Основы объектно-ориентированного проектирования в Java

Анализ и проектирование (Analysis and Design)

Анализ описывает то, **что** система должна делать:

- Моделирование реального мира, включающее действующих субъектов (actors), виды деятельности (activities), объекты (objects) и поведение (behaviors).

Проектирование описывает, **как** в системе будут реализованы :

- Моделирование связей и взаимодействий

Анализ проблемы, используя объектно-ориентированный анализ (Object-Oriented Analysis - OOA)

- Дюк продает одежду (рубашки) из своего электронного каталога. Каждые 3 месяца каталог рассылается покупателям в виде электронных писем. Бизнес растет на 30% в год.
- Задача: Необходима новая система ввода заказов

Процесс получения заказа от Дюка

Абстракция

- Объекты (Objects) - связанный набор атрибутов и поведения в классе.
- Определены следующие ключевые свойства объектов:
  - *Поведение* (behavior) объекта — что с ним можно делать и какие методы к нему можно применять.
  - *Состояние* объекта — чем данный объект отличается от других, характеризующихся таким же поведением.

Классы как схемы объектов

На производстве:

- проект описывает, каким образом построены физические устройства.

В программном обеспечении:

- класс определяет структуру и поведение (данные и код), которые будут совместно использоваться набором объектов.

Каждый объект данного класса содержит структуру и поведение, которые определены классом, как если бы объект был “отлит” в форме класса.

Пример. Создание пользовательского класса

- класс Shirt (рубашка) является типом, который представляет и определяет продаваемые рубашки
- экземпляр класса Shirt представляет собой виртуальную сущность рубашки
- класс содержит:
  - атрибуты типа
  - shirtID(артикул рубашки), price (цена), description (описание), size (размер) и color Code (цвет)
  - операции типа

Класс Shirt

Символы, используемые в Java

- Фигурные скобки
- Круглые скобки
- Точка с запятой
- Запятая
- Одинарные кавычки
- Двойные кавычки
- Двойная косая черта

Объявление класса

- Синтаксис объявление класса:

```
<modifier>* class <class_name> {
  <attribute_declaration>*
  <constructor_declaration>*
  <method_declaration>*
}
```

Объявление атрибутов

- Синтаксис объявления атрибутов:

```
<modifier>* <type> <name> [ = <initial_value>];
```

- Пример:

```
- public class Shirt {
-   public int shirtID = 0; // Default ID
  for the                   // shirt
-   public String description =
  "-description";
-   // The color codes are R=Red, B=Blue,
      // G=Green, U=Unset
```

Объявление методов

- Синтаксис метода:

<modifier>*
<return type>

Использование конструкторов

Конструктор – это специальный метод, предназначенный для создания и инициализации экземпляра класса.

Особенности применения конструкторов:

- Имя конструктора совпадает с именем класса.
- Класс может иметь несколько конструкторов.
- Конструктор может иметь один или несколько параметров либо не иметь их вовсе.
- Конструктор не возвращает никакого

Использование конструкторов

Пример:

- public class Shirt {

```
public Shirt(){
    shirtID = 0;
    description = "-description";
    colorCode = 'U';
    price = 0.0;
```

Конструктор по умолчанию

Если явный конструктор не указан, Java автоматически предоставит конструктор, используемый по умолчанию:

- по умолчанию конструктор без аргументов
- тело конструктора по умолчанию пустое

По умолчанию можно создавать экземпляры объектов с помощью оператора **new** без необходимости указывать конструктор в теле класса.

Доступ к атрибутам и методам объекта

```
Shirt shirt1 = new Shirt();

Shirt shirt2 = new Shirt();
```

- *dot* нотация:

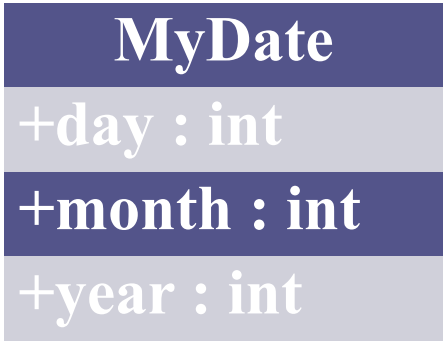
```
<object>.<member>
```

- Используется для доступа к атрибутам и методам.
- Примеры dot нотации:

```
shirt1.displayInformation();
```

Сокрытие информации

Проблема:



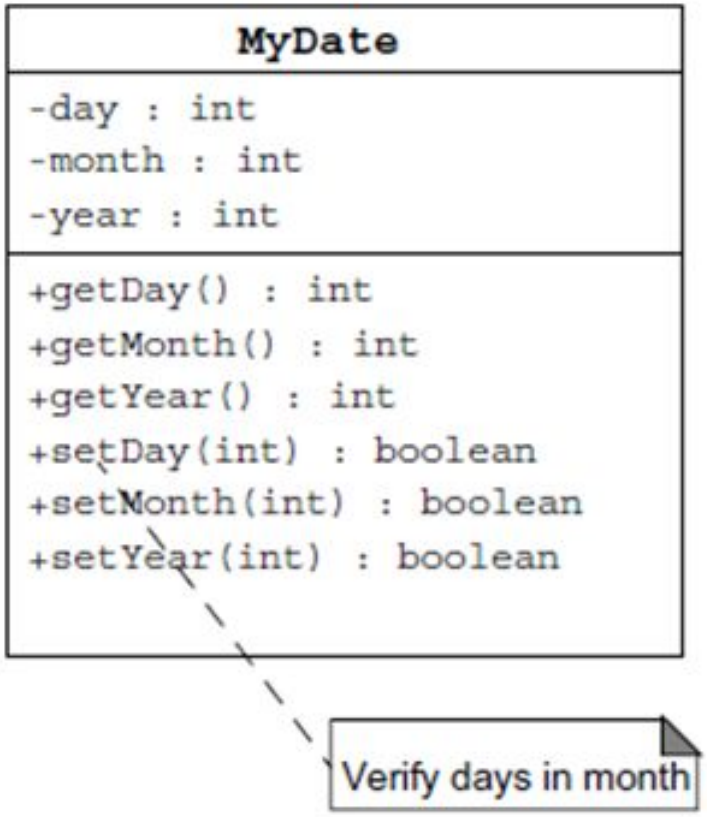
- Клиентский код имеет прямой доступ к внутренним данным:

```
MyDate d = new MyDate();

d.day = 32; // неверная дата

d.month = 2; d.day = 30; // правдоподобно, но
```

Сокрытие информации

Решение:

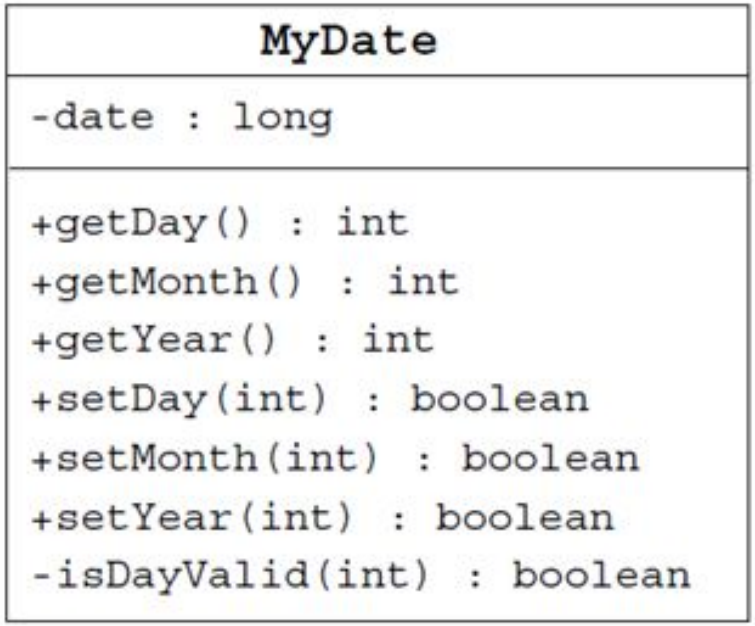
- Клиентский код должен использовать методы установки (setter) и получения (getter) доступа к внутренним данным:



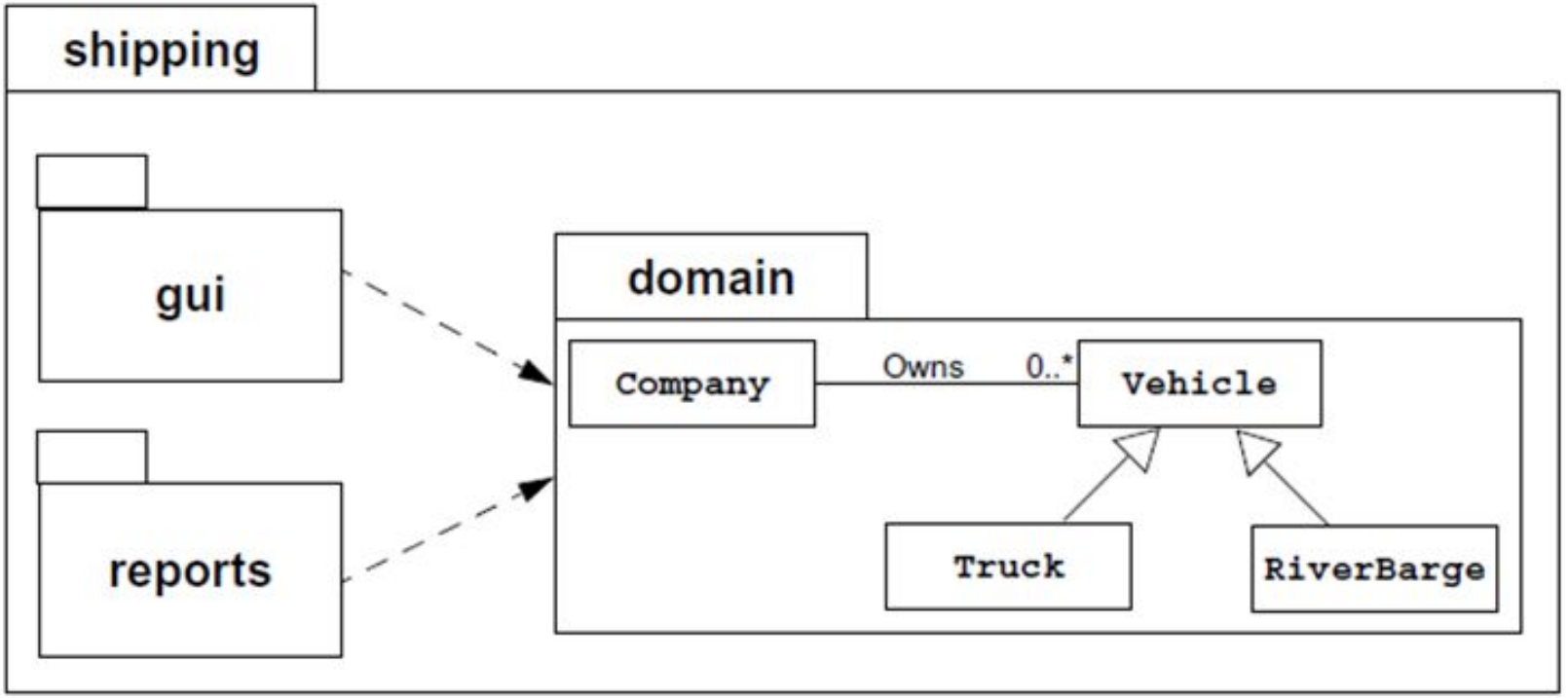

```
MyDate d = new MyDate();

d.setDay(32); // если недействительный день,
              // возвращает false
```

Инкапсуляция

- скрывает детали реализации класса
- вынуждает пользователя использовать интерфейс для доступа к данным
- делает код более легким в обслуживании



Пакеты (package)



Пакеты (package)

- Пакет – специальное средство группировки классов.
  - обеспечивают независимые пространства имен (namespaces), а также ограничение доступа к классам.
  - облегчают организацию работы и позволяют отделить классы, созданные одним разработчиком, от классов, разработанных другими программистами.
  - классы всегда задаются в каком-либо пакете.
  - в составе пакета можно создавать подпакеты и

Макет исходного файла (File Layout)

- Синтаксис исходного файла:

```
[<package_declaration>]
<import_declaration>*
<class_declaration>+
```

- Например, файл VehicleCapacityReport.java:

```
package shipping.reports;


import shipping.domain.*;
import java.util.List;
```

Добавление классов в пакеты

- Синтаксис:

```
package <top_pkg_name>[.<sub_pkg_name>]*;
```

- Пример:

```
package shipping.gui.reportscreens;
```

- Объявление пакета указывается в начале исходного файла
- Если пакет не объявлен, то класс помещается в пакет по умолчанию
- Имена пакетов могут быть иерархическими, в качестве разделителя имен пакетов

Импортирование классов

- Доступ к классам из других пакетов:

```
java.util.Date today = new java.util.Date();
```

или путем импортирования

- Синтаксис выражения import:

```
import <pkg_name>[.<sub_pkg_name>]*.<class_name>;
```

или

```
import <pkg_name>[.<sub_pkg_name>]*.*;
```

- Примеры:

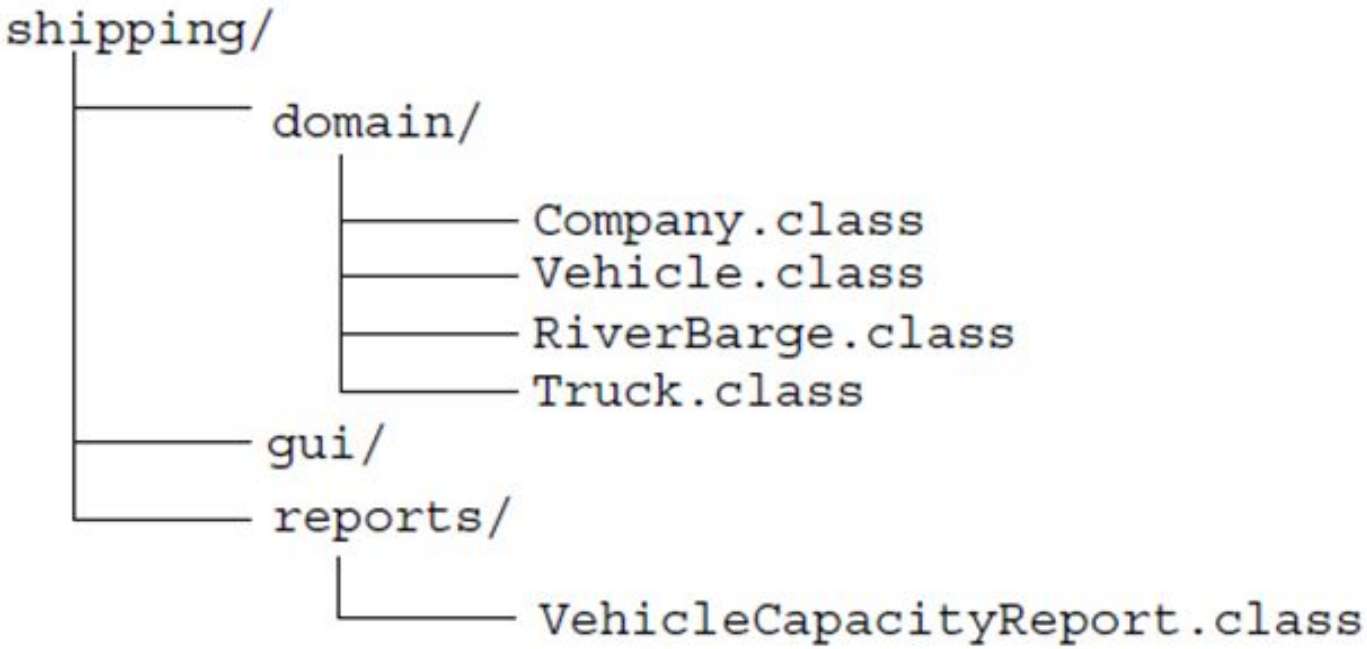
```
import java.util.List;

import java.io.*;
```

Особенности импорта

- Оператор import со звездочкой можно применять для импортирования только одного пакета. Нельзя использовать обозначение import java.*
- В большинстве случаев импортируется весь пакет, независимо от его размера. Но возможен конфликт имен. Например, и пакет java.util, и пакет java.sql содержат класс Date
- Импортируются только имена файлов, находящихся точно на уровне указанного пакета. Импорта имен из вложенных в него пакетов не происходит.

Расположение каталогов и пакетов

- Пакеты хранятся в дереве каталогов содержащего имя пакета
- Пример. Пакеты приложения shipping



Компиляция с помощью опции -d компилятора

cd JavaProjects\ShippingPrj\src

javac -d C:\1 shipping\domain*.java

C:\1 – папка, в которую будут помещены классы пакета (*.classes)

shipping\domain*.java – пакет, который необходимо компилировать

Запуск приложения:

Документирование исходного кода

- **javadoc** – средство обработки внедренных в исходный код комментариев и создания для класса справочных HTML-файлов
- javadoc извлекает информацию о следующих элементах:
  - пакетах
  - классах и интерфейсах, объявленных как public
  - методах, объявленных как public или protected
  - полях, объявленных как public или protected

Специализированные комментарии

//* ..... */

Пример:

/**

* Creates new form GUI_application

*/

Имеется два типа кода внутри блока документационного комментария:

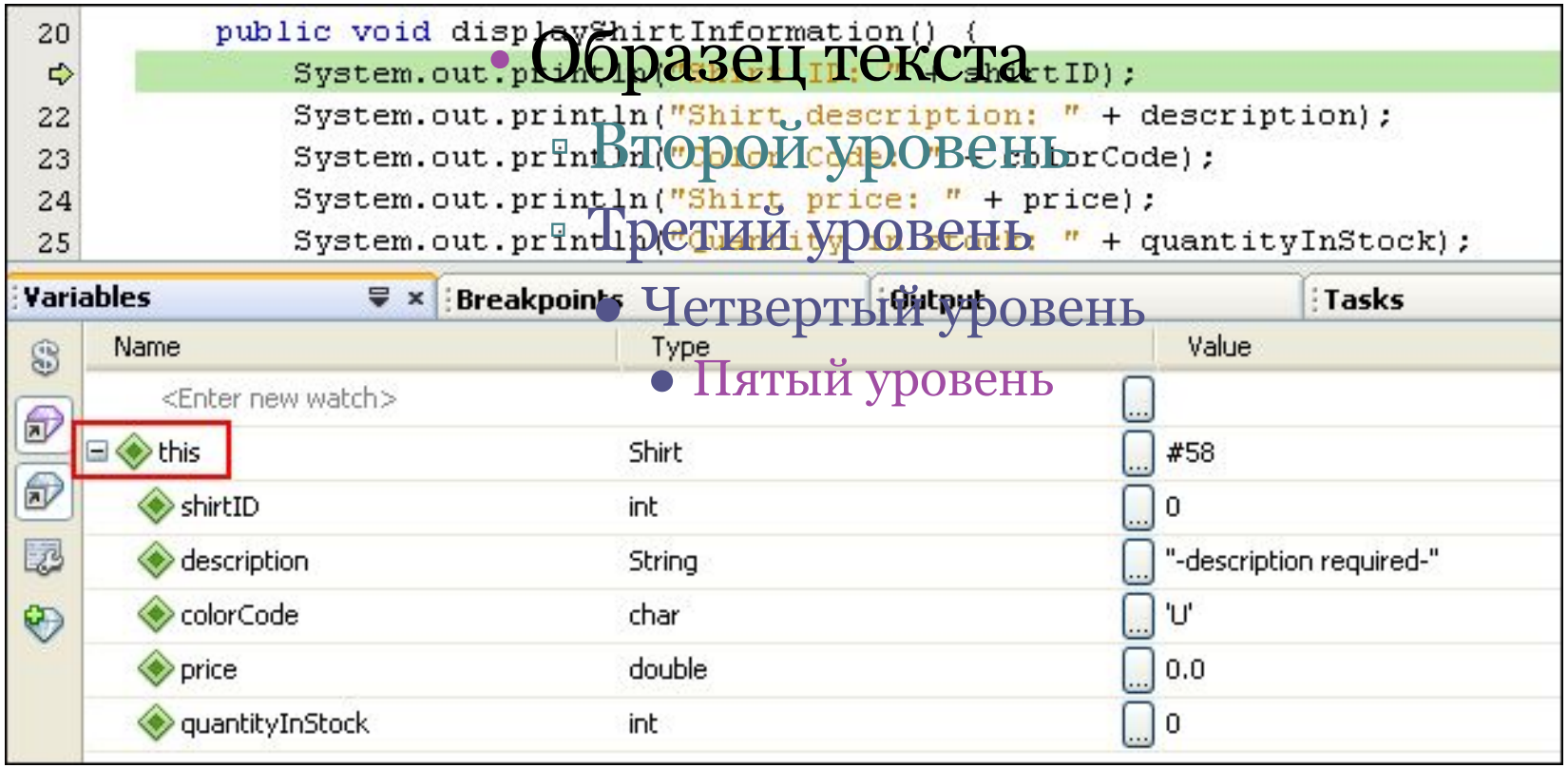
- HTML-текст (<ul>, <li>,
, ...);
- Метаданные (команды документации

Команды документации (метаданные)

Символ метаданных	***Назначение***
@see (“смотри”)	применяется для создания в документе гиперссылок на другие комментарии. Можно использовать для любых конструкций (классов, методов и т.д.)
@version (“версия”)	информация о версии. Используется для классов и интерфейсов.
@author (“автор”)	Информация об авторе. Используется для классов и интерфейсов.
@since (“начиная с”)	Информация о версии JDK, начиная с которой введён или работоспособен класс или интерфейс.
@param (parameter -“параметр”)	- информация о параметре метода. Комментарий /** @param … */ ставится в месте декларации метода в списке параметров перед соответствующим параметром.
@return (“возвращает”)	информация о возвращаемом методом значении и его типе.
@deprecated (“устаревшее”)	информация о том, что данный метод устарел и в последующих версиях будет ликвидирован.

- Образец текста
  - Второй уровень
    - Третий уровень
      - Четвертый уровень
        - Пятый уровень

Работа с Отладчиком (Debugger) в IDE



- Образец текста
  - Второй уровень
    - Третий уровень
      - Четвертый уровень
        - Пятый уровень

Тест

- Выберете объявление класса которое поддерживается в Java

a. `class Shirt`

b. `public Class 501Pants`

c. `public Shirt`

d. `public Class Pants`