Лабораторная работа №4

### Интеграционное тестирование

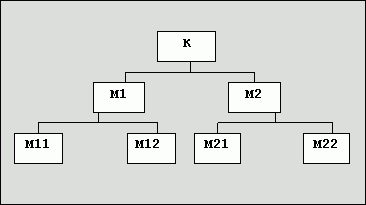
*Цель работы:* научиться выполнять интеграционное тестирование ПО.

*Отчет по лабораторной работе:* тестируемая программа на языке Java, набор интеграционных и модульных тестов к ней на основе JUnit 4.

## Теоретическая часть

**Интеграционное тестирование** - это тестирование части системы, состоящей из двух и более модулей. Основная задача интеграционного тестирования - поиск дефектов, связанных с ошибками в реализации и интерпретации интерфейсного взаимодействия между модулями. Иначе говоря, оно предназначено для проверки связи между компонентами, а также взаимодействия с различными частями системы (операционной системой, оборудованием либо связи между различными системами).

С технологической точки зрения интеграционное тестирование является количественным развитием модульного, поскольку так же, как и модульное тестирование, оперирует интерфейсами модулей и подсистем и требует создания тестового окружения, включая заглушки (stub) на месте отсутствующих модулей. Основная разница между модульным и интеграционным тестированием состоит в целях, то есть в типах обнаруживаемых дефектов, которые, в свою очередь, определяют стратегию выбора входных данных и методов анализа. В частности, на уровне интеграционного тестирования часто применяются методы, связанные с покрытием интерфейсов, например, вызовов функций или методов, или анализ использования интерфейсных объектов, таких как глобальные ресурсы, средства коммуникаций, предоставляемых операционной системой.



На рисунке приведена структура комплекса программ K, состоящего из оттестированных на этапе модульного тестирования модулей M1, M2, M11, M12, M21, M22. Задача, решаемая методом интеграционного тестирования, - тестирование межмодульных связей, реализующихся при исполнении программного обеспечения комплекса K. Интеграционное тестирование использует модель "белого ящика" на модульном уровне. Поскольку тестировщику текст программы известен с детальностью до вызова всех модулей, входящих в тестируемый комплекс, применение структурных критериев на данном этапе возможно и оправдано.

Интеграционное тестирование применяется на этапе сборки модульно оттестированных модулей в единый комплекс. Известны два **метода сборки модулей**:

1. **Монолитный**, характеризующийся одновременным объединением всех модулей в тестируемый комплекс.
2. **Инкрементальный**, характеризующийся пошаговым (помодульным) наращиванием комплекса программ с пошаговым тестированием собираемого комплекса. В инкрементальном методе выделяют две стратегии добавления модулей:
   1. "Сверху вниз" и соответствующее ему нисходящее тестирование.
   2. "Снизу вверх" и соответственно восходящее тестирование.

**Особенности монолитного тестирования** заключаются в следующем: для замены неразработанных к моменту тестирования модулей, кроме самого верхнего (К на рисунке), необходимо дополнительно разрабатывать драйверы (test driver) и/или заглушки (stub), замещающие отсутствующие на момент сеанса тестирования модули нижних уровней.

Сравнение монолитного и инкрементального подхода дает следующее:

* Монолитное тестирование требует больших трудозатрат, связанных с дополнительной разработкой драйверов и заглушек и со сложностью идентификации ошибок, проявляющихся в пространстве собранного кода.
* Пошаговое тестирование связано с меньшей трудоемкостью идентификации ошибок за счет постепенного наращивания объема тестируемого кода и соответственно локализации добавленной области тестируемого кода.
* Монолитное тестирование предоставляет большие возможности распараллеливания работ особенно на начальной фазе тестирования.

**Особенности нисходящего тестирования** заключаются в следующем: организация среды для исполняемой очередности вызовов оттестированными модулями тестируемых модулей, постоянная разработка и использование заглушек, организация приоритетного тестирования модулей, содержащих операции обмена с окружением, или модулей, критичных для тестируемого алгоритма.

Например, порядок тестирования комплекса K при нисходящем тестировании может быть таким, как показано в примере, где тестовый набор, разработанный для модуля Mi, обозначен как XYi = (X, Y)i

1) K->XYK

2) M1->XY1

3) M11->XY11

4) M2->XY2

5) M22->XY22

6) M21->XY21

7) M12->XY12

Недостатки нисходящего тестирования:

* Проблема разработки достаточно "интеллектуальных" заглушек, т.е. заглушек, пригодных к использованию при моделировании различных режимов работы комплекса, необходимых для тестирования
* Сложность организации и разработки среды для реализации исполнения модулей в нужной последовательности
* Параллельная разработка модулей верхних и нижних уровней приводит к не всегда эффективной реализации модулей из-за подстройки (специализации) еще не тестированных модулей нижних уровней к уже оттестированным модулям верхних уровней

**Особенности восходящего тестирования** в организации порядка сборки и перехода к тестированию модулей, соответствующему порядку их реализации.

Например, порядок тестирования комплекса K при восходящем тестировании может быть следующим.

1) M11->XY11

2) M12->XY12

3) M1->XY1

4) M21->XY21

5) M2(M21, Stub(M22))->XY2

6) K(M1, M2(M21, Stub(M22)) ->XYK

7) M22->XY22

8) M2->XY2

9) K->XYK

Недостатки восходящего тестирования:

* Запаздывание проверки концептуальных особенностей тестируемого комплекса
* Необходимость в разработке и использовании драйверов

## Задание

Сформировать иерархию классов вокруг класса, протестированного в лабораторной работе №3. К примеру, для класса «Животное» это могут быть наследники «Кошка», «Собака» и агрегирующие классы «Зоопарк» и «Лес». Провести модульное тестирование написанных классов. Провести инкрементальное интеграционное тестирование «сверху вниз».