# Tutorial course 2: Design patterns

Guillaume Lachaud

## Contents

**Recognizing design patterns** **3**
Recognizing a design from code . . . . . 3
Recognizing from a diagram . . . . . 3
Differentiate two similar patterns . . . . . 3

**Embrace Peer Reviews and Feedbacks** **3**

**Make design generics** **8**

**Refactoring** **8**
Factories . . . . . 8

**Adapter** **8**

## List of Figures

1 Part 1 of the feedbacks . . . . . 4
2 Part 2 of the feedbacks . . . . . 5
3 Part 3 of the feedbacks . . . . . 5
4 Part 4 of the feedbacks . . . . . 6
5 Part 5 of the feedbacks . . . . . 6
6 Part 6 of the feedbacks . . . . . 7

## Recognizing design patterns

### Recognizing a design from code

The class **EnchantedMazeGame** should be implemented like this:

Listing 1: EnchantedMazeGame

```
class EnchantedMazeGame extends MazeGame {

    public Room makeRoom(int n, String spell) {
        return new EnchantedRoom(n, spell);
    }
    public Wall makeWall() {
        return new SecretPassageWall();
    }
    public Door makeDoor(Room r1, Room r2, String spell) {
        return new DoorWithSpell(r1, r2, spell);
    }
```

### Recognizing from a diagram

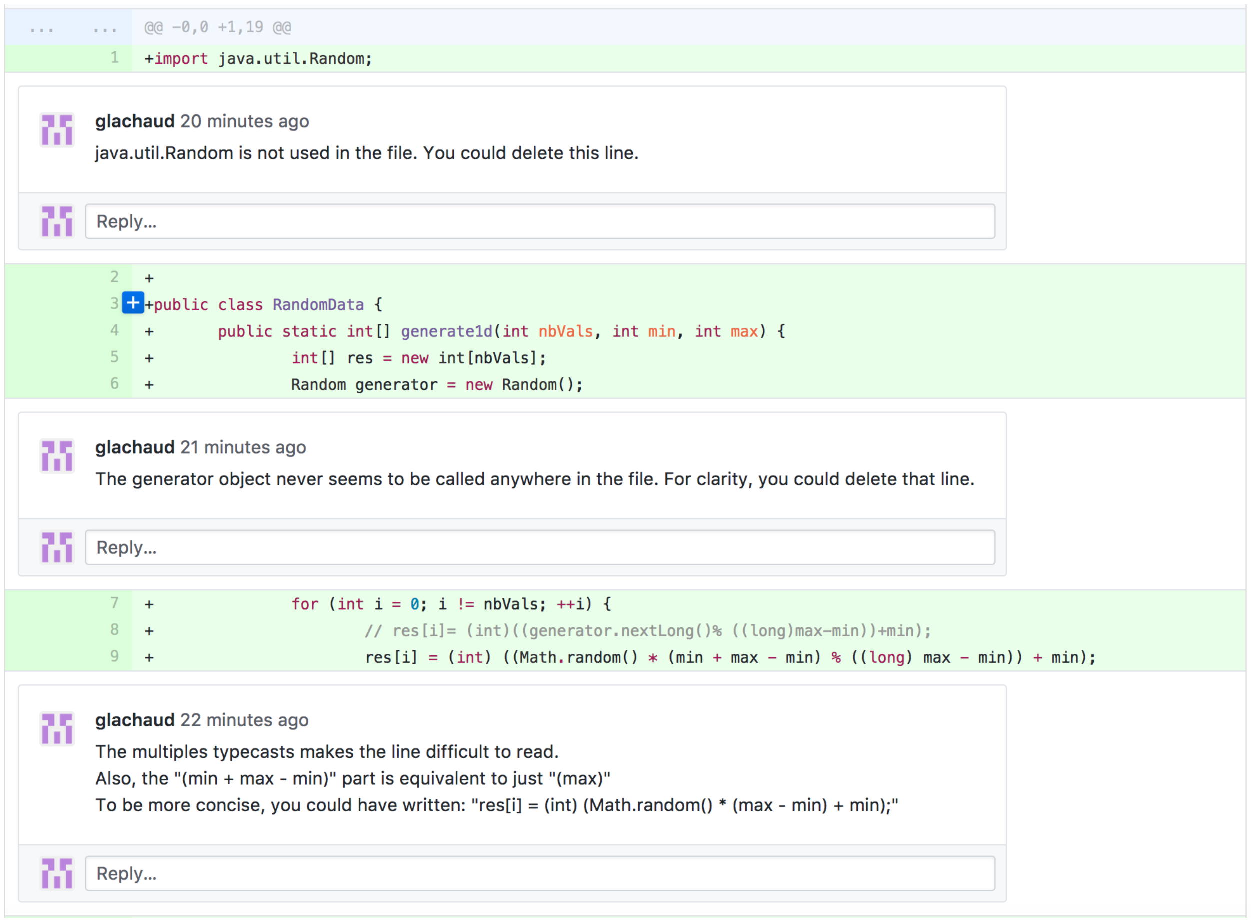
The pattern used is the **Observer pattern**.
When the stockPrices change, the MobileDisplay and the WebDisplay are updated to reflect the change.

### Differentiate two similar patterns

Pattern A is a **Strategy pattern**.
Pattern B is a **State pattern**.
The main difference between them is that in the Pattern A, the different implementations of the interface are independent.
As is said in the book *Design Patterns*, one example of the **Strategy pattern** is a Composition class responsible for the linebreaks in a text viewer, whereas an example of the **State pattern** is an implementation of a TCP connection.

## Embrace Peer Reviews and Feedbacks

The feedbacks were done on Dalila Ahemed's TP1.



```
@@ -0,0 +1,19 @@
+import java.util.Random;
```

**glachaud** 20 minutes ago
java.util.Random is not used in the file. You could delete this line.

Reply...

```
+
+public class RandomData {
+       public static int[] generate1d(int nbVals, int min, int max) {
+               int[] res = new int[nbVals];
+               Random generator = new Random();
```

**glachaud** 21 minutes ago
The generator object never seems to be called anywhere in the file. For clarity, you could delete that line.

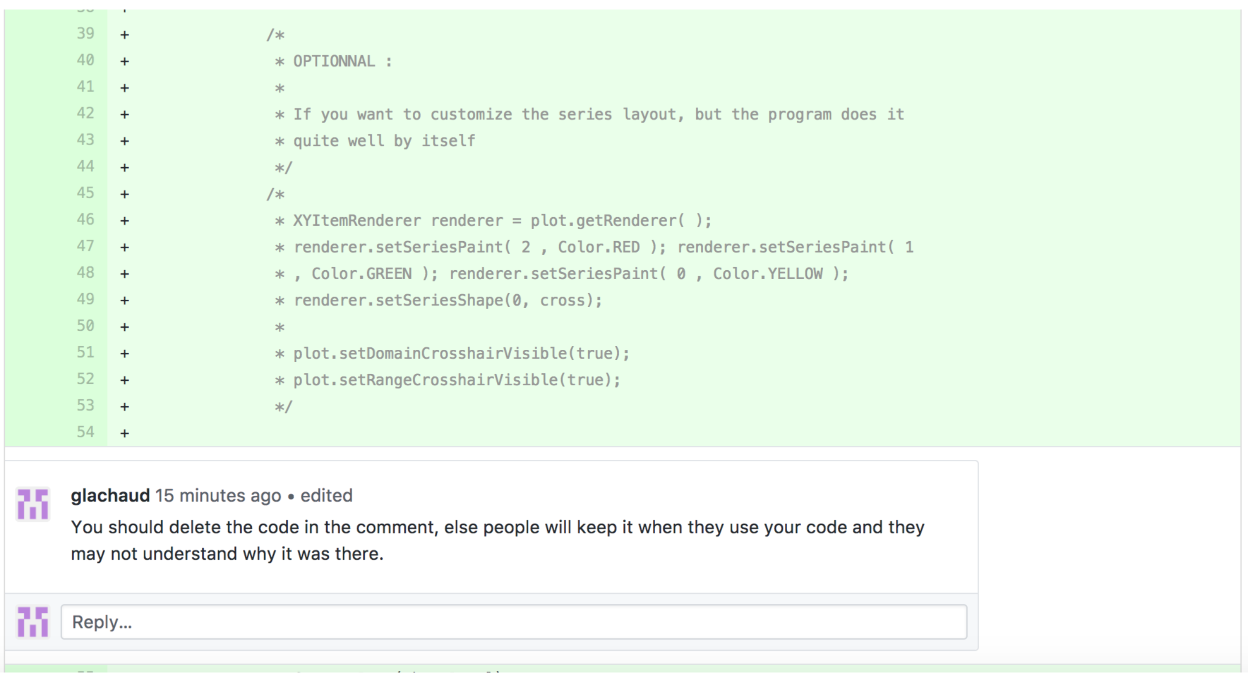
Reply...

```
+               for (int i = 0; i != nbVals; ++i) {
+                       // res[i]= (int)((generator.nextLong()% ((long)max-min))+min);
+                       res[i] = (int) ((Math.random() * (min + max - min) % ((long) max - min)) + min);
```

**glachaud** 22 minutes ago
The multiples typecasts makes the line difficult to read.
Also, the "(min + max - min)" part is equivalent to just "(max)"
To be more concise, you could have written: "res[i] = (int) (Math.random() * (max - min) + min);"

Reply...

Figure 1: Part 1 of the feedbacks



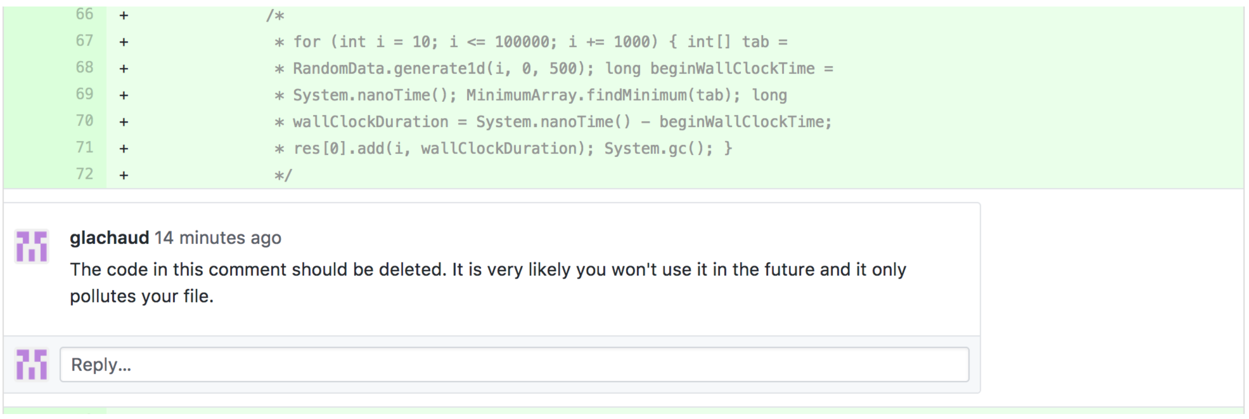
```
+
+               /*
+                * OPTIONNAL :
+                *
+                * If you want to customize the series layout, but the program does it
+                * quite well by itself
+                */
+               /*
+                * XYItemRenderer renderer = plot.getRenderer( );
+                * renderer.setSeriesPaint( 2 , Color.RED ); renderer.setSeriesPaint( 1
+                * , Color.GREEN ); renderer.setSeriesPaint( 0 , Color.YELLOW );
+                * renderer.setSeriesShape(0, cross);
+                *
+                * plot.setDomainCrosshairVisible(true);
+                * plot.setRangeCrosshairVisible(true);
+                */
+
```

**glachaud** 15 minutes ago • edited

You should delete the code in the comment, else people will keep it when they use your code and they may not understand why it was there.

Reply...

Figure 2: Part 2 of the feedbacks



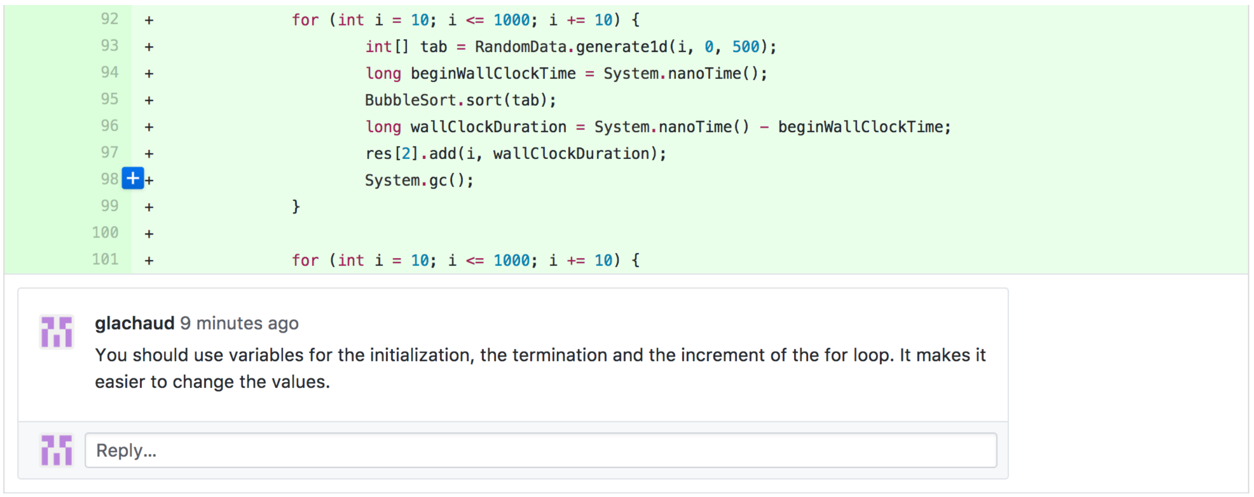
```
+               /*
+                * for (int i = 10; i <= 100000; i += 1000) { int[] tab =
+                * RandomData.generate1d(i, 0, 500); long beginWallClockTime =
+                * System.nanoTime(); MinimumArray.findMinimum(tab); long
+                * wallClockDuration = System.nanoTime() - beginWallClockTime;
+                * res[0].add(i, wallClockDuration); System.gc(); }
+                */
```

**glachaud** 14 minutes ago

The code in this comment should be deleted. It is very likely you won't use it in the future and it only pollutes your file.

Reply...

Figure 3: Part 3 of the feedbacks

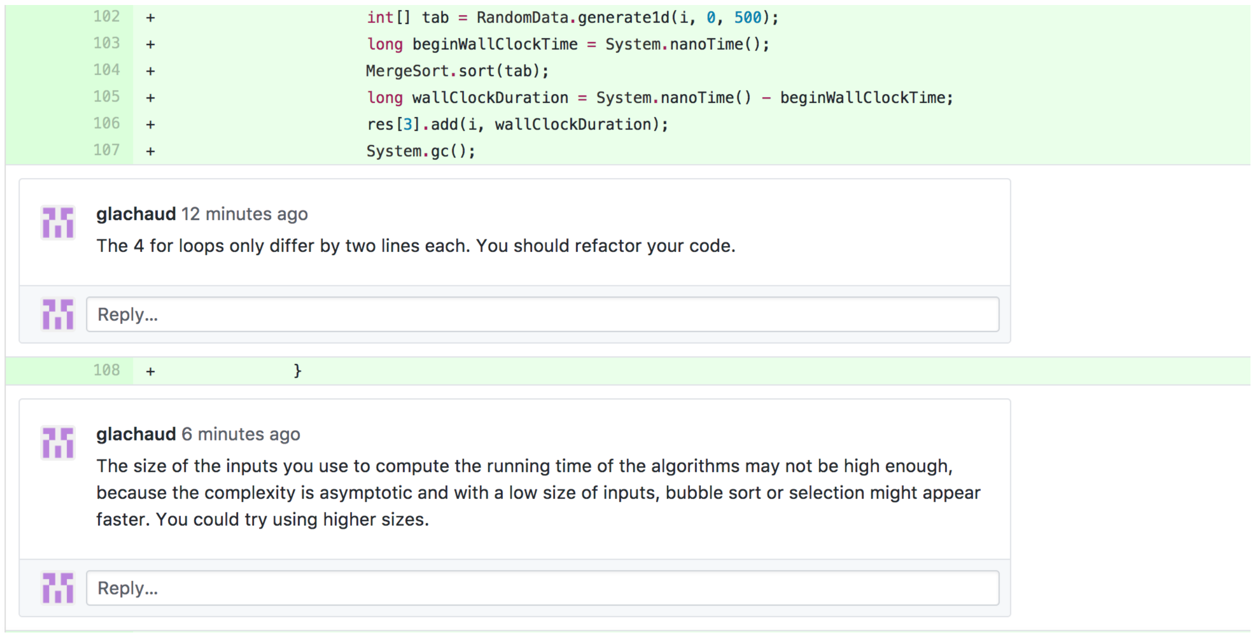


```
+            for (int i = 10; i <= 1000; i += 10) {
+                    int[] tab = RandomData.generate1d(i, 0, 500);
+                    long beginWallClockTime = System.nanoTime();
+                    BubbleSort.sort(tab);
+                    long wallClockDuration = System.nanoTime() - beginWallClockTime;
+                    res[2].add(i, wallClockDuration);
+                    System.gc();
+            }
+
+            for (int i = 10; i <= 1000; i += 10) {
```

**glachaud** 9 minutes ago

You should use variables for the initialization, the termination and the increment of the for loop. It makes it easier to change the values.

Figure 4: Part 4 of the feedbacks



```
+                    int[] tab = RandomData.generate1d(i, 0, 500);
+                    long beginWallClockTime = System.nanoTime();
+                    MergeSort.sort(tab);
+                    long wallClockDuration = System.nanoTime() - beginWallClockTime;
+                    res[3].add(i, wallClockDuration);
+                    System.gc();
```

**glachaud** 12 minutes ago

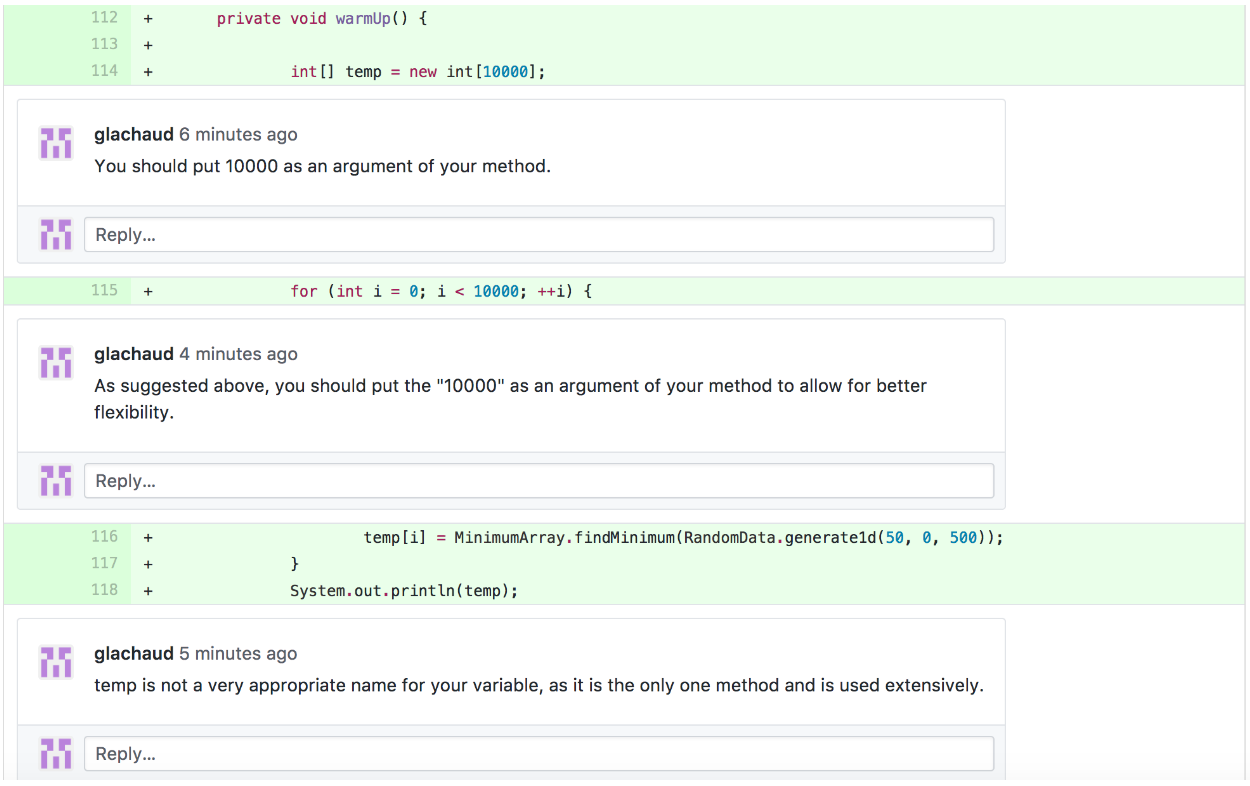
The 4 for loops only differ by two lines each. You should refactor your code.

```
+            }
```

**glachaud** 6 minutes ago

The size of the inputs you use to compute the running time of the algorithms may not be high enough, because the complexity is asymptotic and with a low size of inputs, bubble sort or selection might appear faster. You could try using higher sizes.

Figure 5: Part 5 of the feedbacks



```
+    private void warmUp() {
+
+            int[] temp = new int[10000];
```

**glachaud** 6 minutes ago
You should put 10000 as an argument of your method.

Reply...

```
+            for (int i = 0; i < 10000; ++i) {
```

**glachaud** 4 minutes ago
As suggested above, you should put the "10000" as an argument of your method to allow for better flexibility.

Reply...

```
+                    temp[i] = MinimumArray.findMinimum(RandomData.generate1d(50, 0, 500));
+            }
+            System.out.println(temp);
```

**glachaud** 5 minutes ago
temp is not a very appropriate name for your variable, as it is the only one method and is used extensively.

Reply...

Figure 6: Part 6 of the feedbacks

## Make design generics

This design is very far from generic.
One of the obvious flaws is that the code for the bubble sort benchmarks and the code for the selection sort benchmarks are the same, apart from two lines.
It is not open for extension. If you want to use another sort, you would have to copy and paste the code and change the lines concerned with the instantiation of the sorting method and the sorting.
In addition to that, the values in the *for* loop are hard coded, so if we wanted to use these benchmarks in a larger project, we would have to manually change the values.

To improve the model, we could use an enum to store all the names of the sorts and use an interface that all the sorting class implement.

## Refactoring

### Factories

I already did it for the Strategy Pattern, but we should put the list of all the sorting algorithms in an enum to be able, when we call the factory, to create the right sorting algorithm.
We can use a normal factory, because the sorting algorithms need only one shared function: **sort**.
The model would be the same as the Strategy pattern, except we had a *SortFactory* that creates a sorting algorithm. That is, we still have the interface *Sort* that defines the **sort** method and all the sorting algorithms implement this interface.

## Adapter

The code should like that:

Listing 2: Adapter class

```
class Adapter extends DList implements Stack {

        void push(Object o) {
                this.insertHead(o);
        }
        Object pop() {
                this.removeHead();
        }
        Object top() {
                this.getHead();
        }
```