Testgetriebene Entwicklung mit EasyMock

Sternback-Kaffee verkauft seit einiger Zeit Gutscheine, die mit Geld aufgeladen werden können und dann in den Filialen An Webterminals zur Bestellung von Getränken verwendet werden können.

Die erste User-Story lautet: Kaffe mit Gutschein vorbestellen. Als Tasks lassen sich ausmachen:

1. Daten für die Bestellung, Gutschein und Bestätigung aufnehmen
2. Die Geschäftslogik für die Verarbeitung und Speicherung von Bestellungen implementieren
3. Bestellungsverarbeitung mit Webseite verknüpfen.
4. Datenbank für Gutscheine, Getränke, Kunden und Bestätigung implementieren

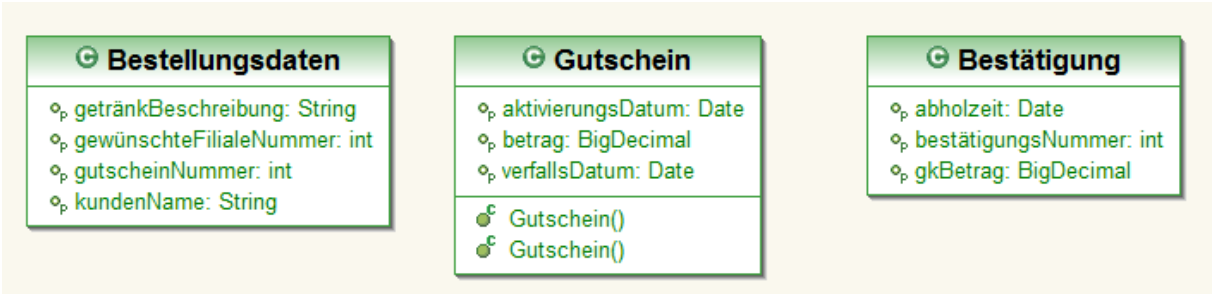
Der erste Task lässt sich durch folgende Aufgaben beschreiben:

- Die Bestellungsdaten darstellen:
  Kundenname, Getränkebeschreibung, Filialnummer, Gutscheinnummer

- Gutscheindaten darstellen:
  Aktivierungsdatum, Verfallsdatum, verbleibender Betrag

- Bestätigungsdaten:
  Bestätigungsnummer, Abholzeit, Restbetrag

Daraus ergeben sich z.B. die folgenden Klassen:



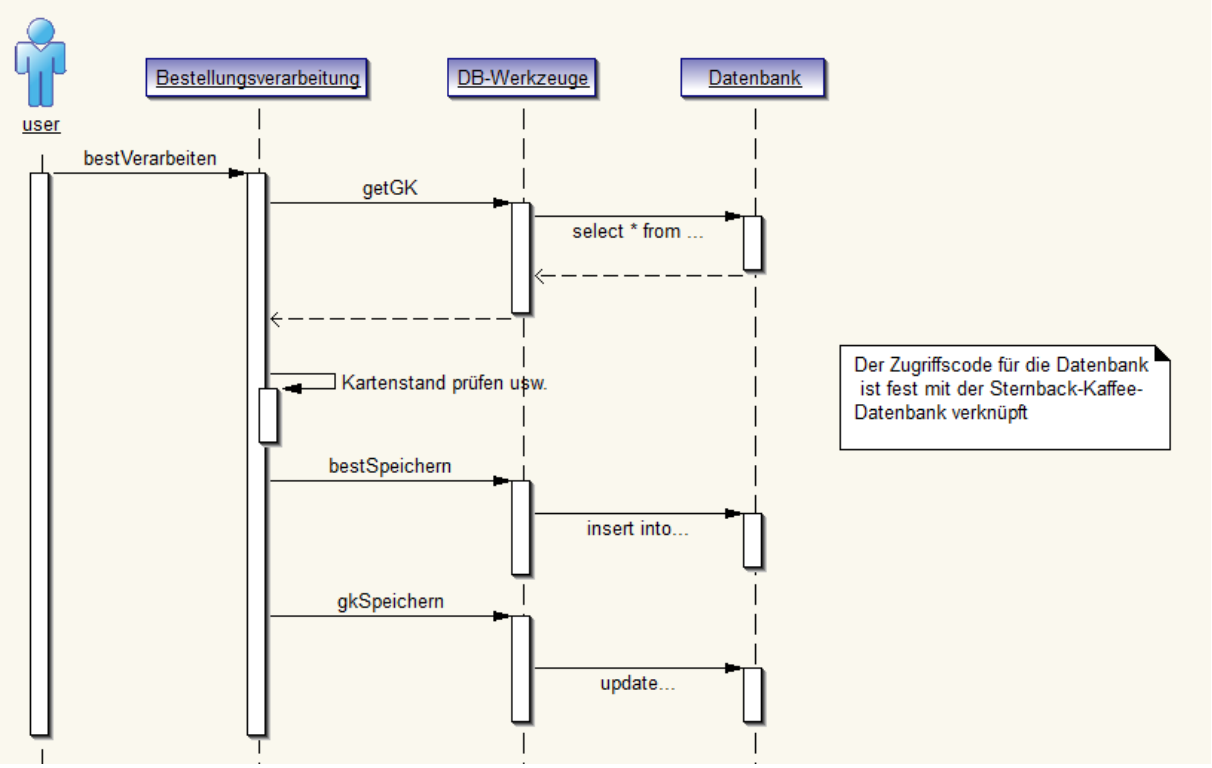
Unit-Tests für alleinstehende Klassen sind recht schnell erstellt. Benötigt eine Klasse jedoch Informationen aus einer oder mehreren anderen Klassen, so hat man mehrere Möglichkeiten:

- Stub
- Fake/Dummy
- Mock

Insbesondere, wenn Daten aus einer Datenbank benötigt werden, ist es in der Testphase sinnvoll, sich alternativer Möglichkeiten zu bedienen.

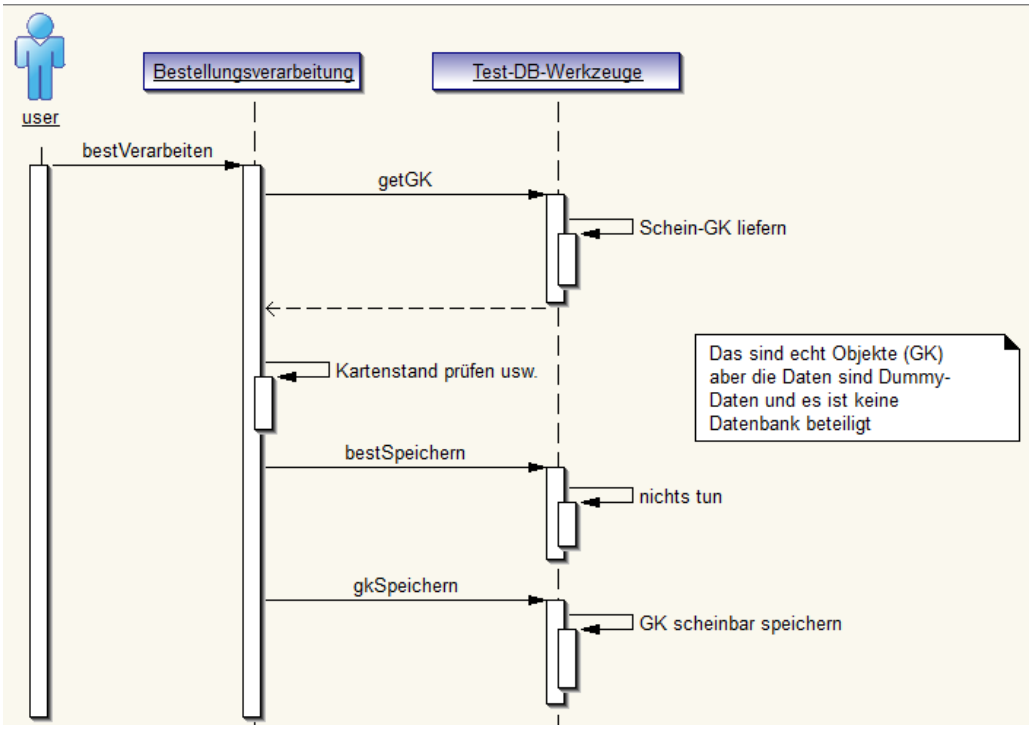
Bei der Verarbeitung einer Bestellung müssen die Gutscheindaten aus der Datenbank ausgelesen, geprüft und upgedatet werden.

Das haben wir…



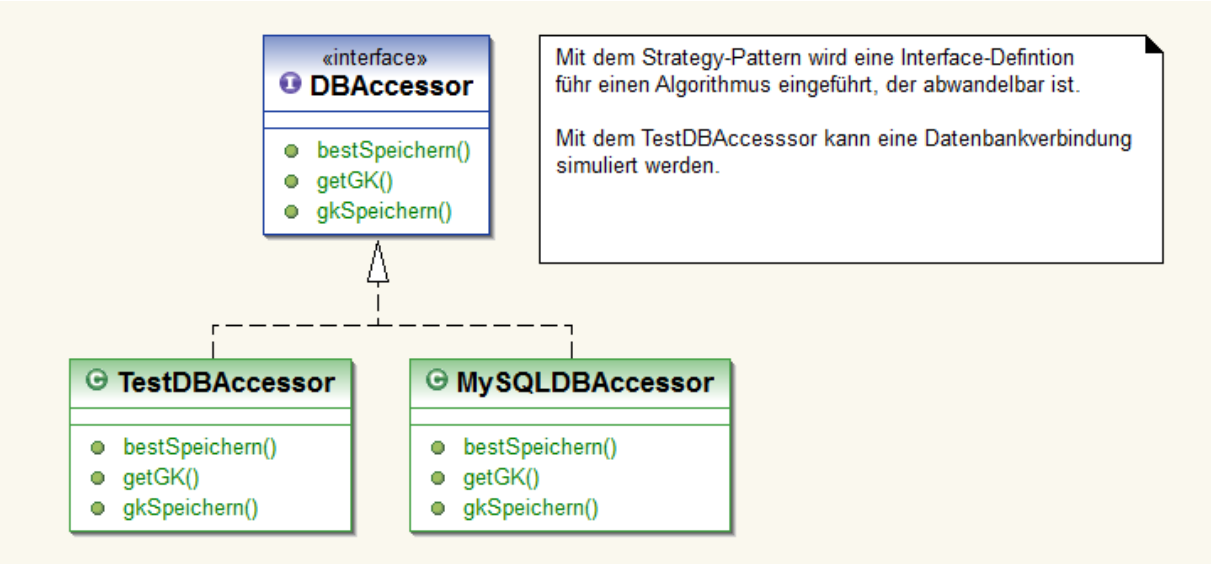
Komplizierter Testvorgang durch notwendige DB-Verbindung.

Und das brauchen wir:



Die Bestellungsverarbeitung soll ohne DB-Zugriff getestet werden können.

Umsetzung:



Der TestDBAccessor wird als Stub implementiert:

```
public class TestDBAccessor implements DBAccessor {
	public void bestSpeichern(Bestellungsdaten bestellung) {}
	public Gutschein getGK(int geld) { return new Gutschein(); }
	public void gkSpeichern(Gutschein card) {}
}
```

Der Test für die Verarbeitung einer einfachen Bestellung könnte demnach wie folgt lauten:

```
public class TestBestellungsverarbeitung_V1 {
	// V1 Testet mit Stub-Implementierung für DBAccessor

	// . . . weitere Tests

	@Test
	public void testEinfacheBestellung(){
		// Erst die Bestellungsverarbeitung erstellen
		Bestellungsverarbeitung_V1 verarbeitung = new
			Bestellungsverarbeitung_V1();
		verarbeitung.setDbAccessor(new TestDBAccessor());

		//Die aufzugebende Bestellung beschreiben
		Bestellungsdaten bestellung = new Bestellungsdaten();
		bestellung.setKundenName("Tim");
		bestellung.setGetränkBeschreibung(
			"Kaffe Latte mit Karamelsoße");
		bestellung.setGutscheinNummer(123456);
		bestellung.setGewünschteFilialeNummer(87654309);

		// Der Bestellungsverarbeitung die Bestellung geben und die
		// Bestätigung prüfen
		Bestätigung best = verarbeitung.bestVerarbeiten(bestellung);
		assertNotNull(best.getAbholzeit());
		assertTrue(best.getBestätigungsNummer()>0);
		assertTrue(best.getGkBetrag().equals(new BigDecimal(0)));
	}
}
```

Die dazu erforderliche Klasse Bestellungsverarbeitung könnte in der ersten Fassung so aussehen:

```java
public class Bestellungsverarbeitung_V1 {

	private DBAccessor dbAccessor;

	public Bestätigung bestVerarbeiten(Bestellungsdaten bestellung) {

		// Leeres Gutscheinobjekt wird vom Stub zurückgeliefert
		Gutschein gc =
			getDbAccessor().getGK(bestellung.getGutscheinNummer());

		// Das wird vom V1-Test erwartet
		gc.setBetrag(new BigDecimal(0));
		getDbAccessor().gkSpeichern(gc);

		// Nur für den Test erforderlich - hier fehlt noch was!
		Bestätigung best = new Bestätigung();
		best.setBestätigungsNummer(12345);
		best.setAbholzeit(new Date());
		best.setGkBetrag(gc.getBetrag());
		return best;
	}
	// weitere Methoden
```

V2 – Dummy-Implement.

Einen Schritt weiter als bei V1 geht man bei der DB-Simulation mit einem Fake. Hier wird das Zurückliefern eines echten Datensatzes vorgetäuscht:

```
public class TestBestellungsverarbeitung_V2 {

	// Dummy/Fake-Implementierung wird nur für Test benötigt
	public class TestAccessor implements DBAccessor {
		public Gutschein getGK(int geld) {
			Gutschein gc = new Gutschein();
			gc.setAktivierungsDatum(new Date());
			gc.setVerfallsDatum(new Date());
			gc.setBetrag(new BigDecimal(100));
			return gc;
		}
		public void bestSpeichern(Bestellungsdaten bestellung) {  }
		public void gkSpeichern(Gutschein card) {   }
	}
	@Test
	public void testEinfacheBestellung(){
		// Erst die Bestellungsverarbeitung erstellen
		Bestellungsverarbeitung verarbeitung = new
			Bestellungsverarbeitung();

		// Konfiguration auf interne Fakeimplementierung für den DB-
			Zugriff
		verarbeitung.setDBAccessor(new TestAccessor());

		//Die aufzugebende Bestellung beschreiben
		Bestellungsdaten bestellung = new Bestellungsdaten();
		bestellung.setKundenName("Tim");
		bestellung.setGutscheinNummer(12345);
		bestellung.setGetränkBeschreibung("Doppelter Espresso");
		bestellung.setGewünschteFilialeNummer(123);

		// Der Bestellungsverarbeitung die Bestellung geben und die
		// Bestätigung prüfen
		Bestätigung best = verarbeitung.bestVerarbeiten(bestellung);
		assertNotNull(best.getAbholzeit());
		assertTrue(best.getBestätigungsNummer()>0);
		assertTrue("sollte 100 sein" , best.getGkBetrag().equals(new
			BigDecimal(100)));
	}
}
		//Die aufzugebende Bestellung beschreiben
		Bestellungsdaten bestellung = new Bestellungsdaten();
		bestellung.setKundenName("Tim");
		bestellung.setGutscheinNummer(12345);
		bestellung.setGetränkBeschreibung("Doppelter Espresso");
		bestellung.setGewünschteFilialeNummer(123);

		// der Bestellungsverarbeitung die Bestellung geben und die
		// Bestätigung prüfen
		Bestätigung best = verarbeitung.bestVerarbeiten(bestellung);
		assertNotNull(best.getAbholzeit());
		assertTrue(best.getBestätigungsNummer()>0);
		assertTrue("sollte 100 sein" , best.getGkBetrag().equals(new
			BigDecimal(100)));
	}
}
```

Mock

Mit Hilfe einer Mocking-Technologie (EaysyMock, JMock, u.a.) können die zusätzlichen Klassen zur Simulation einer Datenbank vermieden werden, in dem einem Mock-Objekt das gewünschte Verhalten mitgeteilt wird.

```
public class TestBestellungsverarbeitungMock {

	@Test
	public void testEinfacheBestellung(){
		// Alles einrichten und uns bereit machen
		Bestellungsdaten bestellung = new Bestellungsdaten();
		bestellung.setKundenName("Tim");
		bestellung.setGutscheinNummer(12345);
		bestellung.setGetränkBeschreibung("Doppelter Espresso");
		bestellung.setGewünschteFilialeNummer(123);

		Date aktivierungsdatum = new Date();
		Date verfallsdatum = new Date(aktivierungsdatum.getTime()+3600);
		BigDecimal gkWert = new BigDecimal("2.75");

		// Wir brauchen einen Gutschein mit dem zu testenden Anfangswert
		Gutschein startGK = new Gutschein(aktvierungsdatum, verfallsdatum,
			gkWert);
		BigDecimal gkEndwert = new BigDecimal(0);

		// .. und einen mit den passenden Endwerten
		Gutschein endGK = new Gutschein(aktivierungsdatum, verfallsdatum,
			gkEndwert);

		// Hier wird das Mock-Objekt erstellt
		DBAccessor mockAccesssor = EasyMock.createMock(DBAccessor.class);

		//Dem Framework sagen, was aufgerufen wird und was erwartet wird
		EasyMock.expect(mockAccesssor.getGK(12345)).andReturn(startGK);
		// Erst sollte ein getGK()-Aufruf mit dem Wert 12345 erfolgen...
		// und dann das Objekt startGK() geliefert werden
		// >> Das simuliert den Kartenabruf auf der DB.

		// Simulation der Bestellungsverarbeitung
		mockAccesssor.bestSpeichern(bestellung);

		// Dann sollte auf dem Mock-Objekt gkSpeichern() mit einer leeren
			gc aufgerufen werden
		mockAccesssor.gkSpeichern(endGK);

		// und sonst sollte nichts aufgerufen werden. Beendet Aufzeichnung
		EasyMock.replay(mockAccesssor);

		// Eine Bestellungsverarbeitung erstellen ...
		Bestellungsverarbeitung processor = new Bestellungsverarbeitung();
		processor.setDBAccessor(mockAccesssor);
		Bestätigung best = processor.bestVerarbeiten(bestellung);

		// Die Bestätigung prüfen...
		assertNotNull("Abholzeit sollte > 0 sein", best.getAbholzeit());
		assertTrue("Bestätigungsnummer >0",
			best.getBestätigungsNummer()>0);
		assertTrue("Restbetrag = 2.75", best.getGkBetrag().equals(new
				BigDecimal(2.75)));
	}
}
```

Damit Mocking funktioniert, müssen weitere Dateien in den ClassPath mit aufgenommen werden, und zwar cglib und asm. Sinnvoll ist die Ablage im lib-Verzeichnis und das Einbinden in den BuildPath.