Тестирование и объектно-ориентированное программирование

Кулаков Кирилл Александрович

ООП

- Программа = совокупность взаимодействующих объектов
- Принципы ООП:
  - Абстракция (выделение важного)
  - Инкапсуляция (сокрытие/объединение)
  - Наследование (родитель-потомок)
  - Полиморфизм (множество реализаций одного интерфейса)

Элементы ООП

- **Класс** — совокупность свойств и методов
  - Public — доступ везде
  - Protected — доступ внутри класса и в наследниках
  - Private — доступ только внутри класса
- **Абстрактный класс** — класс с абстрактным методом
- **Интерфейс** — «класс без свойств и реализаций методов»
- Принадлежность элементов к классу (static) или к объекту (non-static)

- Объект тестирования: экземпляр класса
  - Интерфейс
  - Реализации методов

Тестирование public методов

- Статические методы
  - Аналогично функциям + специфика ООП
- Не статические методы
  - Создание экземпляра класса
  - Установка свойств класса
  - Вызов метода у экземпляра
  - Специфика ООП

Тестирование protected методов

- Тестирование через класс-наследник
- Статические методы
  - Вызов через статический метод класса наследника
- Нестатические методы
  - Создание экземпляра класса-наследника
  - Установка свойств класса через класс-наследник
  - Вызов метода у экземпляра через класс-наследник
  - Специфика ООП

Тестирование private методов

- Нет доступа ни напрямую ни через наследники
- Косвенное тестирование (через public/protected методы)
- Использование специфики ООП конкретного языка (дружественные классы, рефлексии, ...)

Пример тестируемого класса

```
class A {
    public:
        int pub_method(int bb) {
            return bb > 5 ? bb + bb : this->priv_method(bb) * bb;
        }
    protected:
        int prot_method(int a, int b) {
            return a > 0 ? this->priv_method(b + 2) - b: b;
        }
    private:
        int priv_method (int aa) {
            return a < 3 ? aa : aa * aa;
        }
}
```

Специфика ООП

- Тестируемый метод может использовать:
  - Методы и свойства класса
  - Экземпляры классов в качестве параметров
  - Экземпляры классов внутри кода (создание/получение)
  - Внешние сущности: глобальные объекты, статические методы других классов, файлы, потоки данных, …
- Должна существовать возможность подмены зависимостей!!!

Подготовка кода к тестированию

- Разделение описания класса и реализаций методов
  - Каждый класс в отдельном файле
  - (в идеале) каждый метод в отдельном файле
- Использование интерфейсов/абстрактных классов вместо реализаций в параметрах методов
- Минимизация/исключение использование статических методов
- «Правильное» разделение функционала: минимизация взаимодействия между классами

Дружественные классы и методы

- Способ обхода парадигмы ООП (инкапсуляции)
- В описание тестируемого класса добавляется информация о дружественных классах/методах
- Пример:

```
class A {
    friend class B;
    private:
        int method_x(int a) {…}
        friend int call_method_x(A &, int b);
}
class B{
    public:
        int another_call_method_x(A &a, int b) { return a.method_x(b);}
}
```

Рефлексии

- У некоторых языков программирования нет дружественных классов (пример: PHP)
- Нет возможности указать класс дружественным
- Использование рефлексии: вызов метода по его сигнатуре
- Пример:

```
$class = new ReflectionClass('GraphController');
$method = $class->getMethod('compare');
$method->setAccessible(true);
$ret = $method->invokeArgs($smalt, array($graph1, 2));
```

Заглушки

- Заглушка — управляемая замена существующей зависимости (или компаньона) в системе
- Использование:
  - Определение интерфейса вызова (прототипа)
    - модификация кода для выделения интерфейса
  - Определение необходимого результата
    - возвращаемое значение
    - воздействие на систему
  - Реализация заглушки
  - Запуск теста для объекта с заглушкой

Как запустить заглушку?

- Необходима возможность подмены куска кода (зазор)
- Реализация взаимодействия через интерфейс
  - получить интерфейс в конструкторе и сохранить его в поле для последующего использования;
  - получить интерфейс в свойстве и сохранить его в поле для последующего использования;
  - получить интерфейс непосредственно перед вызовом в тестируемом методе одним из следующих способов:
    - в виде параметра метода (внедрение через параметр);
    - с помощью фабричного класса;
    - с помощью локального фабричного метода;
    - варианты вышеупомянутых способов.

Пример (работа с БД)

- Обычный код

```
public Component(String databaseURL) {
    try {
        databaseConnection = DriverManager.getConnection(databaseURL);
        ...
    } catch (SQLException e) {...}
}
public String get(String key) {
    try {
        Statement stmt = databaseConnection.createStatement();
        ResultSet rs = stmt.executeQuery(
            "SELECT value FROM Table1 WHERE key=" + key);
        ...
    } catch (SQLException e) {...}
}
```

Пример (работа с БД)

- Реализация интерфейса

```
interface KeyValuePairs {
    String get(String key);
    void put(String key, String value);
}
```

- Рефакторинг кода

```
class DatabaseAdapter implements KeyValuePairs {
    ...
}
```

Пример (работа с БД)

- Реализация заглушки

```
class FakeDatabase implements KeyValuePairs {
    Hashtable table = new Hashtable();
    public String get(String key) {
        return (String) table.get(key);
    }
    public void put(String key, String value) {
        table.put(key, value);
    }
}
```

Mock-объект

- Mock-объект (подделка, подставка) — тип объектов, реализующих заданные аспекты моделируемого программного окружения.
- Mock-объект представляет собой конкретную фиктивную реализацию интерфейса, предназначенную исключительно для тестирования взаимодействия и относительно которого высказывается утверждение.
- Отличие mock-объекта от заглушки — проверка состояния mock-объекта и его использования (взаимодействия) в ходе теста

Базовые функции mock-объектов

- Ведение лога использования объекта
- Реализация заглушек для методов
- Проверка наличия вызовов методов
- Переопределение методов
- "Слежение" за экземплярами классов

Изолирующие каркасы

- Программные средства для создания заглушек/подделок
- Автоматическая генерация заглушек на основе интерфейса во время выполнения теста
- Использование правил для отслеживания изменений
- Указание возвращаемых значений
- Указание значений свойств

Пример (Java, Mockito)

```
import static org.mockito.Mockito.*;

//вот он - mock-объект (List.class - это интерфейс)
List mockedList = mock(List.class);

//используем его
mockedList.add("one");
mockedList.clear();

//проверяем, были ли вызваны методы add с параметром "one" и clear
verify(mockedList).add("one");
verify(mockedList).clear();
```

Пример (Java, Mockito)

```
LinkedList mockedList = mock(LinkedList.class);

//stub'инг
when(mockedList.get(0)).thenReturn("first");
when(mockedList.get(1)).thenThrow(new RuntimeException());

//получим "first"
System.out.println(mockedList.get(0));

//получим RuntimeException
System.out.println(mockedList.get(1));

//получим "null" ибо get(999) не был определен
System.out.println(mockedList.get(999));
```

Пример (Google Mock)

```
#include "gmock/gmock.h"  // Brings in gMock.

class MockTurtle : public Turtle {
 public:
  ...
  MOCK_METHOD(void, PenUp, (), (override));
  MOCK_METHOD(void, PenDown, (), (override));
  MOCK_METHOD(void, Forward, (int distance), (override));
  MOCK_METHOD(void, Turn, (int degrees), (override));
  MOCK_METHOD(void, GoTo, (int x, int y), (override));
  MOCK_METHOD(int, GetX, (), (const, override));
  MOCK_METHOD(int, GetY, (), (const, override));
};
```

Пример (Google Mock)

```
#include "path/to/mock-turtle.h"
#include "gmock/gmock.h"
#include "gtest/gtest.h"

using ::testing::AtLeast;                         // #1

TEST(PainterTest, CanDrawSomething) {
  MockTurtle turtle;                              // #2
  EXPECT_CALL(turtle, PenDown())                  // #3
      .Times(AtLeast(1));

  Painter painter(&turtle);                       // #4

  EXPECT_TRUE(painter.DrawCircle(0, 0, 10));      // #5
}
```

Тестирование и паттерны ООП

- Подмена зависимостей (в т.ч. для структурных паттернов)
- Наследование объектов (в т.ч. для порождающих паттернов)
- Отдельная реализация методов (в т.ч. для паттернов поведения)
- Пример:

```
class Singletone {
    public:
        static Singletone * getInstance() {… return instance;}
    private:
        static Singletone *instance;
}
Singletone *Singletone::instance = nullptr;
```

Тестирование и паттерны ООП

- Модификация кода:

```
class Singletone {
    public:
        static Singletone * getInstance() {… return instance;}
    protected:
        Singletone *instance;
}
```

- Написание обертки:

```
class MockSingletone : public Singletone {
    public:
        void init() {instance = new MockSingletone();}
        void cleanup() {delete instance; instance = nullptr;}
        ...
}
```