Port me if you can

Using ports & adapters

Jürg Weilenmann, 13.07.2021

Problem

```
class EreignisprotokollServiceOldTest {

    @Test
    void AuslenkungUebersteuertEreignis() {
        final Vorgangstracker vorgangstracker = mock(Vorgangstracker.class);
        final Prozessinformation prozessinformation = mock(Prozessinformation.class);
        final SharkDataproviderContext dataproviderContext = mock(SharkDataproviderContext.class, RETURNS_DEEP_STUBS);
        final BOFactory boFactory = mock(BOFactory.class);
        final SharkBOManager sharkBOManager = mock(SharkBOManager.class);
        final ProzessinformationBO prozessinformationBO = mock(ProzessinformationBO.class);
        final VorgangBO vorgangBO = mock(VorgangBO.class);
        when(vorgangstracker.getProzessinformation()).thenReturn(prozessinformation);
        when(dataproviderContext.getBOFactory()).thenReturn(boFactory);
        when(boFactory.newBOInstance(Ereignisprotokoll.class)).thenReturn(mock(Ereignisprotokoll.class));
        when(dataproviderContext.getBOs()).thenReturn(sharkBOManager);
        when(sharkBOManager.getProzessinformationBO()).thenReturn(prozessinformationBO);
        when(sharkBOManager.getVorgangBO()).thenReturn(vorgangBO);
        when(vorgangBO.loadVorgangstrackerNonNull(any())).thenReturn(vorgangstracker);
        when(dataproviderContext.getBOFactory().newBOInstance(AusfuehrungsEintragRef.class)).thenReturn(mock(AusfuehrungsEintragRef.class));
        when(dataproviderContext.getPersistenceManager().getObjectById(eq(AusfuehrungsEintrag.class), any())).thenReturn(
            mock(AusfuehrungsEintrag.class));

        final List<UebersteuerterFehler> uebersteuerterFehler = Arrays.asList(
            new UebersteuerterFehler(new AusfuehrungseintragIdentity("fehler-1"), new RuleNumber("rule-1")),
            new UebersteuerterFehler(new AusfuehrungseintragIdentity("fehler-2"), new RuleNumber("rule-2"))
        );
        final AuslenkungUebersteuertEreignis ereignis = AuslenkungUebersteuertEreignis.builder()
            .vorgangstrackerId(new VorgangstrackerId("vorgangstracker-id"))
            .guiStepKey(GuiStepKeyEnum.ANZEIGE_AUSLENKUNGSDATEN)
            .uebersteuerterFehler(uebersteuerterFehler)
            .build();

        final EreignisprotokollServiceImpl ereignisprotokollService = new EreignisprotokollServiceImpl(dataproviderContext);

        ereignisprotokollService.protokolliereAuslenkungUebersteuertEreignis(ereignis);

        final ArgumentCaptor<Ereignisprotokoll> captureEreignisprotokoll = ArgumentCaptor.forClass(Ereignisprotokoll.class);
        verify(vorgangstracker.getProzessinformation()).addEreignisprotokoll(captureEreignisprotokoll.capture());
        final Ereignisprotokoll ereignisprotokoll = captureEreignisprotokoll.getValue();
        verify(ereignisprotokoll).setEreignis(Ereignistyp.AUSLENKUNG_UEBERSTEUERT.name());
        verify(ereignisprotokoll).setEreigniszeitpunkt(any());
        verify(ereignisprotokoll).setGui(GuiStepKeyEnum.ANZEIGE_AUSLENKUNGSDATEN.getName());
        verify(ereignisprotokoll).setAusloesenderUser("TU_shark2");
        ArgumentCaptor<AusfuehrungsEintragRef> captureAusfuehrungsEintragRef = ArgumentCaptor.forClass(AusfuehrungsEintragRef.class);
        verify(ereignisprotokoll, times(2)).addAusfuehrungsEintragRef(captureAusfuehrungsEintragRef.capture());
        List<AusfuehrungsEintragRef> erwarteteAusfuehrungsEintragRef = captureAusfuehrungsEintragRef.getAllValues();
        assertThat(erwarteteAusfuehrungsEintragRef, hasSize(2));
        verify(erwarteteAusfuehrungsEintragRef.get(0)).setActingRuleNumber("rule-1");
        verify(erwarteteAusfuehrungsEintragRef.get(1)).setActingRuleNumber("rule-2");
        verifyNoMoreInteractions(ereignisprotokoll);
    }
}
```

mock hell

Problem

```
final EreignisprotokollServiceImpl ereignisprotokollService = new EreignisprotokollServiceImpl(dataproviderContext);

final IDarkDataproviderContext dataproviderContext = mock(IDarkDataproviderContext.class, RETURNS_DEEP_STUBS);
final VorgangsTracker vorgangsTracker = mock(VorgangsTracker.class);
final ProzessInformation prozessInformation = mock(ProzessInformation.class);
final BOFactory boFactory = mock(BOFactory.class);
final IDarkBOManager iDarkBOManager = mock(IDarkBOManager.class);
final ProzessInformationBO prozessInformationBO = mock(ProzessInformationBO.class);
final VorgangBO vorgangBO = mock(VorgangBO.class);
when(vorgangsTracker.getProzessInformation()).thenReturn(prozessInformation);
when(dataproviderContext.getBOFactory()).thenReturn(boFactory);
when(boFactory.newBOInstance(EreignisprotokollBO.class)).thenReturn(mock(EreignisprotokollBO.class));
when(dataproviderContext.getBOs()).thenReturn(iDarkBOManager);
when(iDarkBOManager.getProzessInformationBO()).thenReturn(prozessInformationBO);
when(iDarkBOManager.getVorgangBO()).thenReturn(vorgangBO);
when(vorgangBO.loadVorgangsTrackerNonNullLazy()).thenReturn(vorgangsTracker);
when(dataproviderContext.getBOFactory().newBOInstance(AusfuehrungsEintragRef.class))
        .thenReturn(mock(AusfuehrungsEintragRef.class));
when(dataproviderContext.getPersistenceManager().getObjectById(eq(AusfuehrungsEintrag.class), any()))
        .thenReturn( mock(AusfuehrungsEintrag.class));
```

- dataproviderContext is our class (but in a different modul)
- dataproviderContext provides many functions
- has many nested objects
- I need just few operations, but on different nested objects
- I have to know many internals on how dataproviderContext is used
- how can I reduce/simplify the mocking stuff?

treat dataproviderContext as external and use ports/adapters to decouple

Solution

New interface (port) that defines only the operations I need

The implementation (adapter) just takes the original DataproviderContext as a constructor parameter and delegates all methods to it

The EreignisprotokollService gets the new Dataprovider interface injected (instead the old DataproviderContext)

```
dataprovider = mock(Dataprovider.class);
ereignisprotokollService = new EreignisprotokollServiceNewImpl(dataprovider);
```

Solution

```
class EreignisprotokollServiceNewShould {

    private Dataprovider dataprovider = mock(Dataprovider.class);
    private AuslenkungUebersteuertEreignis ereignis;
    private EreignisprotokollServiceNew ereignisprotokollService;

    @BeforeEach
    void setUp() {
        dataprovider = mock(Dataprovider.class);
        ereignisprotokollService = new EreignisprotokollServiceNewImpl(dataprovider);
        ereignis = null;
    }

    @Test
    void auslenkungUebersteuertEreignis_protokollieren() {
        ereignisTrittAuf();
        undWirdProtokolliert();
        dannWirdEsPersistiert();
    }

    private void ereignisTrittAuf() {
        List<UebersteuerterFehler> uebersteuerterFehler = Arrays.asList(
            new UebersteuerterFehler(new AusfuehrungseintragIdentity("fehler 1"), new RuleNumber("rule 1")),
            new UebersteuerterFehler(new AusfuehrungseintragIdentity("fehler 2"), new RuleNumber("rule 2"))
        );
        ereignis = AuslenkungUebersteuertEreignis.builder()
            .vorgangstrackerId(new VorgangstrackerId("vorgangstracker id"))
            .gui(StepKeyEnum.HK_NO_BEHANDLUNGSGRUND)
            .uebersteuerteFehler(uebersteuerterFehler)
            .build();
    }

    private void undWirdProtokolliert() {
        when(dataprovider.newEreignisprotokollInstance()).thenReturn(mock(Ereignisprotokoll.class));
        when(dataprovider.newAusfuehrungsEintragRefInstance()).thenReturn(mock(AusfuehrungsEintragRef.class));

        ereignisprotokollService.protokolliereAuslenkungUebersteuertEreignis(ereignis);
    }

    private void dannWirdEsPersistiert() {
        final ArgumentCaptor<Ereignisprotokoll> captureEreignisprotokoll = ArgumentCaptor.forClass(Ereignisprotokoll.class);
        verify(dataprovider).addEreignisprotokoll(captureEreignisprotokoll.capture());

        final Ereignisprotokoll ereignisprotokoll = captureEreignisprotokoll.getValue();
        verify(ereignisprotokoll).setEreignis(Ereignistyp.AUSLENKUNG_UEBERSTEUERT.name());
        verify(ereignisprotokoll).setEreigniszeitpunkt(any());
        verify(ereignisprotokoll).setGui(StepKeyEnum.HK_NO_BEHANDLUNGSGRUND.getName());
        verify(ereignisprotokoll).setAusloesenderUser("TO_shark2");
        ArgumentCaptor<AusfuehrungsEintragRef> capturedAusfuehrungsEintrafRef = ArgumentCaptor.forClass(AusfuehrungsEintragRef.class);
        verify(ereignisprotokoll, times(2)).addAusfuehrungsEintragRef(capturedAusfuehrungsEintrafRef.capture());
        List<AusfuehrungsEintragRef> erwarteteAusfuehrungsEintrafRef = capturedAusfuehrungsEintrafRef.getAllValues();
        assertThat(erwarteteAusfuehrungsEintrafRef, hasSize(2));
        verify(erwarteteAusfuehrungsEintrafRef.get(0)).setActingRuleNumber("rule 1");
        verify(erwarteteAusfuehrungsEintrafRef.get(1)).setActingRuleNumber("rule 2");
        verifyNoMoreInteractions(ereignisprotokoll);
    }
}
```

mocking hell reduced to these lines

Conclusion

- ports/adapters can also be used for decoupling internal services (even if it’s not a proper ports/adapters implementation, it was he
- has reduced the complexity in the current implementation
- testing was much easier
- allowed me to define convenience methods to access legacy code

Merçi for listening

and thank you for this great course

Contact:

Mail: juerg.weilenmann@css.ch

Twitter: -

Facebook: -

LinkedIn: -

Instagram: -

WhatsApp: -

Git: -

BeachBar: 18:00 - 20:00