

## Appendices O

### Task Scheduling

```
23  public TaskGenerator(Server server, int time) {  
24      this.server = (ServerImpl) server;  
25      this.timer = new Timer();  
26      this.date = Calendar.getInstance();  
27      this.setDate();  
28      setSchedule(time);  
29  }
```

Fig x – Extract from TaskGenerator, TaskGenerator constructor

As you can see from fig x, I pass a server object and an integer time value, which is the time in milliseconds between each task that is to be generated.

```
87  private final TaskGenerator scheduler;
```

Fig x - Extract from ServerImpl class, create a TaskGenerator variable

```
126  //Schedule to run every Day at midnight  
127  scheduler = new TaskGenerator(this, 1000 * 60 * 60 * 24);
```

Fig x – Extract from ServerImpl constructor, initialize TaskGenerator variable

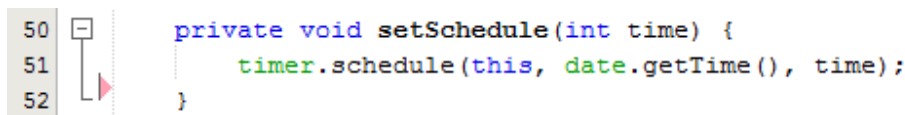
As you can see from fig x and fig x, I create a TaskGenerator variable pass it this server instance and 86,400,000 (the number of milliseconds within a day), which then creates a TaskGenerator object that will run every 24 hours.

As you can see from fig x, I then initialise the timer variable with a new Timer instance, I create a new Calendar instance with Calender.getInstance() and then invoke local methods setDate() and setSchedule() where I pass the time in milliseconds to the setSchedule method as a parameter.

```
43  private void setDate() {  
44      date.set(Calendar.HOUR, 0);  
45      date.set(Calendar.MINUTE, 0);  
46      date.set(Calendar.SECOND, 0);  
47      date.set(Calendar.MILLISECOND, 0);  
48  }
```

Fig x – Extract from TaskGenerator class, setDate()

As you can see from fig x, the setDate() method invoked within the TaskGenerator constructor just sets the time part of the Calendar variable called date to midnight, this will be used as the benchmark for when any tasks should be run, as it would be best to run any jobs at midnight as this is outside 'MSc Properties' business hours.



```

50 private void setSchedule(int time) {
51     timer.schedule(this, date.getTime(), time);
52 }

```

Fig x – Extract from TaskGenerator class, setSchedule()

As you can see from fig x, the setSchedule() method invoked within the TaskGenerator constructor invokes the Timer.schedule() method, and I supply this instance of TaskGenerator (which will be the object that will then do something when the timer has reached midnight), and the Date object returned from invoking Calendar.getTime() on Calendar variable date, and lastly the time in milliseconds between each task to be generated. Once the timer reaches midnight it will then invoke the method run() which I have had to override within the TaskGenerator class.



```

31 @Override
32 public void run() {
33     System.out.println("Daily Tasks");
34     //server.processRentTransactions();
35     //server.processLeaseTransactions();
36     //server.cloneDatabase();
37     Calendar today = Calendar.getInstance();
38     if(today.get(Calendar.DAY_OF_MONTH) == 1) {
39         //server.generateReports();
40         //server.processSalaryTransactions();
41         System.out.println("Monthly Tasks");
42     }
43 }

```

Fig x – Extract from TaskGenerator class, run()

As previously explained I had to override the method run() from TimerTask class, this is because I have extended the TimerTask class.

As you can see from fig x, within the run method I am carrying out a number of daily transactions such as Server.processTransactions() and also some monthly transactions such as Server.generateReports() which will automate tasks like creating rent transactions or generating monthly reports.