonbons ou non, il vous est conseillé d'utiliser l'annotation @Nested (voir documentation JUnit 5).

Exemple de tableau de tests :

Etat de départ	Nombre initial de bonbons	Actions	Nombre final de bonbons
HAS_QUARTER	5	insertQuarter	5
		ejectQuarter	5
		turnCrank	4
	0	insertQuarter	0
		ejectQuarter	0
		turnCrank	0

6. Exécutez vos tests et utilisez l'outil de couverture du code inclus dans Eclipse pour évaluer la couverture de votre code. Que pensez-vous du résultat obtenu ?