Workshop: PowerMock
Übungen

Ablauf

Dieser Workshop gliedert sich in mehrere Vortragsteile, die den Teilnehmern die Thematik PowerMock näherbringen. Im Anschluss daran sind jeweils einige Übungsaufgaben von den Teilnehmern – idealerweise in Gruppenarbeit – am Rechner zu lösen.

Voraussetzungen

1) Aktuelles JDK 11, idealerweise auch JDK 14/15, installiert
2) Aktuelles Eclipse installiert (Alternativ: NetBeans oder IntelliJ IDEA)

Teilnehmer

- Entwickler und Architekten mit Java-Erfahrung, die ihre Kenntnisse zu PowerMock vertiefen möchten

Kursleitung und Kontakt

Michael Inden
Derzeit freiberuflicher Buchautor und Trainer
E-Mail: michael.inden@hotmail.com
Blog: https://jaxenter.de/author/minden

Aufgabe 1: SystemProperty 15 min

Die Klasse `Ex01_SystemProperties` bietet Zugriff auf das von Java bereitgestellte Interface für System-Properties. Für die Klasse `Ex01_SystemProperties` existiert ein Unit Test namens `Ex01_SystemPropertiesTest`. Ermögliche dort den Zugriff auf ein spezifisches System-Property. Worin liegt die Schwierigkeit? Welche Alternative zu PowerMockito haben wir hier? Was verändert sich, wenn die Klasse `Ex01_SystemProperties` `final` definiert wäre?

```
public class Ex01_SystemProperties
{
    public String getSystemProperty(String propertyName)
    {
        return System.getProperty(propertyName);
    }
}
```

Aufgabe 2: PointFactory 15 min

Bei dieser Aufgabe wird in einer Klasse `Ex02_PointFactory` ein Point-Objekt per `new` erzeugt. Zum Testen wollen wir jedoch immer auf genau dasselbe Objekt zugreifen. Die Schwierigkeit liegt also darin, im Unit Test die Instanziierung zu kontrollieren und mit einem Mock zu ersetzen. Was macht hier Schwierigkeiten beim Vergleich? Wie kann man das beheben?

```
public class Ex02_PointFactory
{
    public Point create()
    {
        // complex logic
        return new Point(11, 11);
    }
}
```

Aufgabe 3: FigureFactory 30 min

In dieser Aufgabenstellung ist eine Klasse `Ex03_FigureFactory` zum Erzeugen von grafischen Figuren (hier durch reine Strings abstrahiert) basierend auf einem Enum gegeben. Der korrespondierende Unit Test soll zum Laufen gebracht werden. Im ersten Fall sind die passenden Anweisungen einzufügen. Im zweiten Testfall geht es darum, den bislang nicht implementierten Typ HEXAGON trotzdem in Tests nutzen zu können.

```
public class Ex03_FigureFactory
{
    public enum FigureTypes
    {
        RECT, LINE, OVAL, CIRCLE, HEXAGON
    }
```

```
    public static String getFigure(FigureTypes type)
    {
        if (type == FigureTypes.RECT)
            return "RECT";
        if (type == FigureTypes.LINE)
            return "LINE";

        throw new IllegalStateException("Unsupported Type: " +
type);
    }
}
```

Aufgabe 4: Modifikation der Rückgabe einer finalen Methode 10 min

Nehmen wir an, wir wollten ein System testen, dessen Zustand durch folgende Klasse `Ex04_SystemStateService` ermittelt und bereitgestellt wird. Leider enthält die Klasse noch einen Fehler und wir können die anderen Systembausteine nicht sauber testen, da als Systemzustand immer `false` geliefert wird. Für die Tests soll `isAlive()` den Wert `true` zurückgeben.

```
public class Ex04_SystemStateService
{
    public final boolean isAlive()
    {
        return false;
    }
}
```

Aufgabe 5: Sub- und Superklasse ohne Methodenaufruf 10 min

Gegeben sind die Klassen `Ex05_SuperClass` und `Ex05_SubClass` sowie ein zugehöriger Unit Test. Da die Methode `someMethod()` durch die Subklasse überschrieben ist und in der Basisklasse eine Exception auslöst, gibt es Schwierigkeiten beim Testen. Bringen Sie das Ganze mithilfe von PowerMockito zum Laufen.

```
public class Ex05_SuperClass
{
    public String someMethod()
    {
        throw new IllegalArgumentException();
    }
}
```

Aufgabe 6: Würfel 30 min

Gegeben ist folgende Klasse Ex06_Dice, die einen Würfel simuliert und die Augensumme verschiedener Wiederholungen von Würfen berechnen kann. Analysiere diese Klasse und korrigiere danach den zugehörigen Unit Test, sodass alle Testfälle bestanden werden.

```
public class Ex06_Dice
{
    public int roll10()
    {
        int sum = 0;
        for (int i = 0; i < 10; i++)
        {
            int roll = new Random().nextInt();
            sum += roll;
        }
        return sum;
    }

    public int sumOfNumDiceRolls(int num)
    {
        int sum = 0;
        for (int i = 0; i < num; i++)
        {
            int roll = (int) (6 * Math.random()) + 1;
            sum += roll;
        }
        return sum;
    }

}
```

Aufgabe 7: 45 min

Nachfolgende Klasse stellt über die Methoden `randomMessageId()` und `getUrl()` Daten bereit. Für die erste Methode soll der Fehlerfall getestet werden, also der `catch`-Zweig. Für die zweite Methode soll statt der unsinnigen URL immer eine mit Localhost geliefert werden. Ergänzen Sie den zugehörigen Unit Test so, dass die Testfälle bestanden werden. Zu guter Letzt soll auch die Ermittlung eines Zeitstempels immer einen definierten Wert liefern.

```
public class Ex07_MessageIdAndUrlUtil
{
    public static String randomMessageId()
    {
        try
        {
            return
InetAddress.getLocalHost().getHostAddress();
        }
        catch (UnknownHostException e)
        {
            return generateUUID();
        }
    }

    public static URL getUrl() throws MalformedURLException
    {
        return new URL("SILLY_INVALID");
    }

    private static final String generateUUID()
    {
        return UUID.randomUUID().toString();
    }

    public static final String currentTime()
    {
        return LocalTime.now().toString();
    }
}
```

Aufgabe 8: Design-Diskussion 30 + 15 = 45 min

Im Internet findet man unter https://www.javaindeed.com/3-best-practices-to-test-a-code-that-calls-static-methods/ eine interessante Besprechung mit dem Titel «***3 Best Practices to Test a Code That Calls Static Methods***». Lies den Text und bereite dich auf die Diskussion im Team vor.