Sample Solution for EXERCISE 1

The UML class diagram shown in Fig. 4 provides a sample solution for Exercise 1. Only relevant classes, focusing on the Observer Pattern and the instruments, are shown. First, we added the application domain classes described in the exercise.We created a new class InstrumentPanel, which consists of many Instruments. The class Instrument is abstract and generalizes all concrete instrument classes like the Compass, Speedometer or the GPS. The Instrument class reduces the amount of associations between the InstrumentPanel and the concrete instruments.

Otherwise, the InstrumentPanel would have an association to each concrete instrument, and new associations are required when adding new instruments. We applied the Observer Pattern to notify the instruments of space shuttle changes. The SpaceShuttle becomes the Publisher as it provides the information to present. The Instrument class becomes the Subscriber. Subscriber instances can be subscribed and unsubscribed on the Publisher and, thus, on the SpaceShuttle.

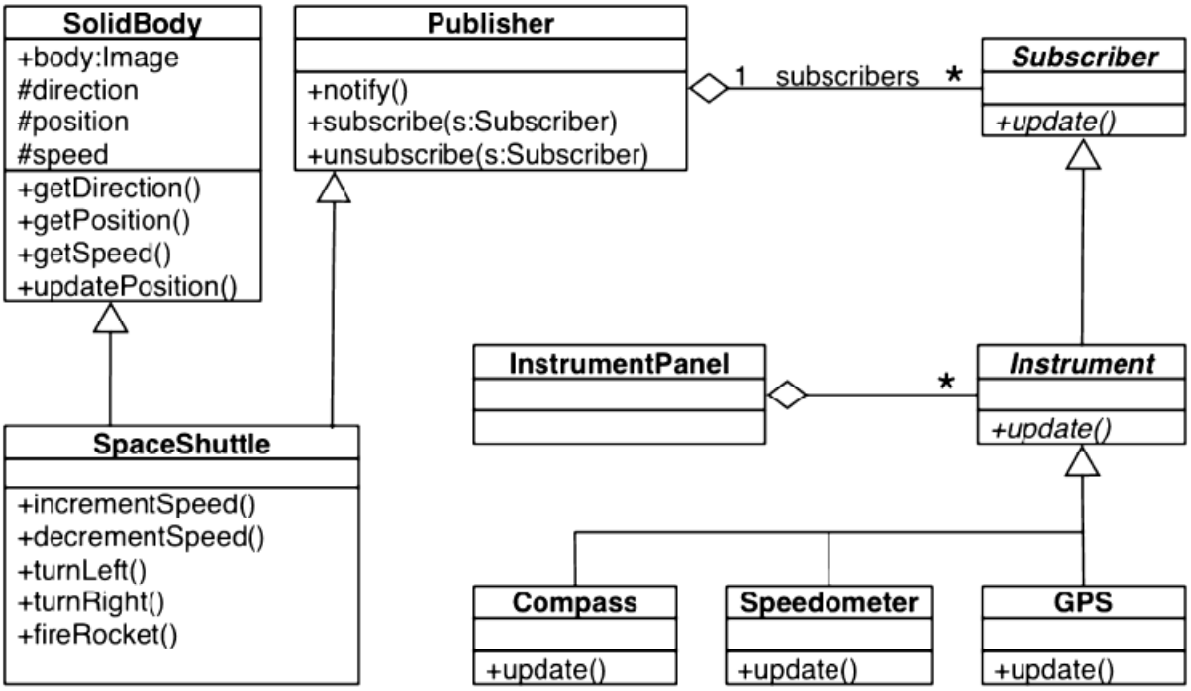


Figure 4 Class model including the instruments and the Observer Pattern

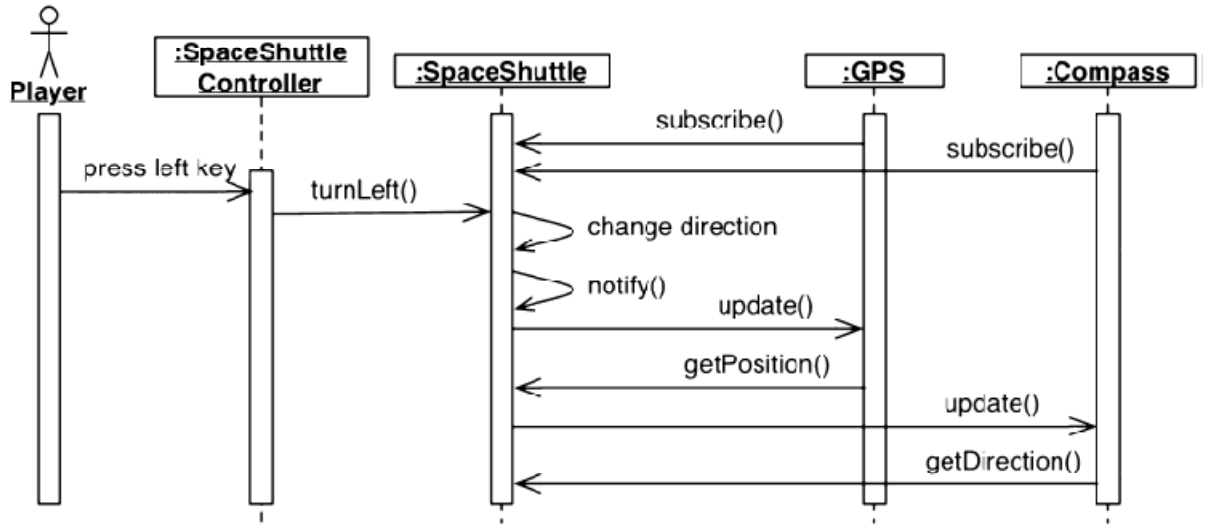


**Figure 5. Interaction between the space shuttle and its instruments**

The notify() operation of the Publisher invokes the update() operation of all subscribed instruments to announce changes of the SpaceShuttle. All concrete instruments implement the abstract operation update(). They retrieve the required information from the SpaceShuttle and update their visualization whenever the update () operation is invoked. Fig. 5 illustrates the interaction as a UML sequence diagram. The provided design is extensible, because no changes on the SpaceShuttle, InstrumentPanel, or Instrument class are needed when adding new concrete instruments. Note that the design uses multiple inheritance for the SpaceShuttle, which is not supported in all programming languages. We provided an implementation-independent design that explains the concept. The concept can be realized in any programming language. The implementation must not exactly reflect the class structure of the design.

Observer Pattern Implementation in Java

Implement the class Compass, showing the direction of the space shuttle. The Compass must extend the Instrument class and must be added to the InstrumentPanel. Subscribe the Compass to the space shuttle to receive notifications of the direction changes from the space shuttle.

The design from Exercise 1 uses multiple inheritance for the SpaceShuttle class. Java does not support multiple inheritance. Therefore, we merge the Publisher class into the SpaceShuttle class by adding the required methods.

```
public class SpaceShuttle extends SolidBody {
    private ArrayList<SpaceShuttleSubscriber> subscribers;
    :
    :
    public void incrementSpeed () {...}
    public void decrementSpeed () {...}
    public void turnRight () {...}
    public void turnLeft () {...}
    public void fireRocket () {...}
    :
    :
    protected void setSpeed(int speed) {
    super.setSpeed(speed);
    notifySpaceShuttleSubscribers ();
    }
    protected void setDirection(int direction) {
    super.setDirection(direction);
    notifySpaceShuttleSubscribers ();
    }
    public void subscribe(SpaceShuttleSubscriber subscriber)
    {...}
    public void unsubscribe(SpaceShuttleSubscriber subscriber)
    {...}
    public synchronized void notifySpaceShuttleSubscribers ()
    {...}
}
```

Note the differences between design and its implementation. In the design we added the class Subscriber to show that we use the Observer Pattern. We realize the Subscriber class with a Java interface called SpaceShuttleSubscriber. The name points out that the subscribers can subscribe on a space shuttle and increases the readability of the source code. The SpaceShuttle class just knows the interface SpaceShuttleSubscriber, which can be any implementing class. The SpaceShuttle class needs no modifications if new concrete SpaceShuttle Subscriber classes are added.

```
public interface SpaceShuttleSubscriber {
    void update ();
```

```
}
```

The abstract Java class Instrument implements the SpaceShuttle Subscriber and extends the class JPanel, a graphical container of Swing. It provides a protected instance variable that concrete subclasses can use to retrieve the space shuttle information to display. The concrete subclasses like the GPS or the Speedometer can add any graphical Swing components to visualize information of the SpaceShuttle. The update () operation gets invoked whenever the attributes of the SpaceShuttle gets changed.

```
public abstract class Instrument extends JPanel implements
    SpaceShuttleSubscriber {
        protected SpaceShuttle spaceshuttle;
        public Instrument(SpaceShuttle spaceshuttle) {
        this.spaceshuttle = spaceshuttle;
    }
    public abstract void update ();
}
```

The class InstrumentPanel is also a graphical Swing container, which contains all instruments. The InstrumentPanel creates the instrument instances and subscribes them to the SpaceShuttle. The following code extract shows the initialization of the instruments GPS and Speedometer.

```
public class InstrumentPanel extends JToolBar {
    :
    :
    public InstrumentPanel(SpaceShuttle theSpaceShuttle) {
        super(JToolBar.VERTICAL);
        setFloatable(false);
        this.spaceshuttle = theSpaceShuttle;
        :
        :
        speedometer = new Speedometer(theSpaceShuttle);
        theSpaceShuttle.subscribe(speedometer);
        add(speedometer);
        gps = new GPS(theSpaceShuttle);
        theSpaceShuttle.subscribe(gps);
        add(gps);
        :
        :
    }
    :
    :
}
```