

Testowanie i wdrażanie systemów informatycznych

Laboratorium: biblioteka EasyMock

Zadania zostały zaczerpnięte z wykładu „Programowanie zespołowe” Pana dr hab. Jana G. Bazana , prof. UR

1. Proszę stworzyć klasę DataSet przechowującą wartości liczbowe:

```
import java.util.ArrayList;

public class DataSet {
    ArrayList<Integer> values;
    int currentIndex;

    public DataSet(){
        values = new ArrayList<>();
        currentIndex = 0;
    }

    public void resetIndex(){
        currentIndex = 0;
    }

    public void addValue(int val){
        values.add(val);
    }

    public boolean isNextValue(){
        if (currentIndex<values.size()) return true;
        else return false;
    }

    public int nextValue(){
        if (currentIndex>=values.size()) throw new IllegalStateException("No more values");
        int nv = values.get(currentIndex).intValue();
        currentIndex++;
        return nv;
    }
}
```

2. Następnie tworzymy klasę udostępniającą usługi statystyczne jak np. liczenie średniej arytmetycznej na danych zawartych w obiektach klasy DataSet:

```
public class Statystyki {
    public Statystyki(){}

    public double getMean(DataSet dataSet){
        dataSet.resetIndex();
        double sum = 0.0;
        int count =0;
        while (dataSet.isNextValue()){
            int nVal = dataSet.nextValue();
            sum = sum + nVal;
            count++;
        }

        if(count==0){
            throw new IllegalStateException("No values!");
        }

        return sum/count;
    }
}
```

3. Do stworzonej klasy Statystyki piszemy test jednostkowy sprawdzający poprawność liczenia średniej arytmetycznej jak poniżej:

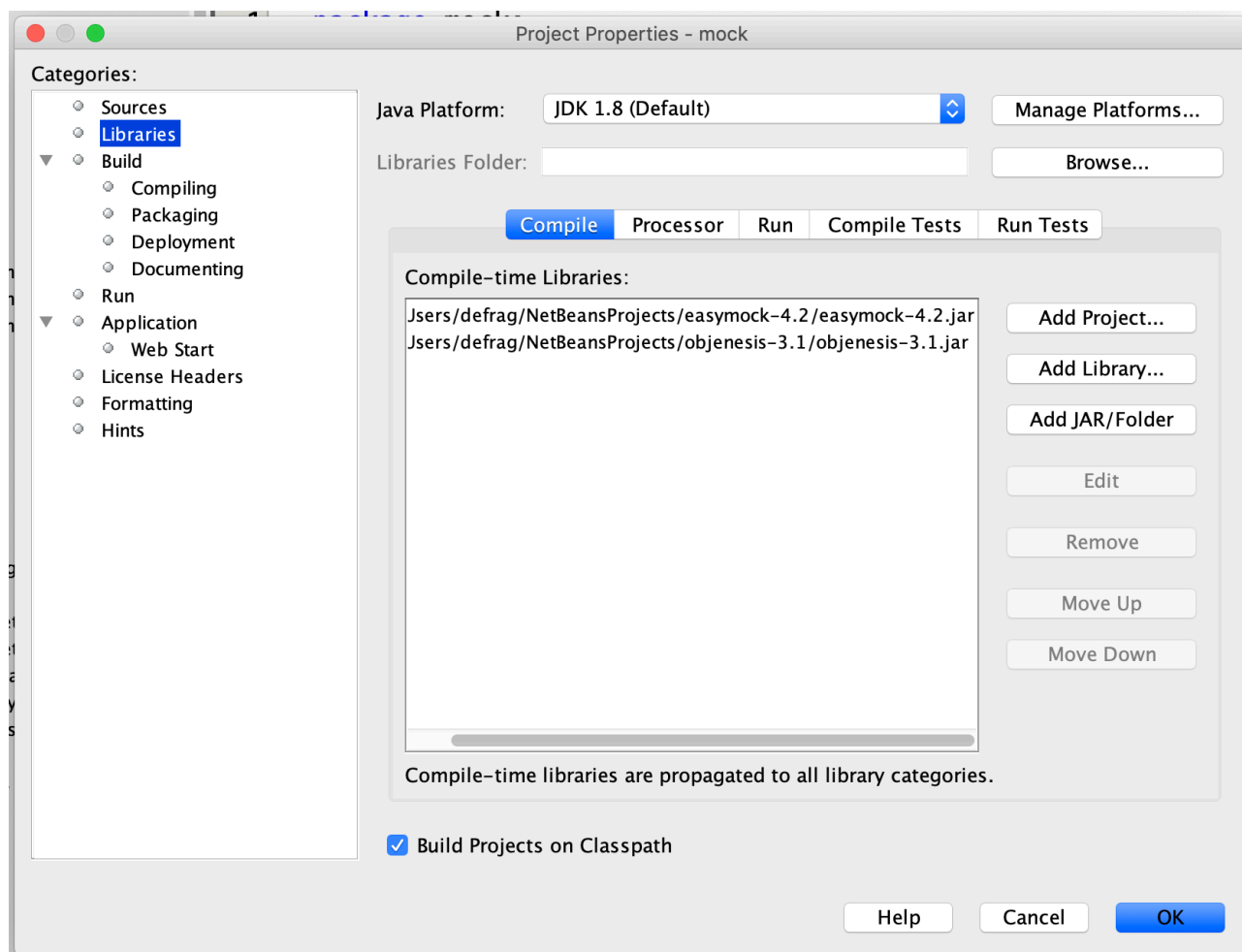
```
public class StatystykiTest {

    public StatystykiTest() {
    }

    @Test
    public void testGetMean() {
        DataSet dataSet = new DataSet();
        dataSet.addValue(3);
        dataSet.addValue(4);
        dataSet.addValue(2);
        Statystyki instance = new Statystyki();
        double expectedResult = 3.0;
        double result = instance.getMean(dataSet);
        assertEquals(expectedResult, result, 0.01);
    }
}
```

4. Z założenia testy jednostkowe powinny być niezależne od siebie i od innych części systemu jak np. bazy danych. Taka zależność powodowałaby trudności w uruchomieniu testów kiedy nie dysponuje się pozostałymi komponentami systemu. Może się także zdarzyć, że testowany kod korzysta z klas, które nie zostały jeszcze zaimplementowane i zostały na razie zdefiniowane tylko w formie interfejsów. Co zrobić jeżeli nie dysponujemy klasą DataSet ? Można użyć mechanizmu zwanego z

języka angielskiego mock - po polsku można by to nazwać słowem „imitacja” lub „zaślepka”. Pomysł polega na stworzeniu obiektu, który w wymaganych sytuacjach będzie zachowywał się jak oryginał pomimo, że nie będzie oferował pełnej jego funkcjonalności. W ćwiczeniu wykorzystamy bibliotekę EasyMock. Aby z niej skorzystać należy pobrać ją z poniższego adresu: <https://easymock.org/> oraz dodatkowo wymaganą bibliotekę Objenesis ze strony <http://objenesis.org/download.html> . Pobrane pliki zip należy rozpakować po czym dodać do projektu w jego właściwościach odpowiednie pliki jar:



5. Do klasy z testami należy dodać odpowiednie importy:

```
import static org.easymock.EasyMock.createMock;  
import static org.easymock.EasyMock.replay;  
import static org.easymock.EasyMock.verify;  
import static org.easymock.EasyMock.expectLastCall;
```

6. Dalej tworzymy nowy test wykorzystujący mechanizm mockowania:

```
@Test
public void testGetMeanMock() {
    DataSet dataSetMock = createMock(DataSet.class);
    dataSetMock.resetIndex(); expectLastCall();
    dataSetMock.isNextValue(); expectLastCall().andReturn(true);
    dataSetMock.nextValue(); expectLastCall().andReturn(2);
    dataSetMock.isNextValue(); expectLastCall().andReturn(true);
    dataSetMock.nextValue(); expectLastCall().andReturn(3);
    dataSetMock.isNextValue(); expectLastCall().andReturn(true);
    dataSetMock.nextValue(); expectLastCall().andReturn(4);
    dataSetMock.isNextValue(); expectLastCall().andReturn(false);
    replay(dataSetMock);

    Statystyki instance = new Statystyki();
    double expResult = 3.0;
    double result = instance.getMean(dataSetMock);
    verify(dataSetMock);
    assertEquals(expResult, result, 0.01);
}
```

Jak widać najpierw za pomocą specjalnej metody *createMock()* tworzymy obiekt imitujący obiekt podanej klasy. Na tym etapie mamy obiekt nieznanej klasy, który jednak możemy przypisać do zmiennej typu klasy wzorcowej (czyli albo dziedziczy po klasie wzorcowej, lub jeżeli podaliśmy interfejs to go implementuje).

Dalej nowo powstały obiekt uczymy spodziewanych wywołań jego metod oraz reakcji (zwracanych wartości). Proszę zwrócić uwagę, że uczone wywołania mają identyczną sekwencję jak te w metodzie *getMean()* klasy *Statystyki*, gdy klasa *DataSet* zawiera trzy elementy *'2','3','4'*.

Nauczoną „imitację” obiektu klasy *DataSet* przełączamy w trym odtwarzania poprzez wywołanie metody *replay()*. Od tej chwili jeżeli sekwencja wywołań jej metod będzie identyczna z założoną będą one zwracać założone wartości. W przeciwnym wypadku zgłoszony zostanie błąd.

Dalsza część testu wygląda analogicznie do poprzedniego testu. Dodano jedynie wywołanie metody *verify()*, która sprawdza czy cała założona sekwencja została odtworzona prawidłowo.

7. Proszę uruchomić testy i sprawdzić czy mechanizm działa. Oba testy powinny zostać zaliczone.

8. W powyższym przykładzie tworzymy obiekt mocka na podstawie istniejącej klasy. Ktoś mógłby powiedzieć, że w praktyce ma to średni, bo sens skoro i tak musimy mieć już napisaną odpowiednią klasę na wzór to przecież można użyć właśnie jej. Sytuacja nie byłaby już taka prosta gdyby nasza klasa *DataSet* pobierała informacje z bazy danych - wtedy takie użycie miałoby większy sens - klasa by istniała, ale baza już nie musi wcale istnieć. Spróbujmy jednak założyć, że nie posiadamy aktualnie implementacji klasy *DataSet*, a jedynie dysponujemy „jej planem” - wiemy jakie będzie posiadać metody i jakie będą ich zadania. Takim planem jest właśnie interfejs - zawiera on bowiem definicje metod, bez ich implementacji.

9. Zmienimy nazwę klasy `DataSet` na `DataSetOrg` zmieniając jak poniżej (nie korzystamy z opcji `Refactor/Rename` gdyż zmieniłaby nam ona także odwołania do tej klasy w pozostałym kodzie):

```
public class DataSetOrg {
    ArrayList<Integer> values;
    int currentIndex;

    public DataSetOrg(){
        values = new ArrayList<>();
        currentIndex = 0;
    }

    public void resetIndex(){
        currentIndex = 0;
    }

    public void addValue(int val){
        values.add(val);
    }

    public boolean isNextValue(){
        if (currentIndex < values.size()) return true;
        else return false;
    }
}
```

Klikamy w żarówkę i wybieramy „Rename file to `DataSetOrg`”. Tym sposobem nie ma już implementacji klasy `DataSet`.

10. Dalej tworzymy odpowiedni interfejs definiujący jakie funkcje powinna oferować klasa `DataSet`:

```
public interface DataSet {
    public void resetIndex();
    public void addValue(int val);
    public boolean isNextValue();
    public int nextValue();
}
```

11. Jak Państwo zauważycie po odświeżeniu projektu pierwszy z testów będzie zawierał błąd:

```
@Test
public void testGetMean() {
    DataSet dataSet = new DataSet();
    dataSet.addValue(3);
    dataSet.addValue(4);
    dataSet.addValue(2);
    Statystyki instance = new Statystyki();
    double expResult = 3.0;
    double result = instance.getMean(dataSet);
    assertEquals(expResult, result, 0.01);
}
```

Brak jest implementacji klasy `DataSet`, więc jest to prawidłowe, że nie możemy stworzyć nowego obiektu tej klasy. Widzimy, że w tej sytuacji klasyczny sposób pisania testu jednostkowego nie jest możliwy do wykorzystania. Test ten musimy zatem wykomentować.

12. Proszę sprawdzić czy drugi test oparty na mechanizmie mock zadziała. Aby test przeszedł prawidłowo potrzebne jest jeszcze wyczyszczenie projektu poprzez opcję „*Run/Clean and Build Project*”. Inaczej częściowo skompilowany kod będzie sugerował bibliotece EasyMock, że ma emulować klasę zamiast interfejsu.
13. Jeżeli wszystko poszło prawidłowo jasno widać, że wcale nie jest nam potrzebna sama klasa `DataSet`, aby testować metodę `getMean()` klasy `Statystyki` pomimo, że do jej działania teoretycznie powinna być wymagana.