Design Patterns

1. Observer Pattern

The following interface (GPCalendarListener) represents an Observer Pattern because the subject notifies a list of observers about a change that occurred so that the observers can change their status.

*/\*\*  
 \* Calendar listeners are notified when calendar is changed, namely,  
 \* when weekends days change or holidays list change.  
 \*   
 \** ***@author*** *dbarashev (Dmitry Barashev)  
 \*/*public interface GPCalendarListener {  
 void onCalendarChange();  
}

Path: biz/ganttproject/core/calendar/GPCalendarListener.java

1. State Pattern

The class “WorkingUnitCounter” represents a state pattern because allows the object to alter its behaviour based on its state. In this case, the state is represented by the Boolean variable “isMoving”:

public class WorkingUnitCounter extends ForwardTimeWalker {  
 private Date myEndDate;  
 private boolean isMoving = true;  
 private int myWorkingUnitCounter;  
 private int myNonWorkingUnitCounter;  
  
 public WorkingUnitCounter(GPCalendarCalc calendar, TimeUnit timeUnit) {  
 super(calendar, timeUnit);  
 }  
  
 @Override  
 protected boolean isMoving() {  
 return isMoving;  
 }  
  
 @Override  
 protected void processNonWorkingTime(Date intervalStart, Date workingIntervalStart) {  
 myNonWorkingUnitCounter++;  
 isMoving = workingIntervalStart.before(myEndDate);  
 }  
  
 @Override  
 protected void processWorkingTime(Date intervalStart, Date nextIntervalStart) {  
 myWorkingUnitCounter++;  
 isMoving = nextIntervalStart.before(myEndDate);  
 }

Path: biz/ganttproject/core/calendar/walker/WorkingUnitCounter.java

1. Bridge

In the following two classes can be found a “Bridge” design pattern as both of them implements the same abstract class in a different way:

Abstract class: abstract class GPCalendarBase implements GPCalendarCalc {

Path: biz/ganttproject/core/calendar/GPCalendarBase.java

Class 1:

public class WeekendCalendarImpl extends GPCalendarBase implements GPCalendarCalc {

public List<GPCalendarActivity> getActivities(Date startDate, final Date endDate) {  
 if (getWeekendDaysCount() == 0 && myOneOffEvents.isEmpty() && myRecurringEvents.isEmpty()) {  
 return myRestlessCalendar.getActivities(startDate, endDate);  
 }  
 List<GPCalendarActivity> result = new ArrayList<GPCalendarActivity>();  
 Date curDayStart = myFramer.adjustLeft(startDate);  
 boolean isWeekendState = (getDayMask(curDayStart) & DayMask.*WORKING*) == 0;  
 while (curDayStart.before(endDate)) {  
 Date changeStateDayStart = doFindClosest(curDayStart, myFramer, MoveDirection.*FORWARD*,  
 isWeekendState ? DayType.*WORKING* : DayType.*NON\_WORKING*, endDate);  
 if (changeStateDayStart == null) {  
 changeStateDayStart = endDate;  
 }  
 if (changeStateDayStart.before(endDate) == false) {  
 result.add(new CalendarActivityImpl(curDayStart, endDate, !isWeekendState));  
 break;  
 }  
 result.add(new CalendarActivityImpl(curDayStart, changeStateDayStart, !isWeekendState));  
 curDayStart = changeStateDayStart;  
 isWeekendState = !isWeekendState;  
 }  
 return result;  
}

Path: biz/ganttproject/core/calendar/WeekendCalendarImpl.java

Class 2:

public class AlwaysWorkingTimeCalendarImpl extends GPCalendarBase implements GPCalendarCalc {  
 @Override  
 public List<GPCalendarActivity> getActivities(Date startDate, Date endDate) {  
 return Collections.*singletonList*((GPCalendarActivity) new CalendarActivityImpl(startDate, endDate, true));  
 }

Path: biz/ganttproject/core/calendar/AlwaysWorkingTimeCalendarImpl.java

Martin Magdalinchev Nº 58172 MIEI