Design Document (PSM)

For Insulin Pump

(Version 1)

Written by:

Morteza Yousef-Sanati

Hamid Mohammad-Gholizadeh

Assignment 3

Course: 703 Software Design

Professor: Dr. Tom Maibaum

April 2011

Table of Contents

PIM to PSM

Before describing of the model, there are some points that we have mentioned below.

1. We selected JAVA as our desired implementation programming language.
2. The types mapping have been mentioned in the table 1.

Type in Model	Desired Type in JAVA
Integer	int
Real, Double	Double
String	String
Array of String	Vector<String>
Bool	boolean

Table 1- Types in Design maps to the Java

3. We didn't have any multiple inheritances therefore we could transfer our model into PSM in this viewpoint without any problems.
4. All setter and getter methods have been added to the PSM in order to have better design and better implementation at the latter step.
5. All association direction displayed in the model and there is no ambiguity in the referencing object to others.
6. All details such as visibilities, methods signatures, and … have been specified completely.
7. In the PSM a sort constraint has been used for some lists.

Design Decisions:

We modified some parts of the design model to get the platform specific model. Class diagram also changed specially the gui package.

Since we needed to simulate the functionality of the buttons of the pump as the real one we needed to assign different functionality to each buttons (ACT, ESC, Bolus, UP and Down) in every status of the pump. E.g. when the user wants to set the basal in the pump the UP button increases the value of a field which has a focus.

Our software architecture is based on three layer architecture: Data, Controller and GUI. Packages are defined based on this structure too. There is an Interface between these modules to **ensure the loose coupling** between them. **Two interfaces named PumpGUI and PumpInterface provide Modula interfaces for the gui and ccontroller packags respectively.** Two classes that implement these two interfaces are MainFram and Pump respectively as you see in figure below:

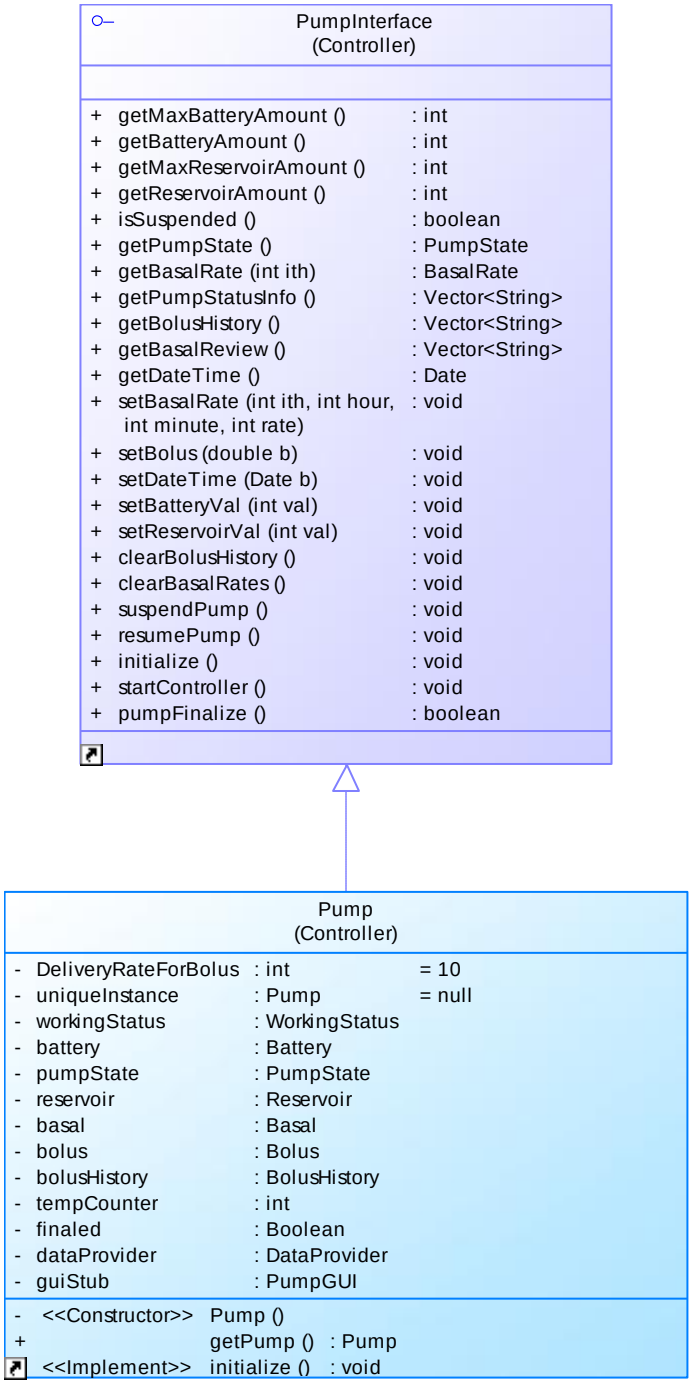

Figure 1 - PumpInterface in controller package. Pump implements this interface

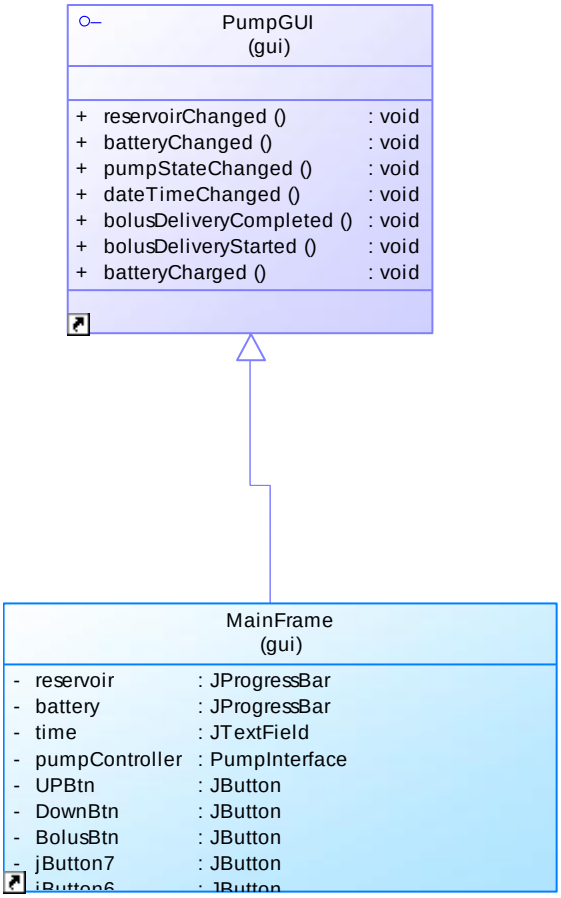

Figure 2- PumpGui Interface is an interface for gui modual. MainFrame implements this interface. Most attributes and operations of Mainframe is omitted in this diagram.

Packages Hierarchy:

We refined the packages structure in gui package. Every functionality of the pump that has interaction with a user we created a package and put the associated commands and screen in that package. For example basal package includes classes like SetBasalScreen, BasalActCommand, BasalUPCommand, BasalDownCommand representing the screen of the setting basal, ACT, UP and Down functionality in basal screen respectively.

When the state of a MainFrame changes to one special state, it loads the corresponding screen to that states and binds the associated commands class to any buttons on the screen properly. So it is insured that the buttons act appropriately in each state. You can see in figure below that gui package contains six other packages as demonstrated in figure below.

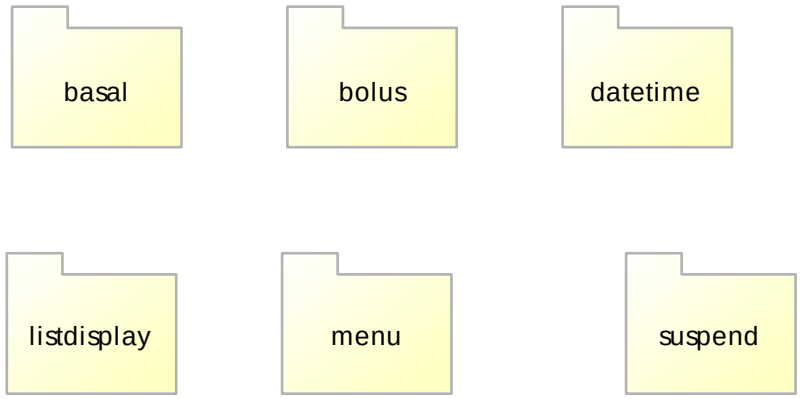

Figure 3- Gui package contains six other packages organizing the classes corresponding to each screen of the pump

Gui Package:

GUI package includes some sub packages which each of them contains classes related to each functionality of pump. E.g. basal package is related to the basal screen and contains the screen class needed as a user interface, as well as corresponding commands for the ACT, UP, Down, ESC and Bolus if it is necessary.

MainFram Different States:

As you see in the fig. 1-1 when the focus is on the minute field the UP button should increase the value of the minute field and when user click the Act button the focus switches to Rate field and now UP button should increase the value of a

Rate field! As you see even **in one screen the functionality of the buttons changes according the state of the screen!** Now consider that we are in another screen, say menus. Up button should scroll the menus up and down button should scroll the menu down. In the case the number of menus is more than the lines of the screen, these buttons should change the appearance of the screen to make the hidden menu items to be available too!

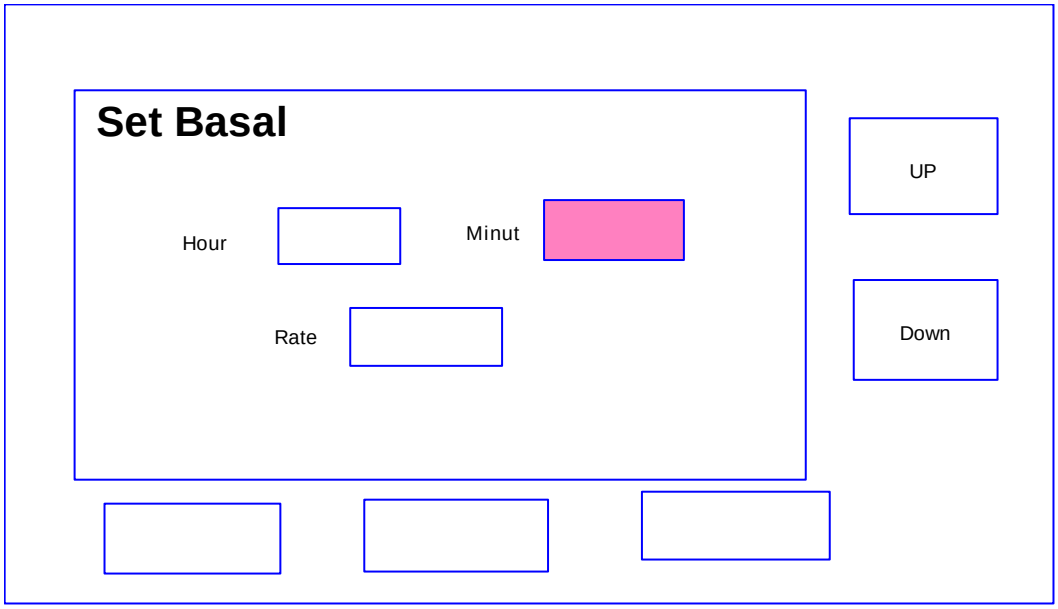

Figure 4- UP and Down Functionality changes according to the State the MainFram is.

To deal with the complexity of the actions of the buttons on the screen we decided to exploit the Command Pattern as we previously stated in our design document.

Figure 5 shows the state chart diagram representing the MainFram class. **The purpose of this statechart is showing how the functionality of each button, as the state of the MainFram changes. GUIStatus Enumeration** represents possible states which MainFram class can be.(Figure 6)

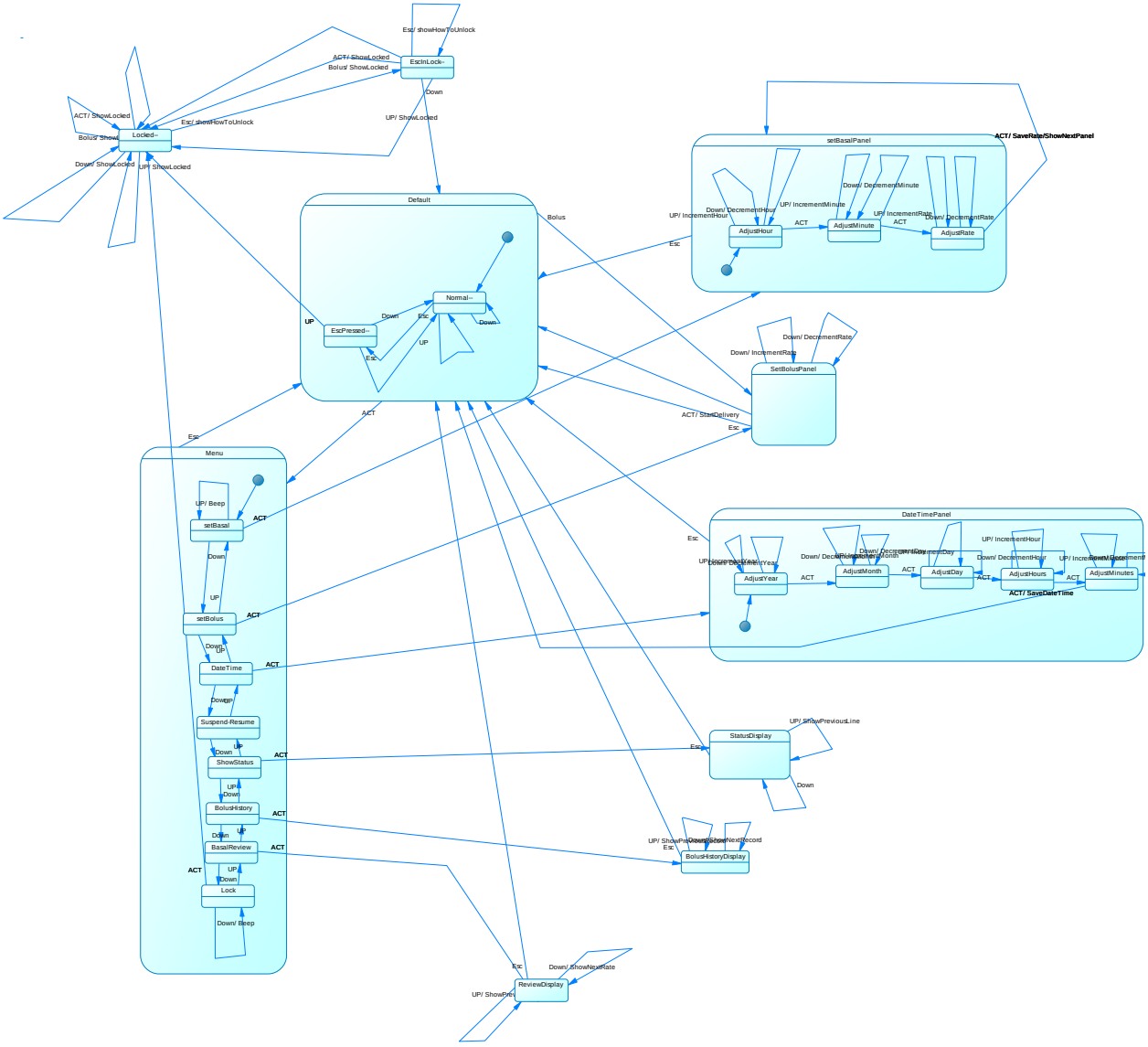

Figure 5 - StateChart diagram representing the state of the MainFram according to the user actions in pressing buttons on the MainFrame.

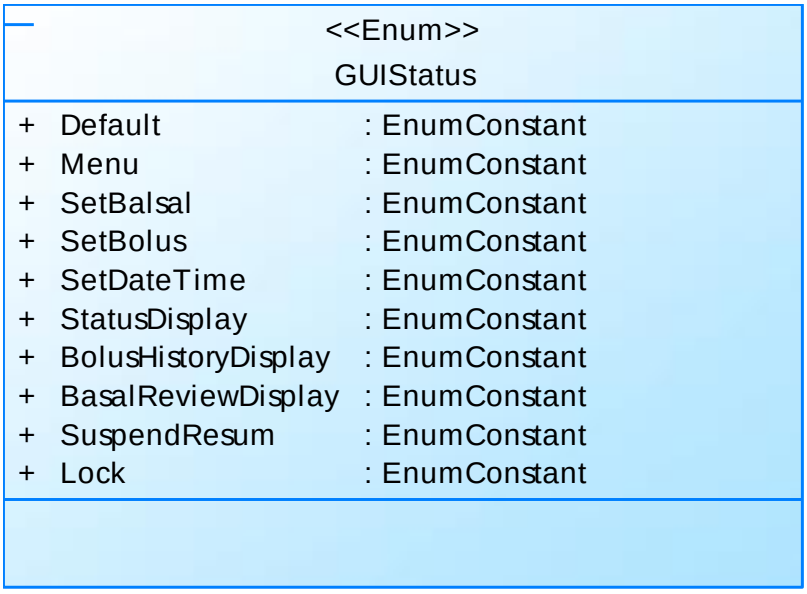

Figure 6 - Enumeration GUIStatus, representing different Status which MainFram can be on.

Command Classes:

Since the functionality of buttons changes in every step we encapsulated that in a command Classes. **Every command class should implement the Command interface,** and when the user clicks any button the execute method of the command object associated with that button in the current state automatically executed as a reaction to the click action.

Menu Design:

We deal with menu as a list of menu Items which should be displayed to the user on the limited screen size. As you see in the Figure 7. We should control the window going up and down using UP and Down buttons. For Controlling the boundries of this window and defining the status of menu window, we draw an statechart for the menuScreen class.(Figure 5)

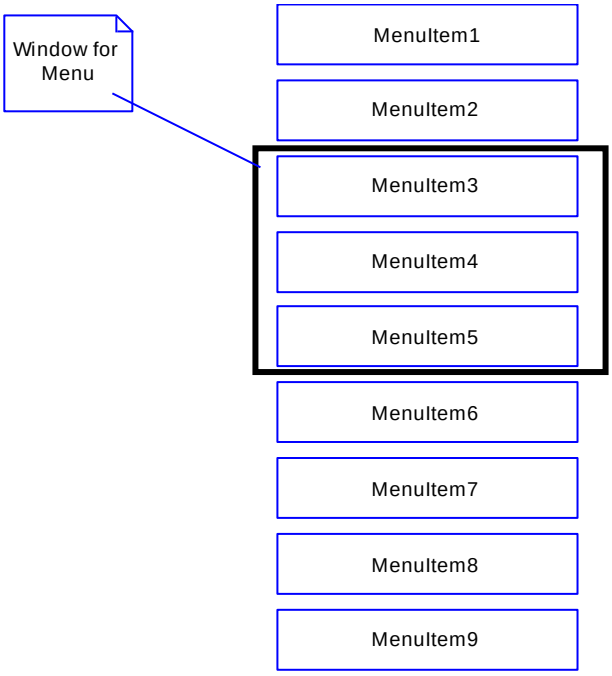

Figure 7 - only the black part is visible to the user, Up and Down buttons control rolling this black window up and down

Three sub states in Figure 5 indicate the state of the black window in in figure 7 is. There should be a control on this window that prevents it going beyond the limits of the menu items.

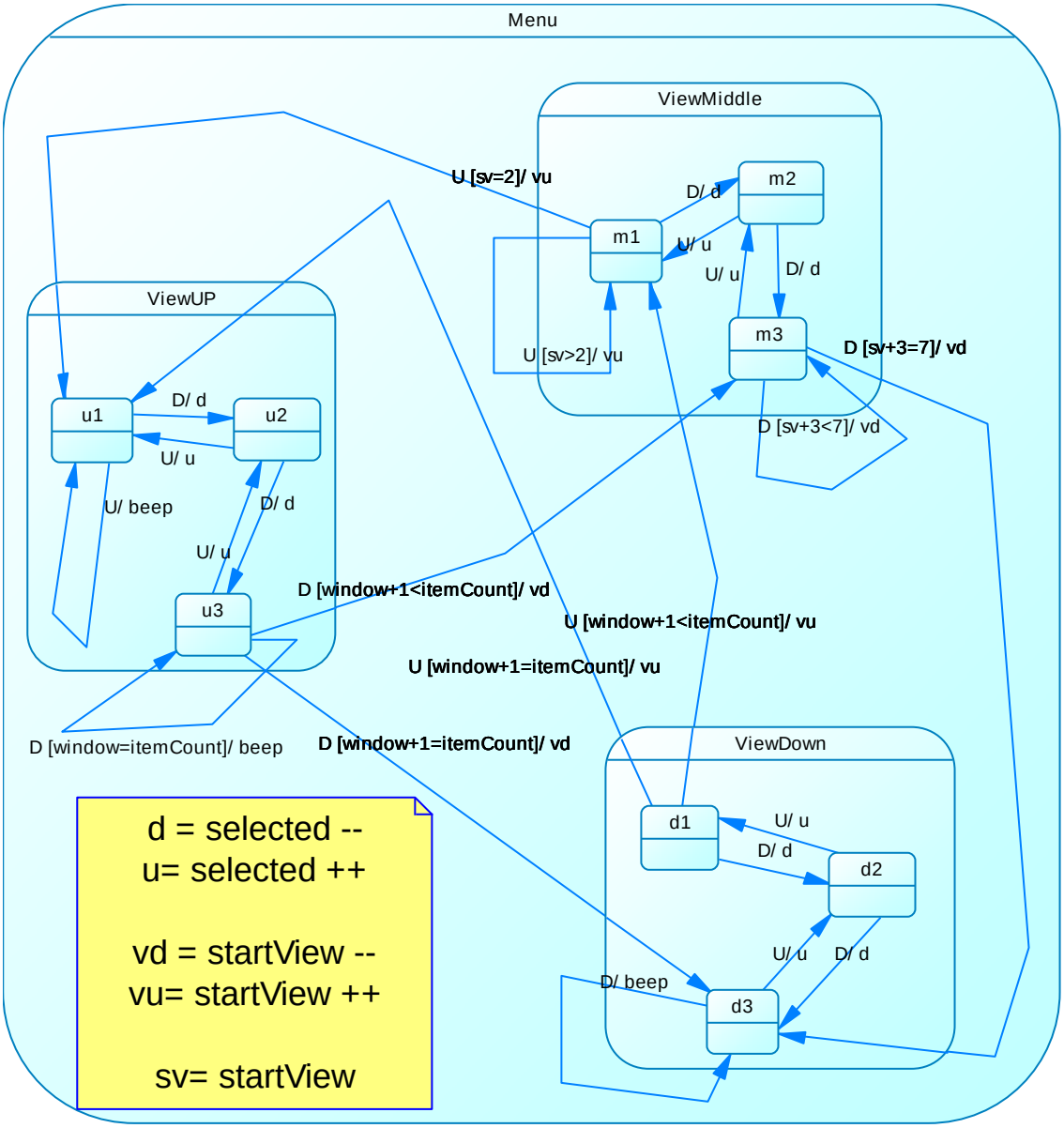

Figure 8 - Showing stats of MenuScreen Class.

Data Provider Package

In this package we have one class, which do all data transactions such as storing and retrieving:

- Battery amount
- Reservoir amount
- Bolus History
- Basal Rates
- Date and Time

This class is a Singleton class therefore we use **Singleton Pattern** to implement it. Therefore, the constructor of this class is private and there is a static function to return an instance of the class and controls the uniqueness of its object.

Also, we use some specific variables' types to deal with file manipulation such as Formatter type and Scanner Type, … .

Controller Package

This package is most important package in our design since all processes are done in it. In this package we have three third which are:

- Timer thread
- Battery thread
- Pump thread

Timer thread

Timer class has many specifications:

1. This class is a thread, which is started at the first execution of the program.
2. The timer is responsible to keep the time and increment the time in each second.
3. The object of this class is singleton and we use Singleton Pattern to implement it.
4. A **major decision** in this class for simulation is **Time Stopping**. It means if the battery finishes the timer will stop and time will be recovered after battery recharges. This decision is made since in real world we cannot do this thing but for convince of TA this feature have been designed.

Battery thread

Battery class is responsible to manage the battery amount for pump activities. Battery class has a singleton object and the Singleton Pattern is also used for this class. On the other hand, battery is decreasing each second to keep pump live. If the battery level fall to zero level pump wouldn't do any activities. Therefore in this case, battery must be changed or in our simulation the level of battery must be increased by the trackbar, which is designed for this case.

In the battery class MaxBatteryAmount is a final static variable that is used for the maximum battery capacity.

Also, as a design decision we have the rate variable which indicates the ration of battery consumption when the pump deliver the insulin to the patient.

Pump thread

The most important thread is the pump thread. The pump manages all activities which are needed by the patient such as Insulin delivery (Bolus, Basal), … .

There are some design decisions, which are:

- The pump implements a specific interface to provide less coupling to the other architecture tier such as GUI.
- In every second, pump checks the base rates and bolus that are should be delivered and if it will deliver the needed insulin amount in the right time.
- Singleton Pattern was used in the design of this class to ensure that one object of this class will be created in the runtime.
- The pump use GUI interface to inform GUI about reservoir change, battery change, start bolus delivery, finish bolus delivery and ….

PSM class Diagram

Gui Package:

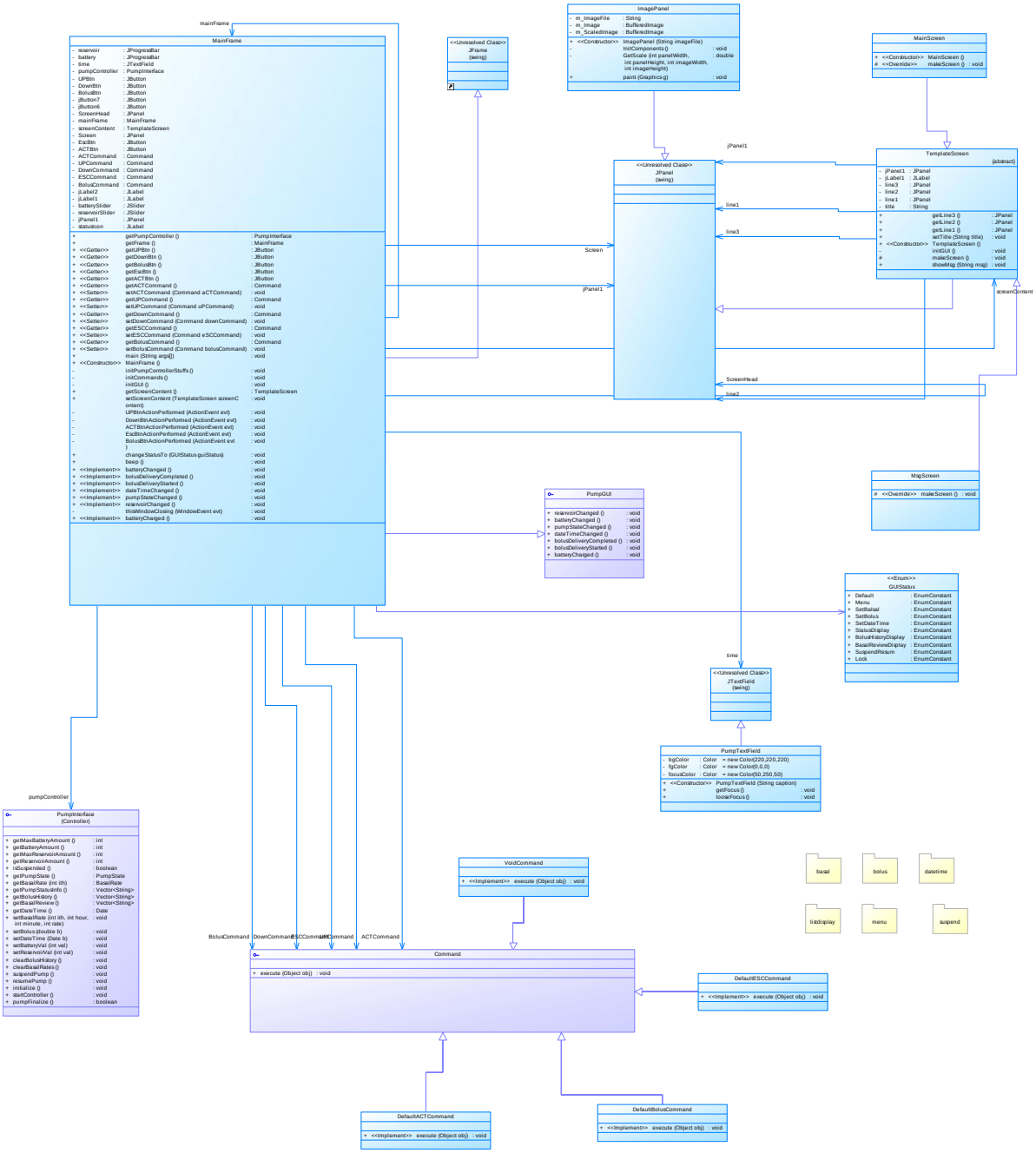

Figure 9 - Gui Package

MainFrame Class:

MainFrame (gui)			
-	reservoir	: JProgressBar	
-	battery	: JProgressBar	
-	time	: JTextField	
-	pumpController	: PumpInterface	
-	UPBtn	: JButton	
-	DownBtn	: JButton	
-	BolusBtn	: JButton	
-	jButton7	: JButton	
-	jButton6	: JButton	
-	ScreenHead	: JPanel	
-	mainFrame	: MainFrame	
-	screenContent	: TemplateScreen	
-	Screen	: JPanel	
-	EscBtn	: JButton	
-	ACTBtn	: JButton	
-	ACTCommand	: Command	
-	UPCommand	: Command	
-	DownCommand	: Command	
-	ESCCommand	: Command	
-	BolusCommand	: Command	
-	jLabel2	: JLabel	
-	jLabel1	: JLabel	
-	batterySlider	: JSlider	
-	reservoirSlider	: JSlider	
-	jPanel1	: JPanel	
-	statusIcon	: JLabel	
+		getPumpController ()	: PumpInterface
+		getFrame ()	: MainFrame
+	<<Getter>>	getUPBtn ()	: JButton
+	<<Getter>>	getDownBtn ()	: JButton
+	<<Getter>>	getBolusBtn ()	: JButton
+	<<Getter>>	getEscBtn ()	: JButton
+	<<Getter>>	getACTBtn ()	: JButton
+	<<Getter>>	getACTCommand ()	: Command
+	<<Setter>>	setACTCommand (Command aCTCommand)	: void
+	<<Getter>>	getUPCommand ()	: Command
+	<<Setter>>	setUPCommand (Command uPCommand)	: void
+	<<Getter>>	getDownCommand ()	: Command
+	<<Setter>>	setDownCommand (Command downCommand)	: void
+	<<Getter>>	getESCCommand ()	: Command
+	<<Setter>>	setESCCommand (Command eSCCommand)	: void
+	<<Getter>>	getBolusCommand ()	: Command
+	<<Setter>>	setBolusCommand (Command bolusCommand)	: void
+		main (String args[])	: void
+	<<Constructor>>	MainFrame ()	
-		initPumpControllerStuffs ()	: void
-		initCommands ()	: void
-		initGUI ()	: void
+		getScreenContent ()	: TemplateScreen
+		setScreenContent (TemplateScreen screenC ontent)	: void
-		UPBtnActionPerformed (ActionEvent evt)	: void
-		DownBtnActionPerformed (ActionEvent evt)	: void
-		ACTBtnActionPerformed (ActionEvent evt)	: void
-		EscBtnActionPerformed (ActionEvent evt)	: void
-		BolusBtnActionPerformed (ActionEvent evt)	: void
+		changeStatusTo (GUIStatus guiStatus)	: void
+		beep ()	: void
+	<<Implement>>	batteryChanged ()	: void
+	<<Implement>>	bolusDeliveryCompleted ()	: void
+	<<Implement>>	bolusDeliveryStarted ()	: void
+	<<Implement>>	dateTimeChanged ()	: void
+	<<Implement>>	pumpStateChanged ()	: void
+	<<Implement>>	reservoirChanged ()	: void
-		thisWindowClosing (WindowEvent evt)	: void
+	<<Implement>>	batteryCharged ()	: void

Figure 10 - MainFrame Class

PumpGUI interface

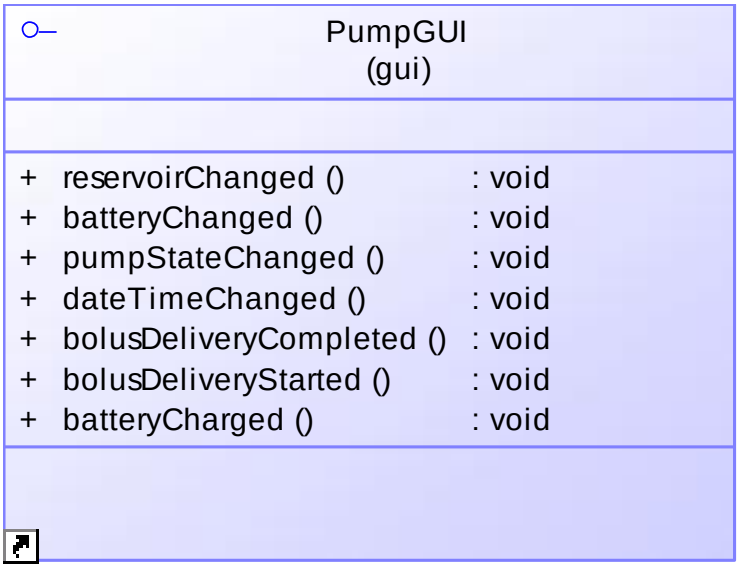

Figure 11 - PumpGUI interface

TemplateScreen Class:

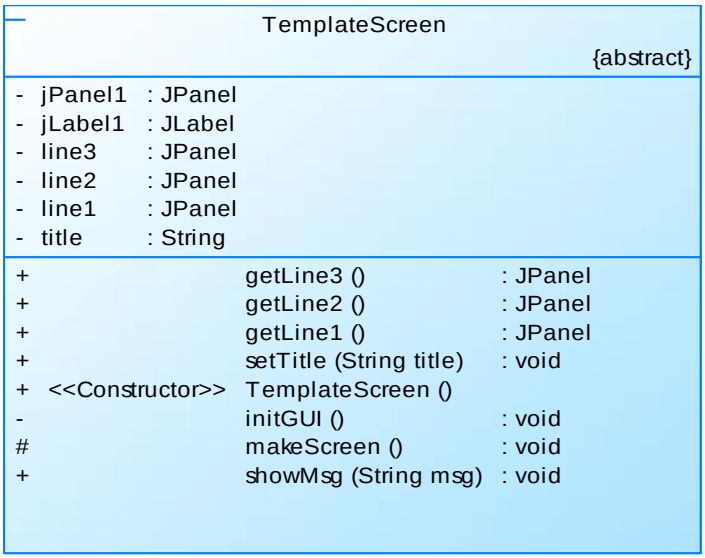

Figure 12 - TemplateScreen Class

Basal Package Class Diagram:

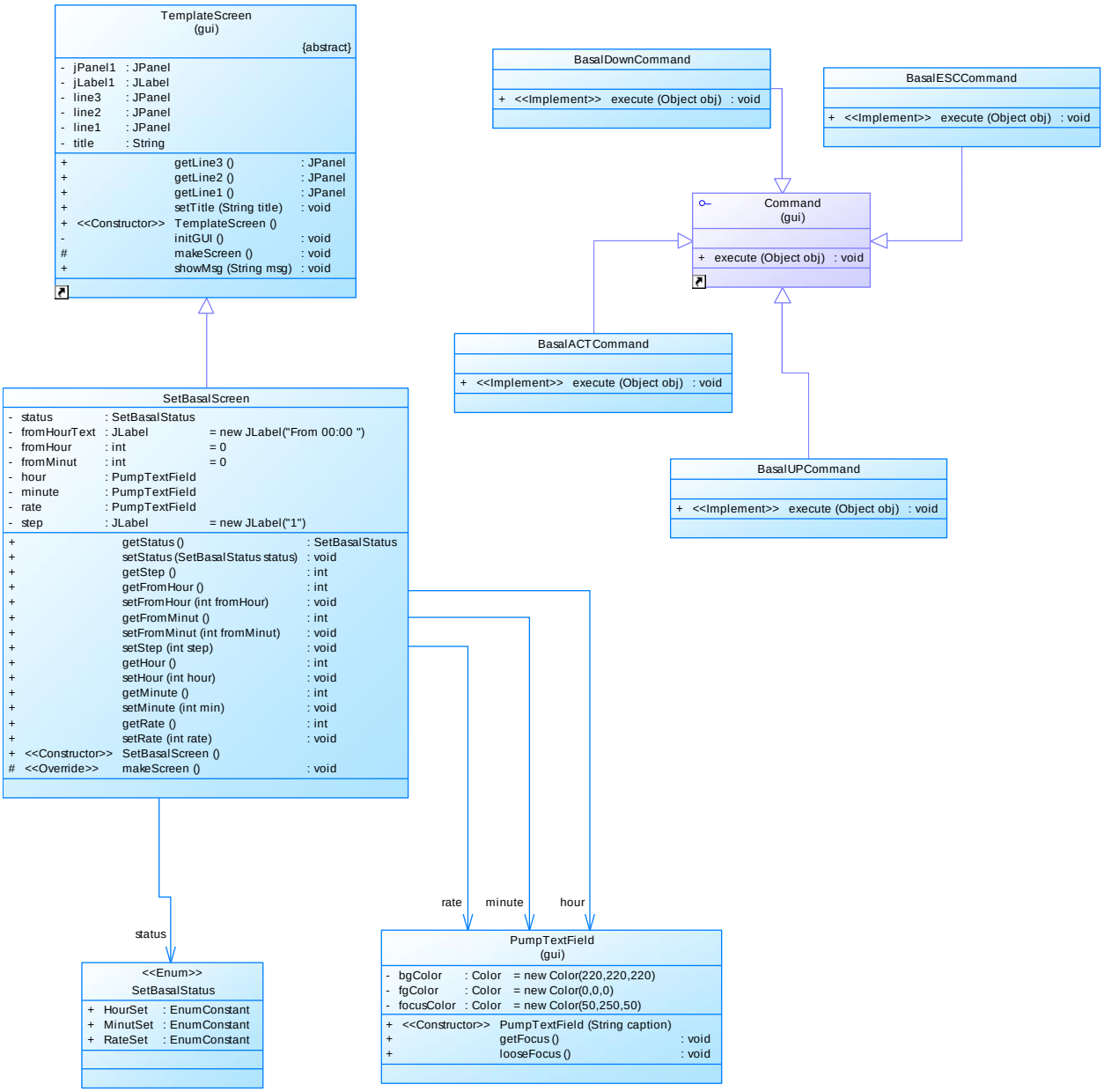

Figure 13 - Basal Package Class Diagram

Bolus Package Class Diagram:

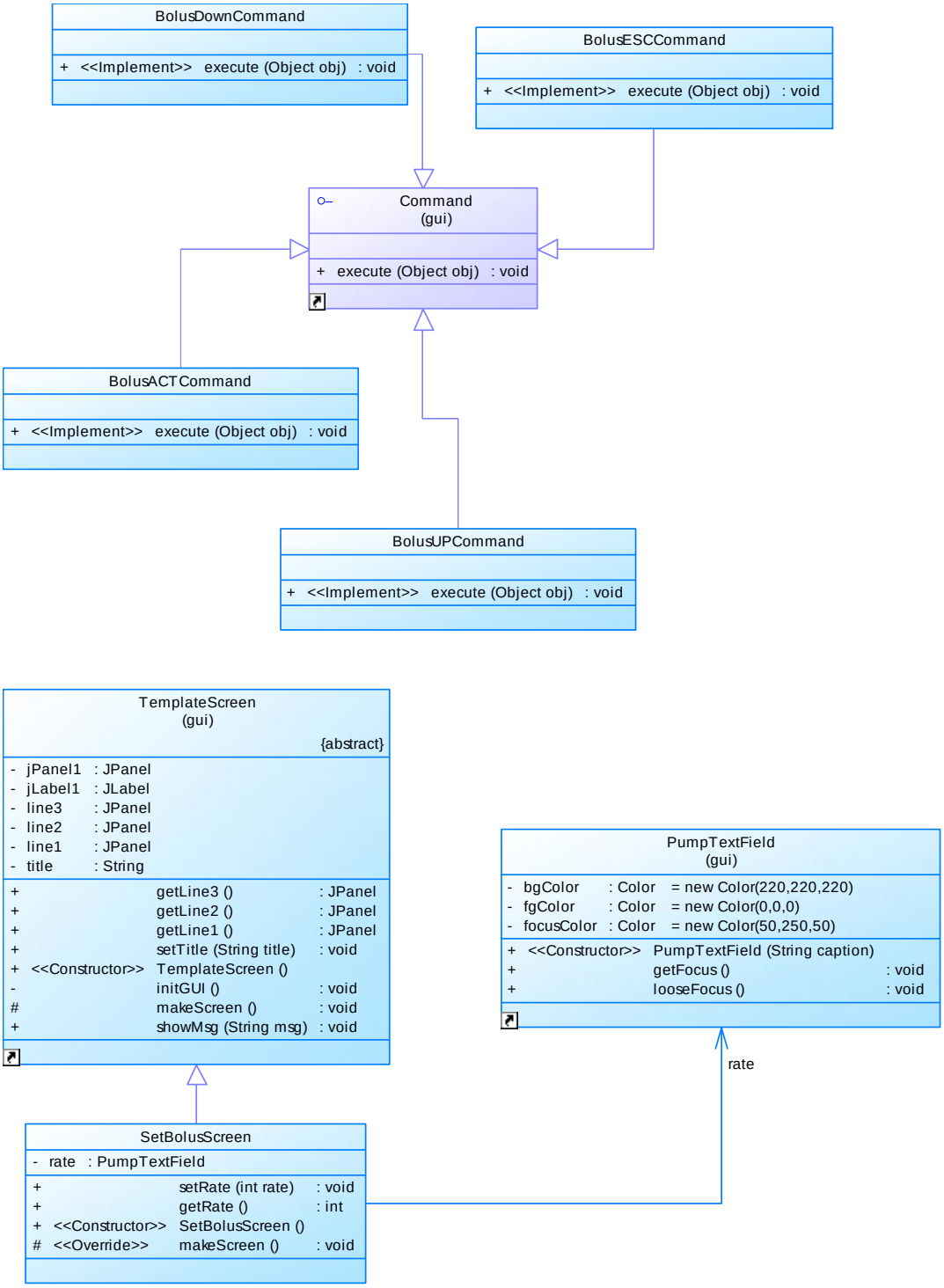

Figure 14 - Bolus Package Class Diagram

DateTime Package Class Diagram:

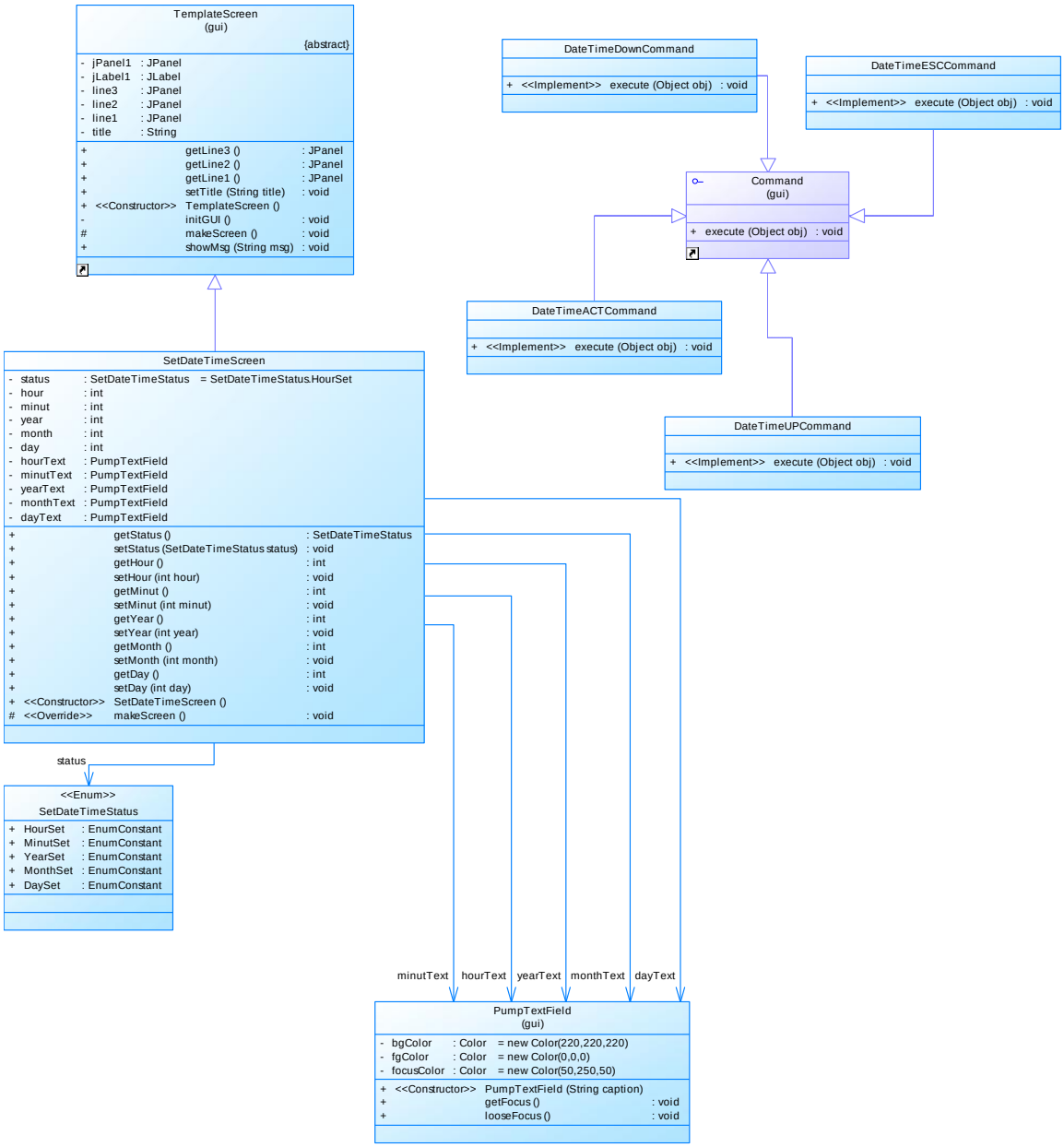

Figure 15 - DateTime Package Class Diagram

ListDisplay Package Class Diagram:

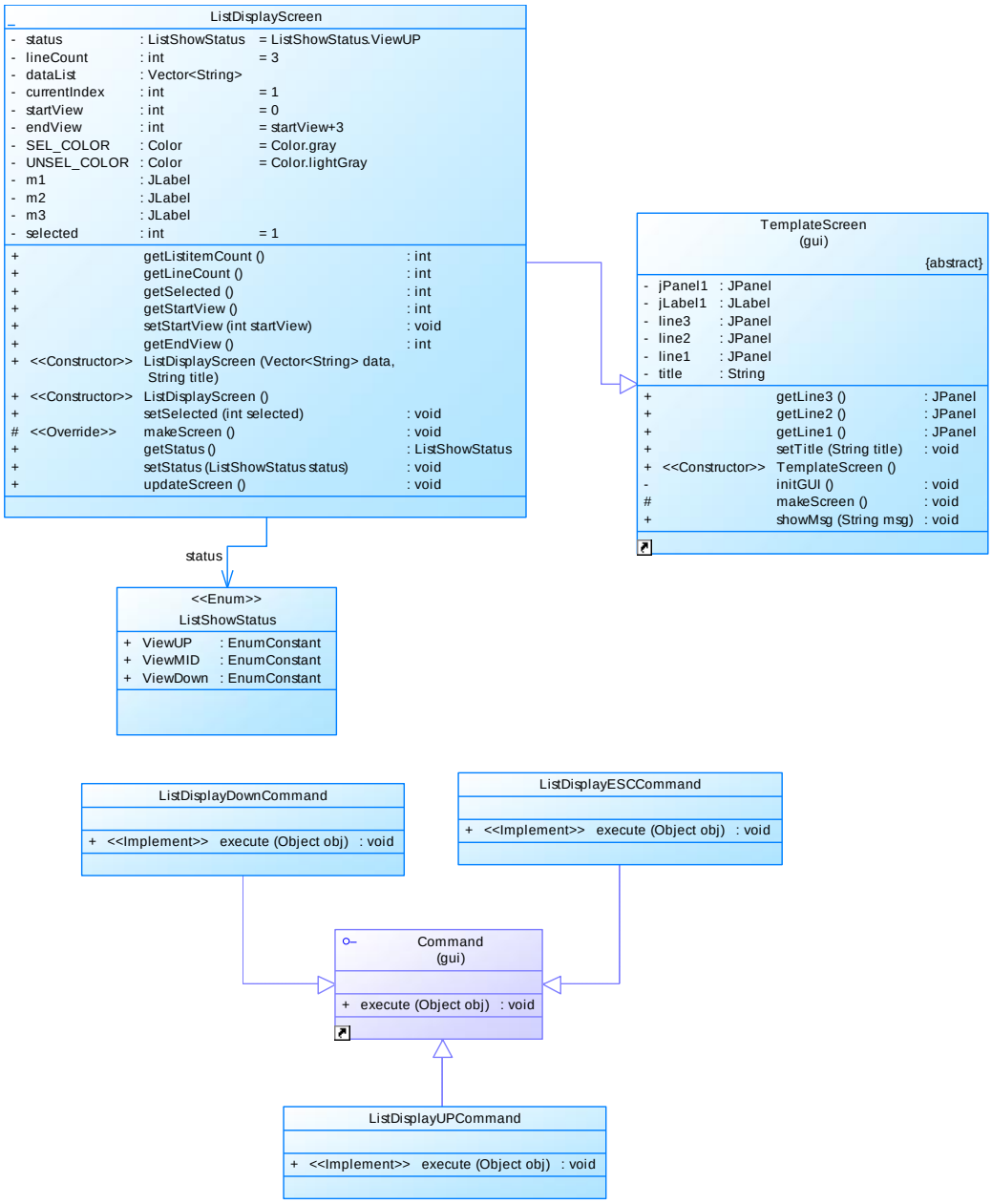

Figure 16 - ListDisplay Package Class Diagram

Menu Package Class Diagram:

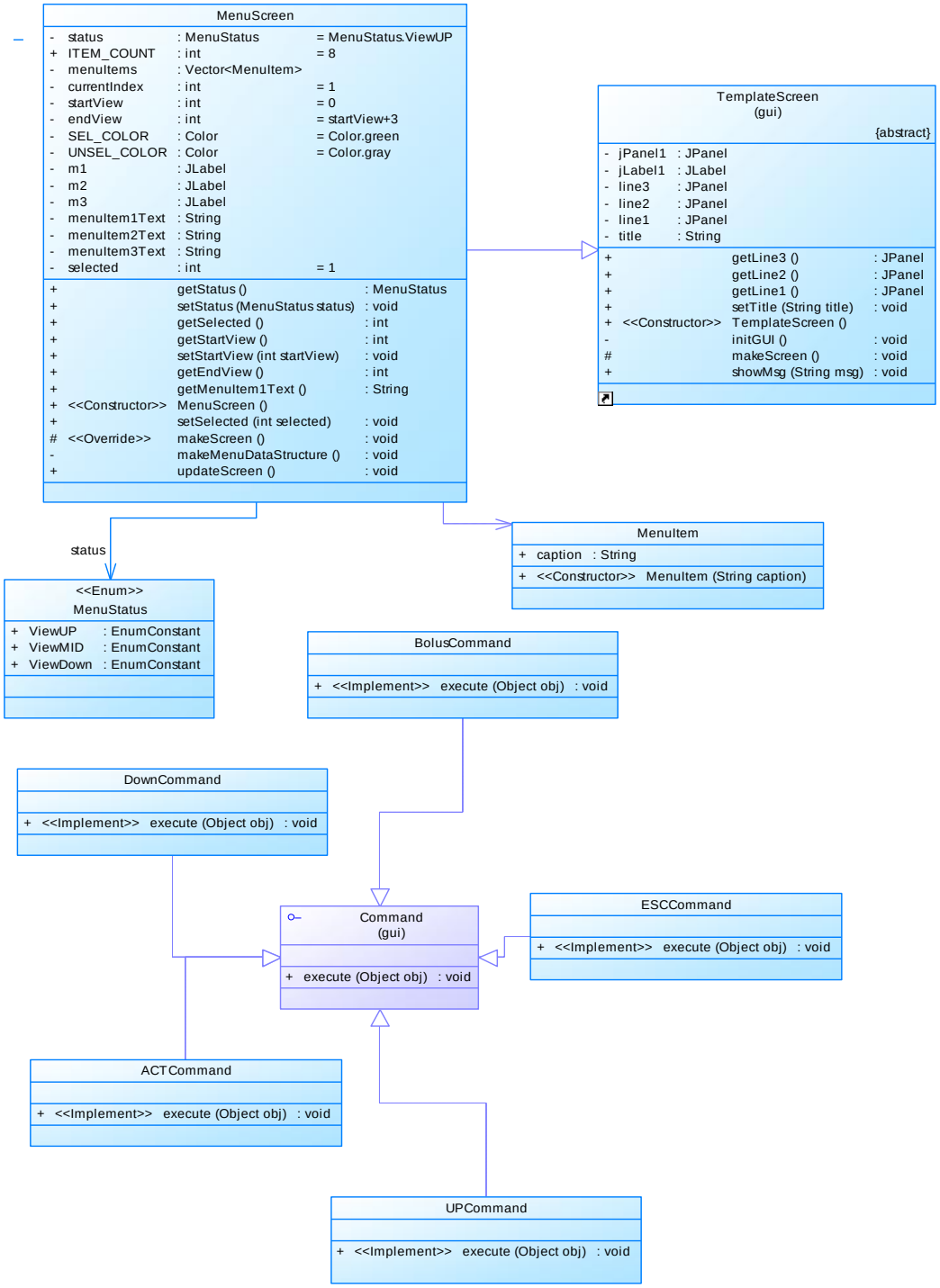

Figure 17 - Menu Package Class Diagram

Suspend Package Class Diagram:

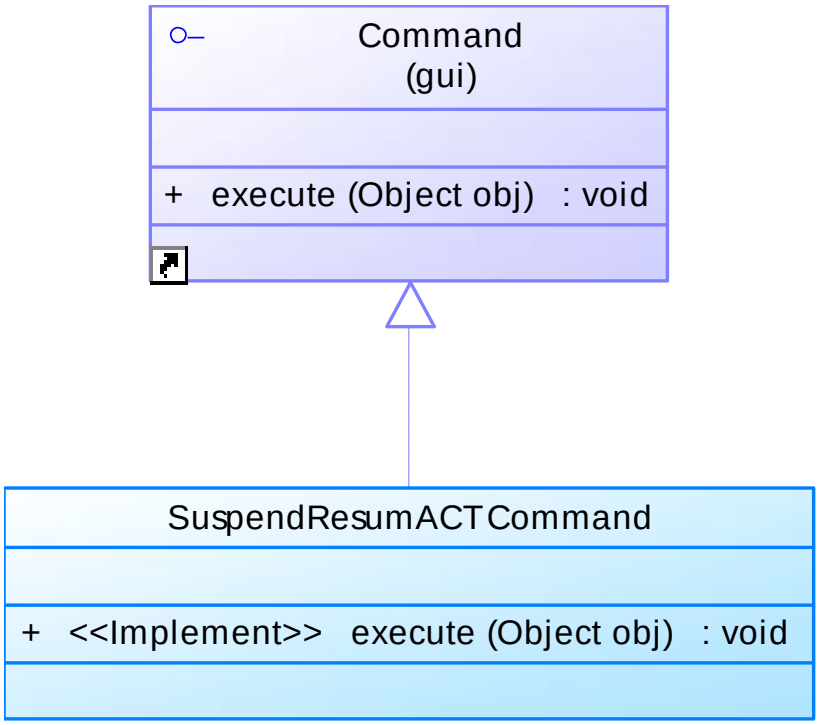

Figure 18 - Suspend Package Class Diagram

Controller Package Class Diagram:

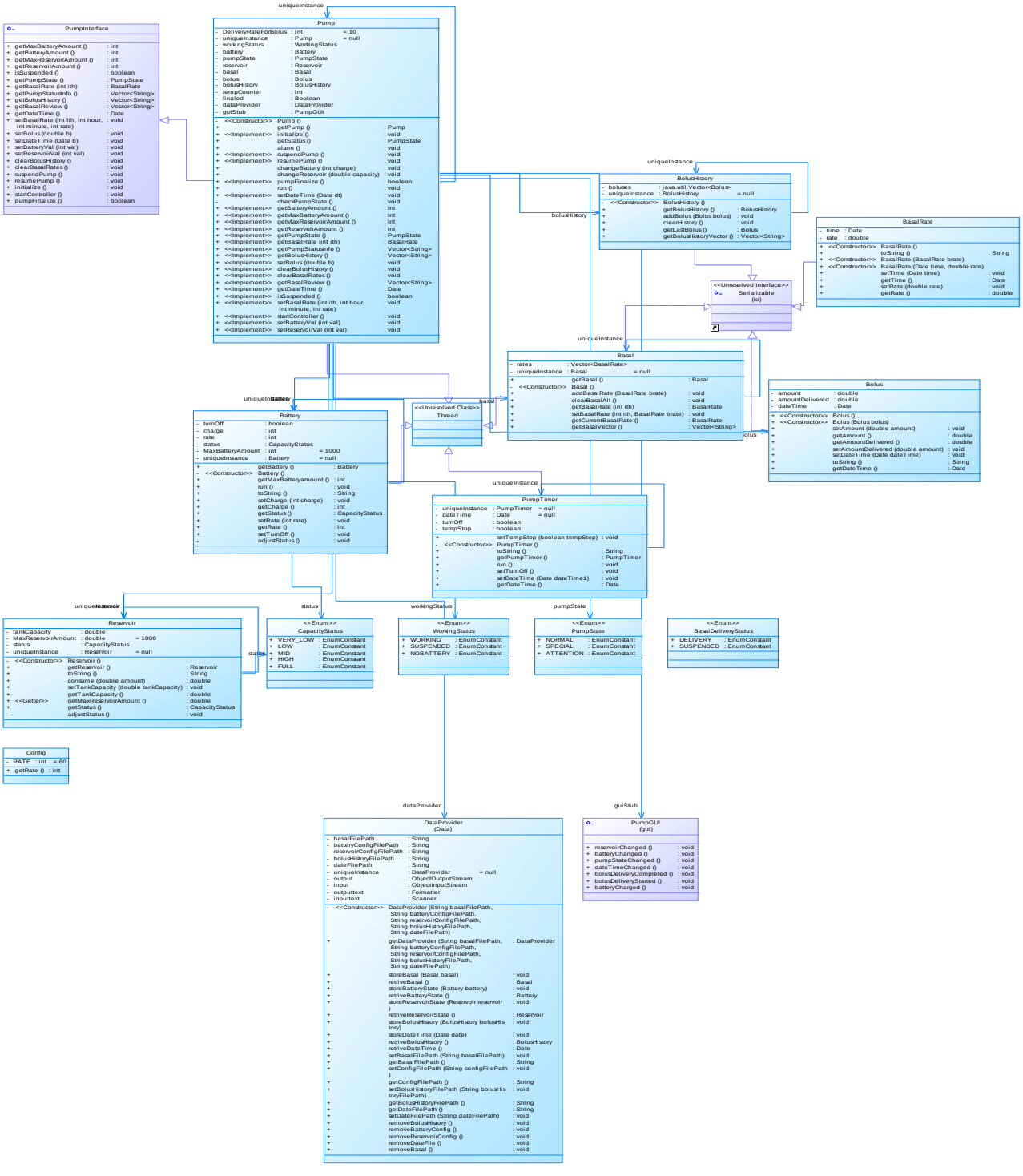

Figure 19 - Controller Package Class Diagram

Data Package Class Diagram:

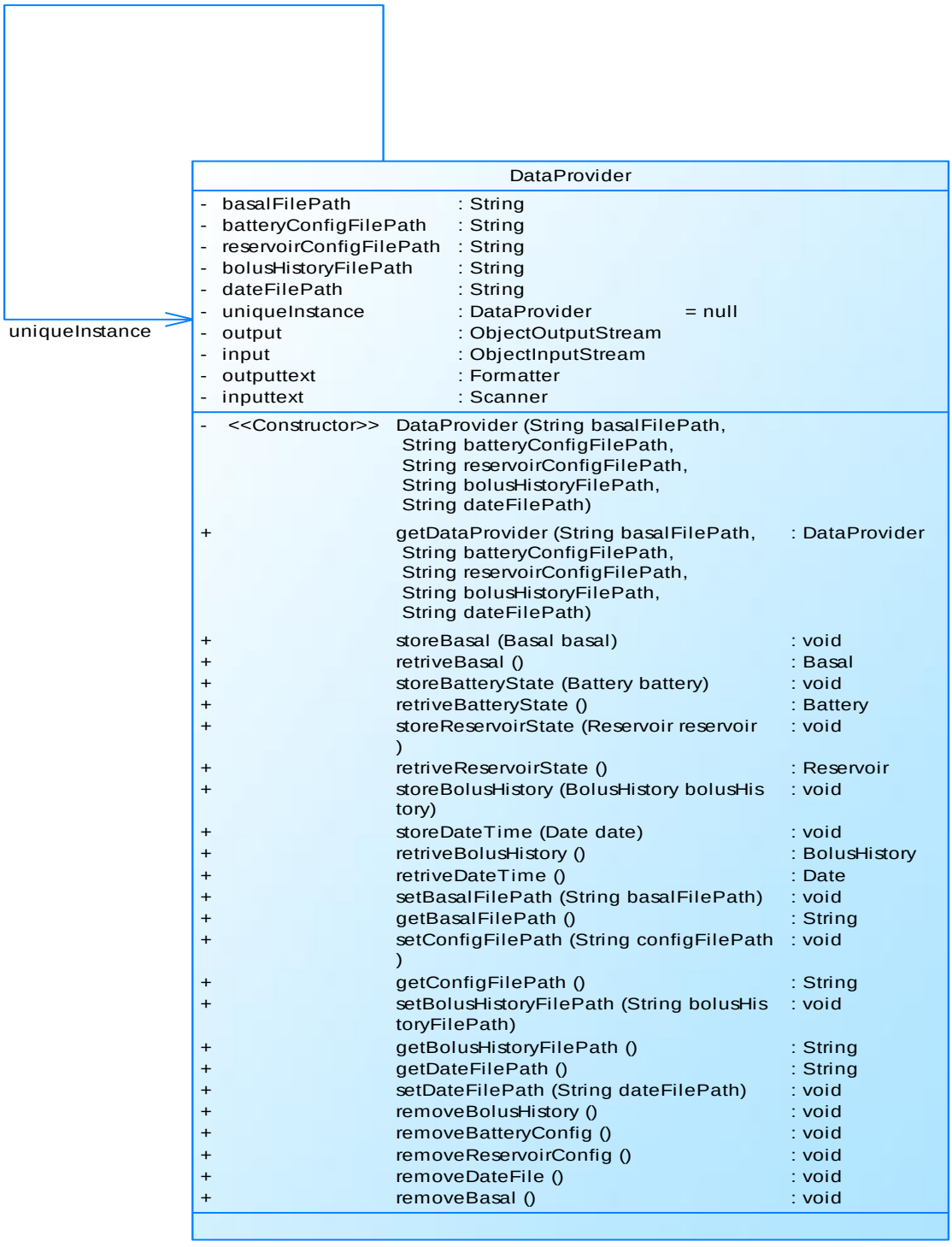

Figure 20 - Data Package Class Diagram