ORACLE

Объектно-ориентированное программирование в Oracle PL/SQL

Игорь Мельников
Старший консультант по базам данных

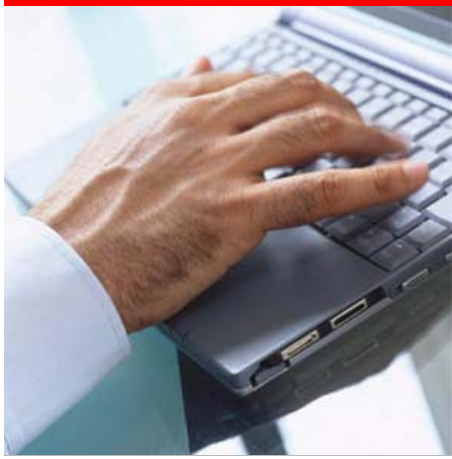
План

- Реализация ООП в Oracle PL/SQL
- Рефлексия типов в PL/SQL
- Паттерны проектирования в PL/SQL
- СУБД Oracle Database как фабрика и репозитарий объектов
- Пример реализации универсальной библиотеки типов PL/SQL

Реализация парадигм ООП в PL/SQL

Парадигмы ООП в PL/SQL

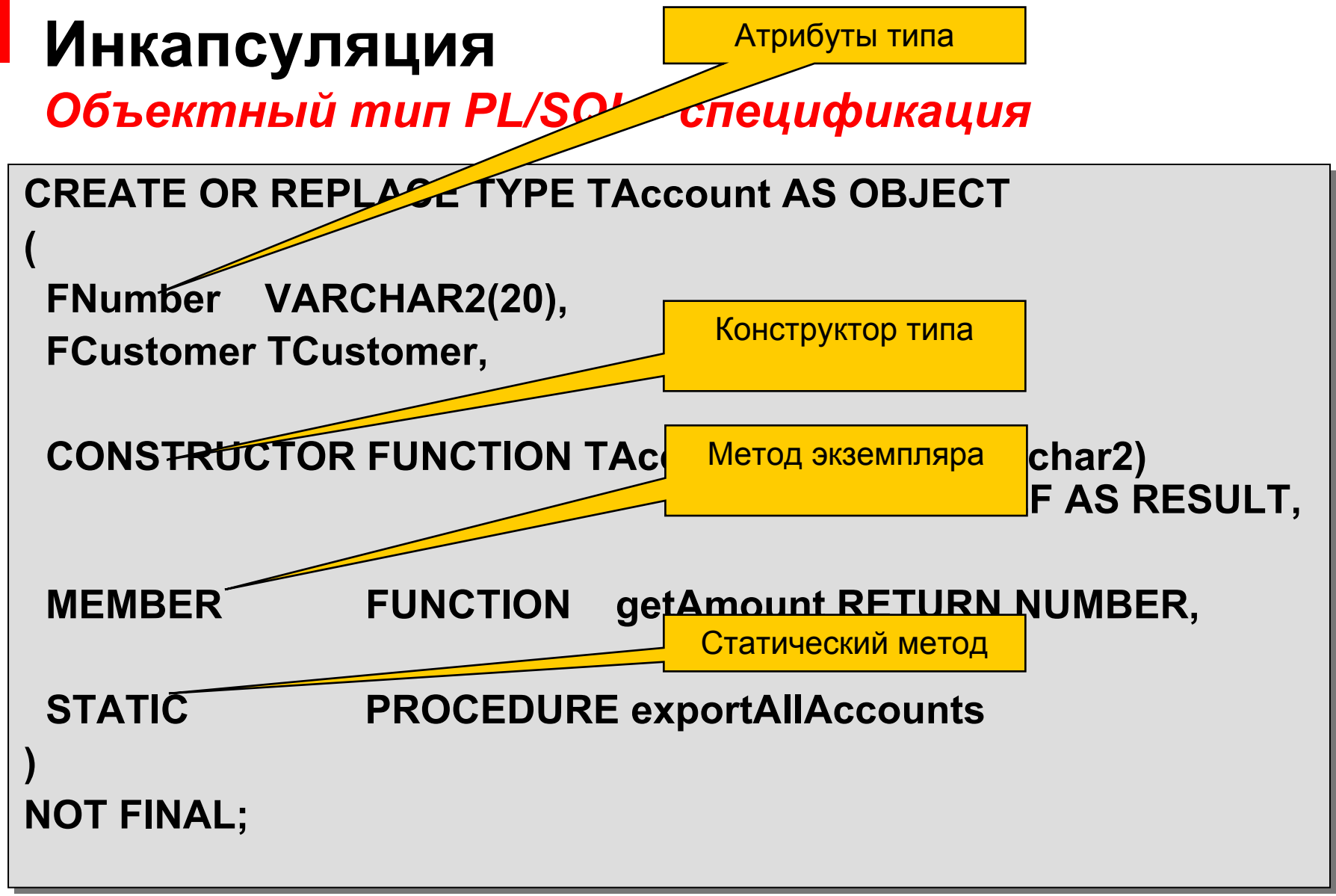
Реализация в PL/SQL



- Инкапсуляция
- Наследование
- Полиморфизм
- Ограничения в реализации
- Дополнительные возможности
  - Приведение типов
  - Перегрузка операторов сравнения
  - Коллекции объектов
  - Хранение объектов в СУБД
  - Ссылки на объекты (OBJECT REF)

Инкапсуляция

Объектный тип PL/SQL – спецификация




```
CREATE OR REPLACE TYPE TAccount AS OBJECT
(
  FNumber    VARCHAR2(20),
  FCustomer TCustomer,

  CONSTRUCTOR FUNCTION TAc                    char2)
                                              F AS RESULT,

  MEMBER          FUNCTION    getAmount RETURN NUMBER,

  STATIC          PROCEDURE exportAllAccounts
)
NOT FINAL;
```

Инкапсуляция

Объектный тип PL/SQL – реализация (тело)

```
CREATE OR REPLACE TYPE BODY TAccount IS

  CONSTRUCTOR FUNCTION TAccount(pAccNo in varchar2) RETURN SELF AS
        RESULT IS
  BEGIN
    self.FNumber := pAccNo;
    self.load;

    RETURN;
  END;

  MEMBER            FUNCTION    getAmount RETURN NUMBER IS
    xResult NUMBER(20,2);
  BEGIN
      … … … … …
    return v_xResult;
  END;
END;
```

Переменные объектного типа

Использование

```
DECLARE
  xCreditAcc    TAccount;
  xDebetAcc     TAccount;
  xTransaction TTransaction;
BEGIN
  xCreditAcc := new TAccount('40101810400000010801');
  xDebetAcc := new TAccount('40101810400000020304');

  xTransaction := new TTransaction(xDebetAcc,
                                   xCreditAcc,
                                   100);
  xTransaction.doIt;
  COMMIT;
END;
```

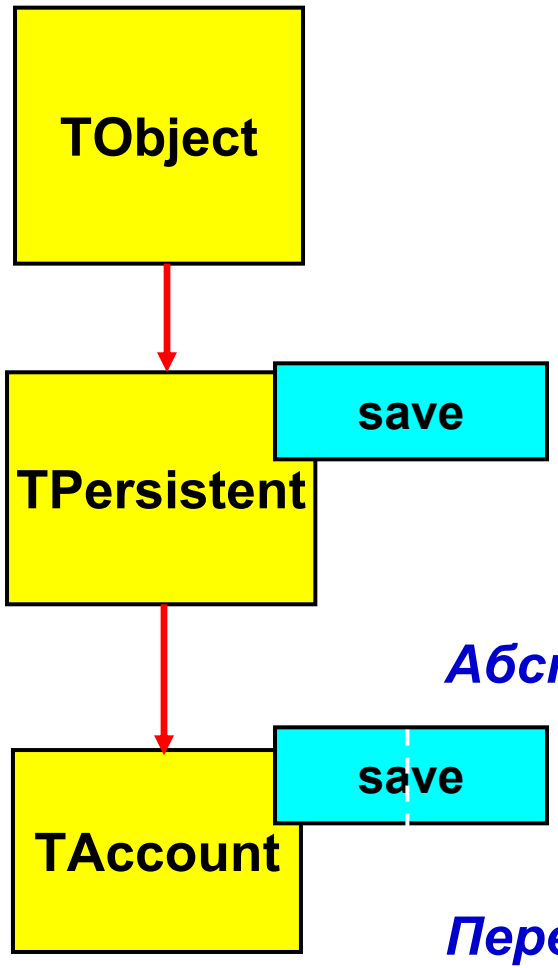
Наследование

Наследование типов PL/SQL - спецификация

```
CREATE OR REPLACE TYPE TAccount UNDER TPersistent
(
  FNumber    VARCHAR2(20),
  FCustomer TCustomer,

  CONSTRUCTOR FUNCTION TAccount RETURN SELF AS RESULT,

  MEMBER          FUNCTION    getAmount RETURN NUMBER,

  STATIC          PROCEDURE exportAllAccounts
)
NOT FINAL;
```

Наследование

Пример: визуальное представление



- Перегрузка метода сохранения объекта (save)

Абстрактный метод

Перегруженный метод

Полиморфизм

Перегрузка методов в типе-потомке PL/SQL

```
CREATE OR REPLACE TYPE TAccount UNDER TPersistent
(
  FNumber    VARCHAR2(20),
  FCustomer TCustomer,

  OVERRIDING MEMBER   PROCEDURE  save,

  OVERRIDING MEMBER   PROCEDURE  remove,

  STATIC         FUNCTION load(v_pId IN NUMBER) RETURN
                                                TAccount
)
NOT FINAL;
```

Не терминальный тип в иерархии !

Полиморфизм

Вызов унаследованного метода в типе-потомке

```
CREATE OR REPLACE TYPE BODY TAccount IS

 OVERRIDING MEMBER   PROCEDURE  save IS
 BEGIN
   (SELF AS TPersistent).save;
   INSERT INTO Accounts …
   … … … … …
 END;

  … … … … …
END;
```

Ограничения в ООП PL/SQL

- Нет разграничения видимости членов типов (все PUBLIC!)
- Не поддерживаются свойства (property)
- Нет атрибутов типа (static members) – можно эмулировать через статический метод и пакет

```
STATIC FUNCTION getRootDict RETURN TDictionary IS
BEGIN
  RETURN dict_service.getRootDict;
END;
```

Дополнительные возможности

Приведение типов - оператор TREAT

```
DECLARE
  v_xObj  TCustomer;
BEGIN
  v_xObj := TREAT(TObjectFactory.Load(123) AS TCustomer);
END;
```

Дополнительные возможности

Перегрузка операторов сравнения

- Необходимо реализовать метод сравнения ORDER

```
/**
 * @param pObject - объект с которым происходит сравнение
 * @return 0, - если объекты равны,<br> 1 - если текущий
     объект больше, чем pObject<br>, -1 - если текущий объект
     меньше, чем pObject
 */
ORDER MEMBER FUNCTION compare(pObject in
    TObject)              RETURN pls_integer
```

- Метод ORDER неявно вызывается вместо операций: **<, <=, >=, =, >**

Дополнительные возможности

Коллекции объектов

- Создаются на уровне СУБД:

```
CREATE TYPE TTableOfString AS TABLE OF VARCHAR2(128);

CREATE TYPE TTableOfAccount AS  TABLE OF TAccount;
```

- Имеют встроенные методы:
  - Неявный конструктор
  - COUNT
  - DELETE
  - EXTEND
  - TRIM

Дополнительные возможности

Операторы для коллекций объектов

- <>, = - проверяет две коллекции на (не) равенство;
- SET – удаляет дубликаты из коллекции;
- [NOT] MEMBER – проверка вхождения элемента;
- MULTISET INTERSECT – пересечение коллекций;
- MULTISET UNION – объединение коллекций;
- MULTISET EXCEPT – из одной вычесть другую;
- IS A [NOT] SET – содержит уникальные элементы;
- NOT SUBMULTISET OF – проверка на подмножество;

Дополнительные возможности

Коллекции объектов - использование

- Для использования операторов с коллекциями, объектный тип должен иметь метод ORDER – для сравнения объектов

```
DECLARE
  xAccArray TAccountArray;
  xAcc      TAccount := new TAccount('101');
BEGIN
  xAccArray := new TAccountArray(new TAccount('101'),
                                 new TAccount('102'));

  IF xAcc  MEMBER  xAccTArray THEN
    println('Is member!');
  END IF;

END;
```

Хранение объектов в БД

Объектные таблицы

- Объектные таблицы

```
CREATE TABLE Accounts OF TAccount;
```

- Колонка реляционной таблицы - объект

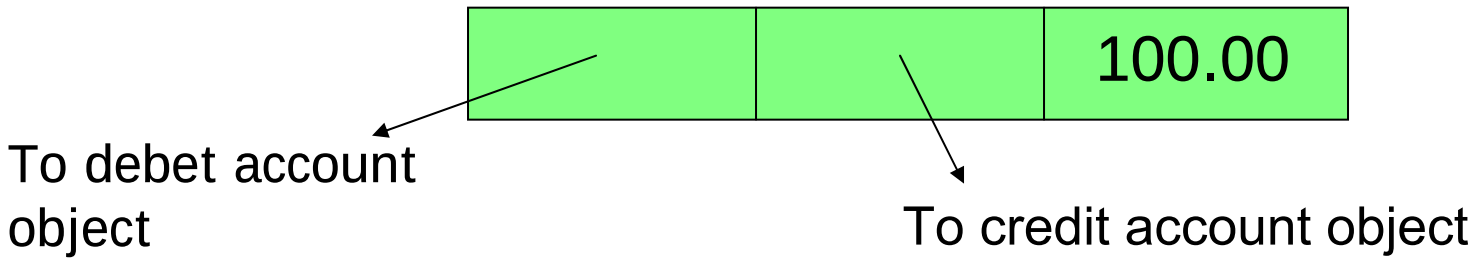
```
CREATE TABLE Users
(
  Id          NUMBER(6),
  Properties TObjectProperties
);
```

Ссылки на объекты

Object REF

- Ссылка на хранимый объект в БД

```
CREATE TYPE TTransaction AS OBJECT
(
  DebetAcc  REF TAccount,
  CreditAcc  REF TAccount,
  Amount     NUMBER
);
```



Ссылки на объекты

Дополнительные возможности

- Дополнительный пакет UTL_REF
  - DELETE_OBJECT
  - LOCK_OBJECT
  - SELECT_OBJECT
  - UPDATE_OBJECT
- Эмуляция REF через объектные представления:

```
CREATE VIEW v$AccountsRef OF TAccount
  WITH OBJECT IDENTIFIER(Id)
SELECT
  a.Id            AS OID,
  a.FNumber
FROM
  accounts a;
```

Рефлексия объектных типов в PL/SQL

Generic PL/SQL types

Обобщенные типы

- ANYTYPE – получение информации о типе, а также создание анонимных типов
- ANYDATA – представляет собой экземпляр любого типа
- ANYDATASET - представляет собой множество экземпляров какого-либо типа (*generic recordset*)
- DBMS_TYPES – пакет (*package*) предоставляет набор констант и исключений для работы с обобщенными типами

Тип ANYTYPE

Работа с уже существующими типами в БД

```
STATIC FUNCTION GETPERSISTENT(
                schema_name IN VARCHAR2,
                type_name   IN VARCHAR2,
                version     IN VARCHAR2 DEFAULT NULL)
                                RETURN ANYTYPE;
```

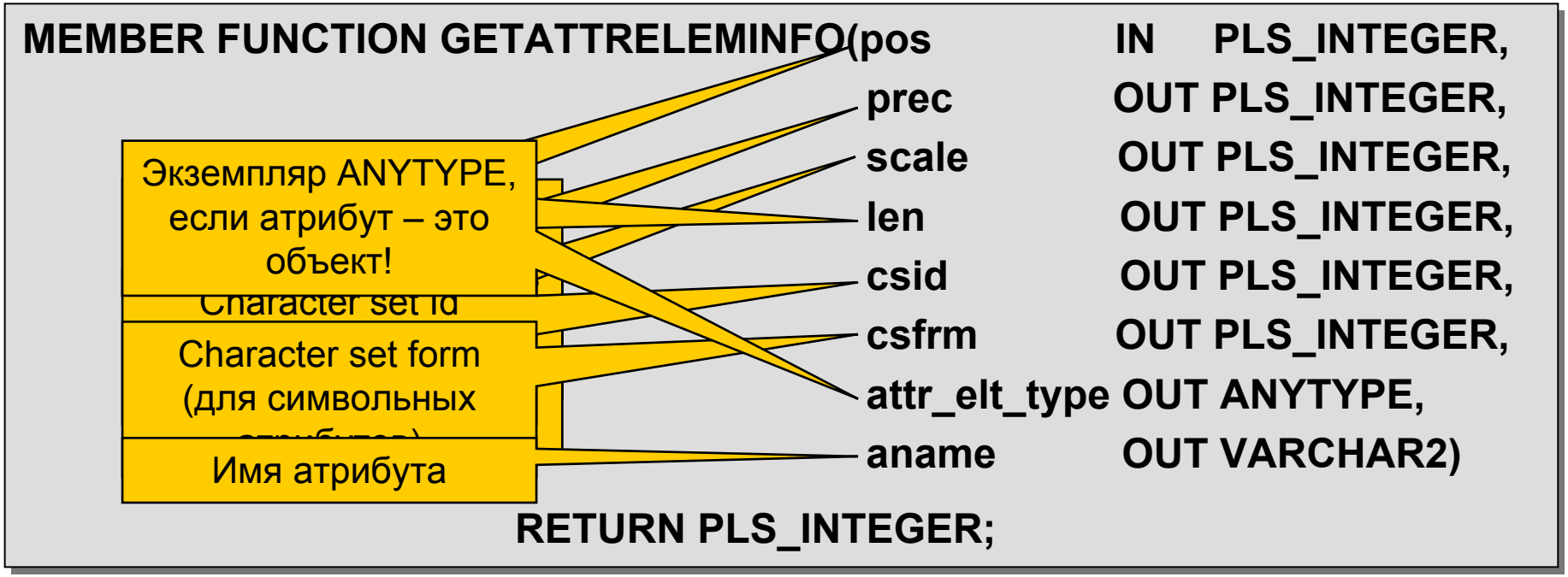
- Статическая функция GETPERSISTENT – создание экземпляра ANYTYPE по уже существующему в БД типу

```
DECLARE
  v_xType ANYTYPE;
BEGIN
  v_xType := ANYTYPE.getPersistent('MY_SCHEMA','TACCOUNT');
END;
```

В том числе и по примитивным типам!

Тип ANYTYPE

Получение информации об атрибуте



- Возвращает код типа элемента (определены в пакете DBMS_TYPES)

Тип ANYTYPE

Создание анонимного типа

```
SQL> DECLARE
  2     xType AnyType;
  3  BEGIN
  4     AnyType.BeginCreate(DBMS_TYPES.TYPECODE_OBJECT,
  5                                    xType);
  6     xType.addAttr(aname    => 'AccountNumber',
  7                 typecode => DBMS_TYPES.TYPECODE_VARCHAR2,
  8                 prec     => 0,
  9                 scale    => 0,
 10                  len      => 0,
 11                  csid     => 0,
 12                  csfrm    => 0);
 13     xType.EndCreate;
 14  END;
```

Тип ANYDATA

Получение информации об экземпляре типа

- **BEGINCREATE –** начало создания нового экземпляра AnyData
- **ENDCREATE –** завершение создания экземпляра AnyData
- **PIECEWISE –** установка режима создания экземпляра для не примитивных объектов
- **SET* –** установить значение атрибута
- **GET* –** получение значения атрибута
- **GETTYPENAME–** получить имя типа (для типов уже существующих в БД)
- **GETTYPE –** получение экземпляра ANYTYPE описывающего тип объекта
- **CONVERTOBJECT –** получение экземпляра ANYDATA из любого объекта

Два способа создания экз. ANYDATA

1-ый: с помощью вызова методов CONVERT

```
SQL> DECLARE
  2     xType        ANYTYPE;
  3     xAccount     TAccount := new TAccount('123');
  4     xVar         ANYDATA;
  5  BEGIN
  6     xVar := ANYDATA.ConvertObject(xAccount);
  7     DBMS_OUTPUT.PUT_LINE(xVar.getType(xType));
  8     DBMS_OUTPUT.PUT_LINE(xVar.getTypeName);
  9  END;
 10  /
108
ABC.TACCOUNT
```

Два способа создания экз. ANYDATA

2-ой: вызов методов BeginCreate(), Set(), EndCreate*

```
SQL> DECLARE
  2    xObject AnyData;
  3    xType    AnyType := TAccount.getAnyType;
  4    xAcc     TAccount;
  5  BEGIN
  6    AnyData.BeginCreate(xType,xObject);
  7    xObject.PieceWise();
  8    xObject.setVarchar2('MyAccNumber'); xObject.setNumber(100);
  9    xObject.EndCreate;
 10
 11    IF xObject.getObject(xAcc) = DBMS_TYPES.SUCCESS THEN
 12      dbms_output.put_line(xAcc.getString());
 13    END IF;
 14  END;
 15  /
AccountNo => MyAccNumber | Amount => 100
```

Наполнение объекта по атрибутам, а не сразу!

Тип ANYDATASET

Обобщенные выборки

- Основное использование – реализация обобщенных табличных функций,структура курсора которых неизвестна на этапе компиляции

```
CREATE OR REPLACE FUNCTION
 getReport(pType in number) RETURN AnyDataSet
   PIPELINED USING TReportRowSet;
```

- Тело табличной функции отсутствует!
- Его заменяет тип **TReportRowSet** – реализует интерфейс ODCITable

Тип ANYDATASET

Интерфейс ODCITable – информация для SQL-Engine

- ODCITableDescribe – возвращает структуру курсора
- ODCITableStart – открытие курсора
- ODCITableFetch – извлечение записи
- ODCITableClose – закрытие курсора
- ODCIPrepare – получение контекста выполнения курсора

Интерфейс ODCITable

Пример

```
create or replace type TReportRowSet as object  (

 constructor function TReportRowSet(v_pType in varchar2 := null)  return self as
       result,
 static function ODCITableStart(sctx  in out nocopy TReportRowSet,
                                v_pType  in     varchar2 := null)    return number,
 member function ODCITableFetch(self  in out nocopy TReportRowSet,
                                nrows in     number,
                                rws   out    AnyDataSet)          return number,
 static function ODCITablePrepare(sctx     out nocopy TReportRowSet,
                                tf_info  in  Sys.ODCITabFuncInfo,
                                v_pType  in  varchar2 := null)     return number,
 member function ODCITableClose(self in TReportRowSet)           return number,
 static function ODCITableDescribe(rtype   out AnyType,
                                v_pType in  varchar2 := null)     return number,
 static function getRecordType(v_pType in  varchar2 := null)       return AnyType
```

 http://www.oracle.com/global/ru/oramag/nov2004/gen_dev_10g.html

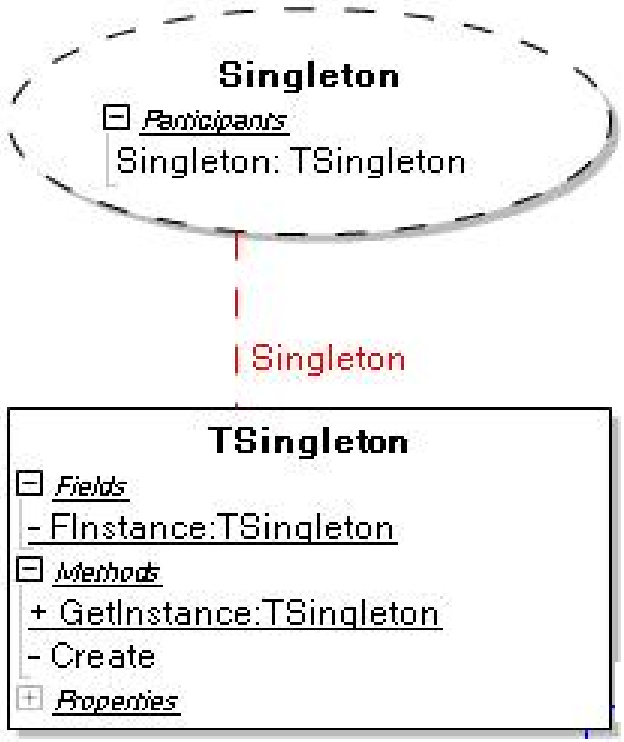
Паттерны проектирования в PL/SQL

Шаблон “Одиночка” (Singleton)

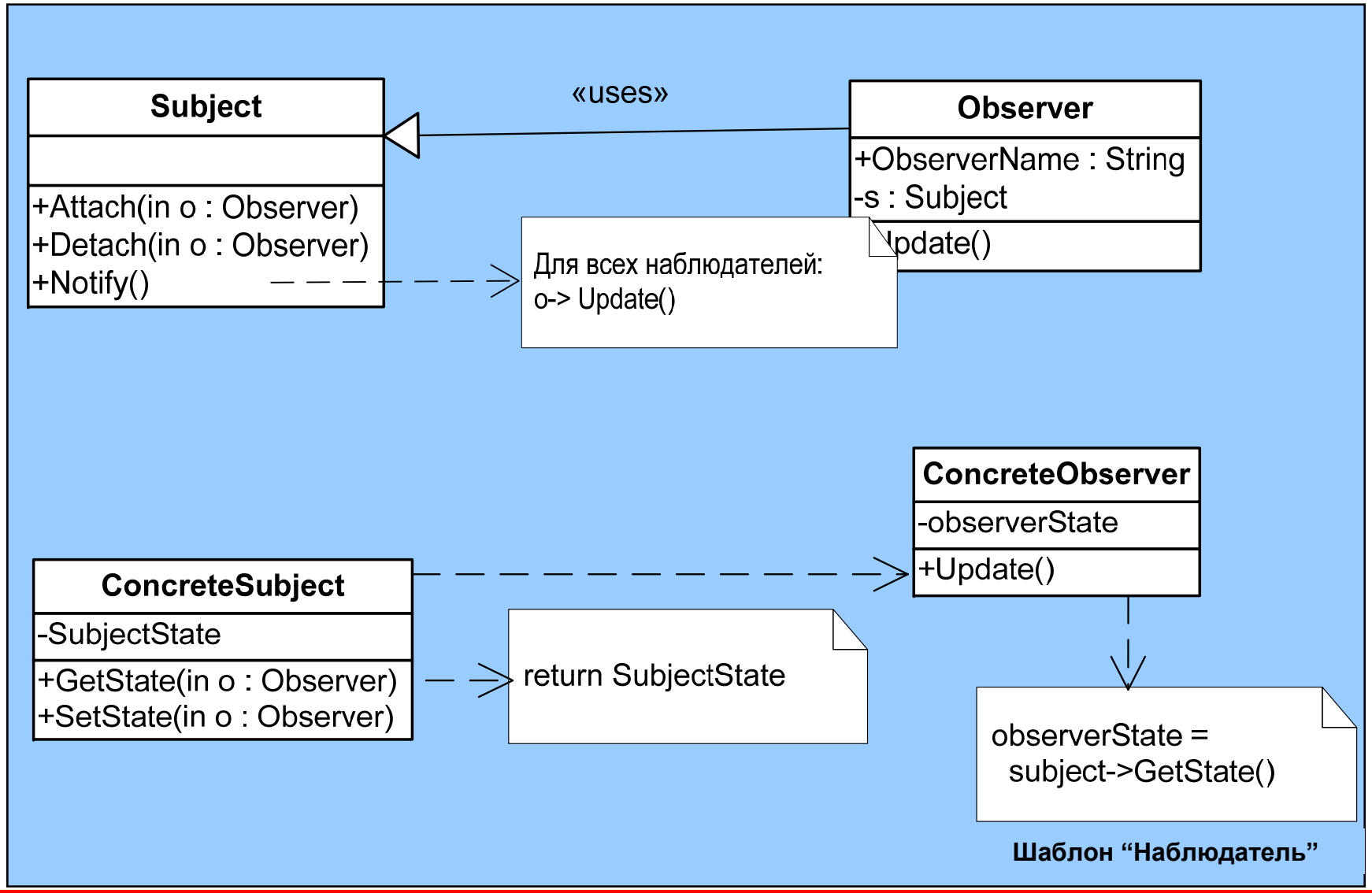
Должен быть только один экземпляр типа

```
CREATE  TYPE BODY TSingleton IS

 STATIC FUNCTION getInstance RETURN
                                TSingleton IS
 BEGIN
  RETURN singleton_service.getInstance;
 END;
```

Экземпляр хранится в глобальной переменной пакета !



Шаблон “Наблюдатель” (Observer)



Шаблон “Наблюдатель” (Observer)

Реализация в PL/SQL (спецификация)

```
CREATE TYPE TArrayOfObserver AS TABLE OF TObserver;

CREATE TYPE TSubject AS OBJECT
(
  FObservers TArrayOfObserver,

  MEMBER PROCEDURE  attach(pObserver IN OUT NOCOPY TObserver),

  MEMBER PROCEDURE  detach(pObserver IN OUT NOCOPY TObserver),

  MEMBER PROCEDURE  notify,

  … … …
);
```

Шаблон “Наблюдатель” (Observer)

Реализация в PL/SQL (тело)

```
CREATE TYPE BODY TSubject IS

 MEMBER PROCEDURE  attach(pObserver IN OUT NOCOPY TObserver) IS
 BEGIN
   FObservers.extend(1);
   FObservers(FObservers.count) := pObserver;
 END;

 MEMBER PROCEDURE  detach(pObserver IN OUT NOCOPY TObserver) IS
 BEGIN
   IF pObserver MEMBER FObservers THEN
     FObservers.DELETE(findObserver(pObserver));
   END IF;
 END;
```

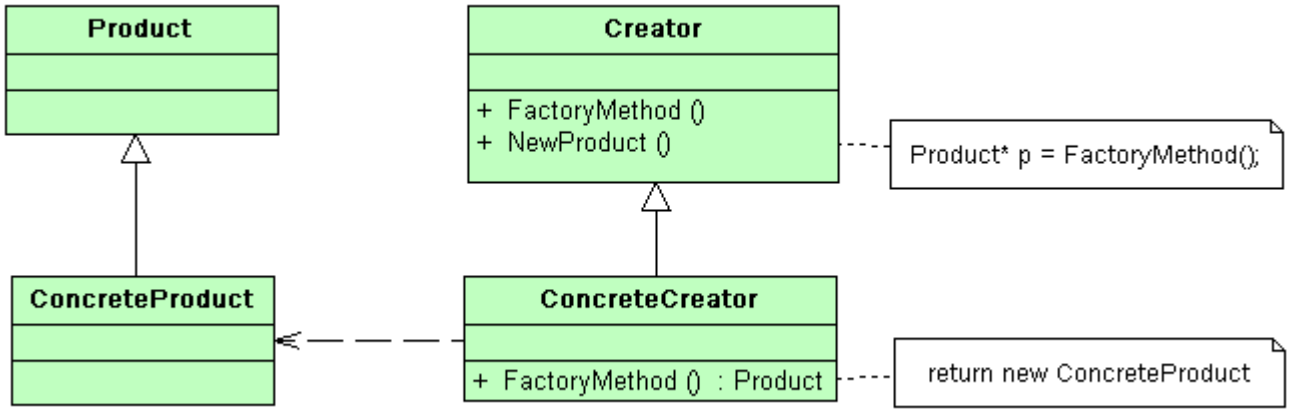
Шаблон “Наблюдатель” (Observer)

Реализация в PL/SQL (тело - продолжение)

```
MEMBER PROCEDURE  notify IS
   xCount SIMPLE_INTEGER := FObservers.count;
BEGIN
  FOR xIndex IN 1..xCount
  LOOP
     FObservers(xIndex).Update;
  END LOOP;
END;
```

Шаблон “Фабричный метод”

Factory Method




```
CREATE TYPE TObjectFactory AS OBJECT
(
  STATIC FUNCTION createObject(pTypeName VARCHAR2) RETURN TObject
  … … …
);
```


http://ru.wikipedia.org/

СУБД Oracle Database как фабрика и репозитарий объектов

Недостатки хранения объектов

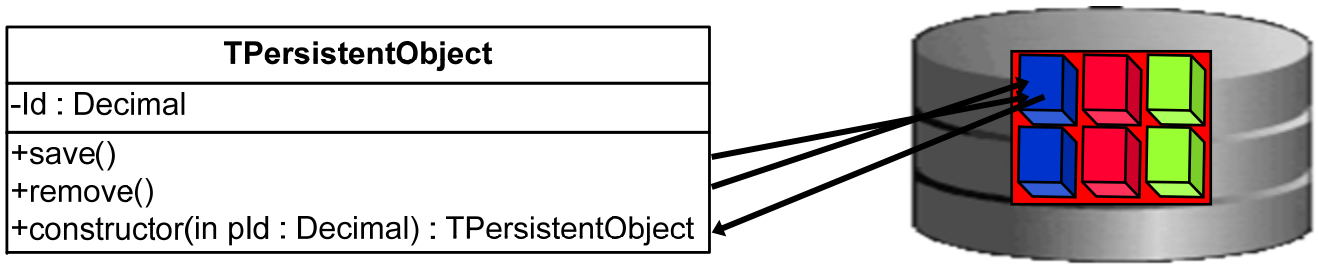
Хранение в объектных таблицах Oracle

- Крайне затруднена модификация исх. кода объектов (есть связь с таблицей!)
- Сложный синтаксис обращения к объектной таблице
- Изменение атрибутов объекта ведет к тяжелой процедуре модификации структуры объектной таблицы

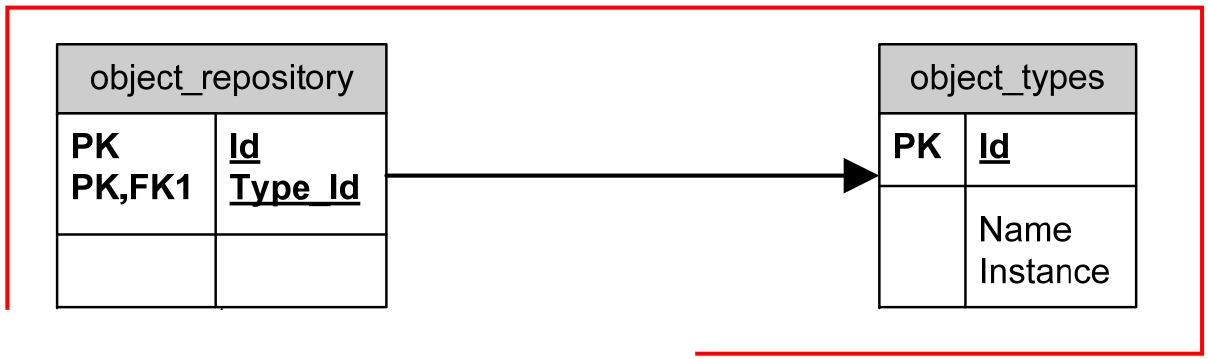
Хранение объектов в реляц. таблицах

Совмещение преимуществ ООП и реляционного подхода

- Хранить Persistent-объекты в обычных реляционных таблицах
- Доступ реализовать в виде обычного реляционного SQL, в коде методов объектного типа
- Когда необходимо, применять обычный реляционный доступ к таблицам

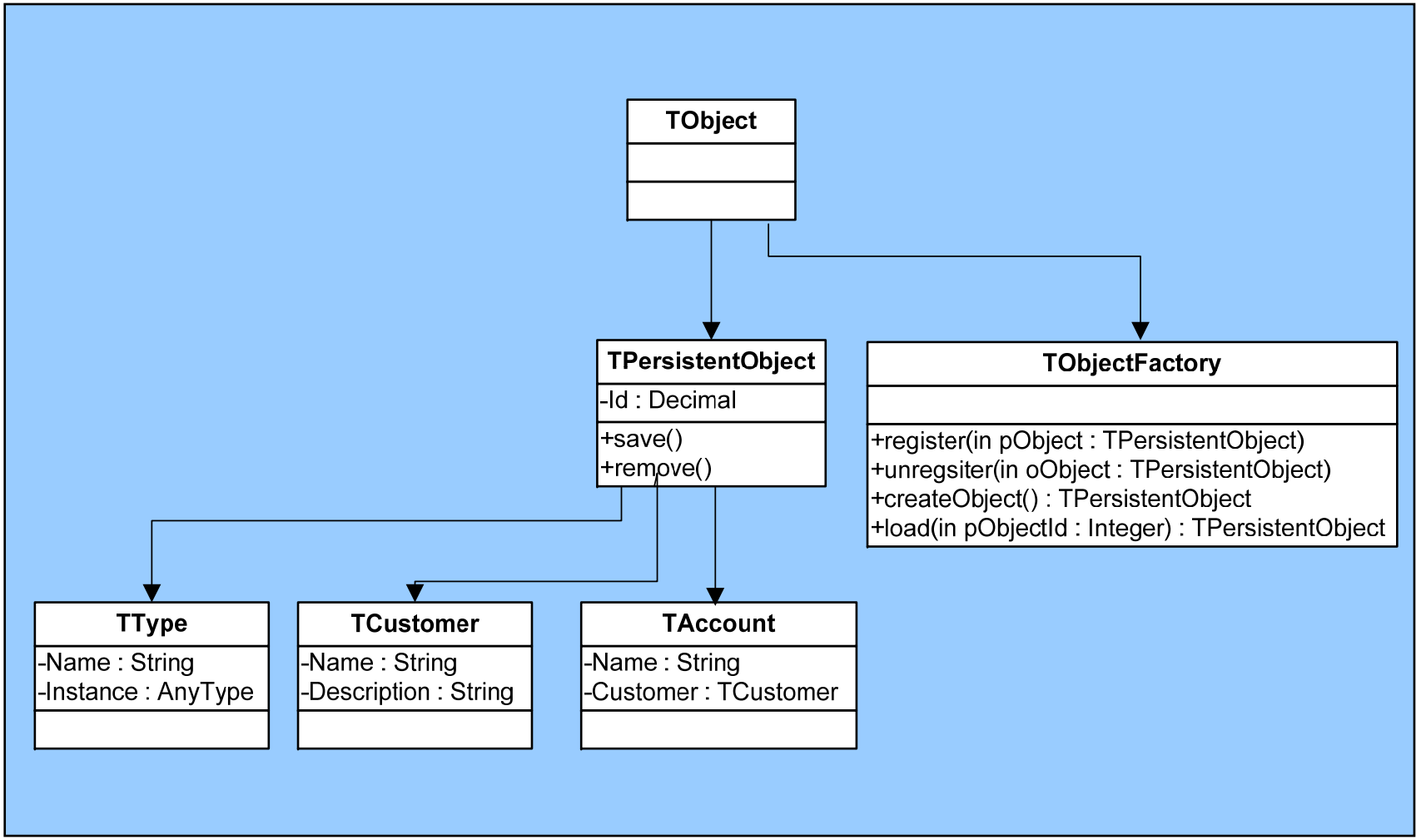


Репозитарий объектов в СУБД Oracle



Репозитарий объектов в Oracle

Базовая иерархия классов



Работа с репозитарием

Тип TPersistentObject

```
CREATE TYPE BODY TPersistentObject IS

  MEMBER PROCEDURE save IS
  BEGIN
    TObjectFactory.register(self);
  END;


  MEMBER PROCEDURE remove IS
  BEGIN
    TObjectFactory.unregister(self);
  END;
  ... ... ...
```

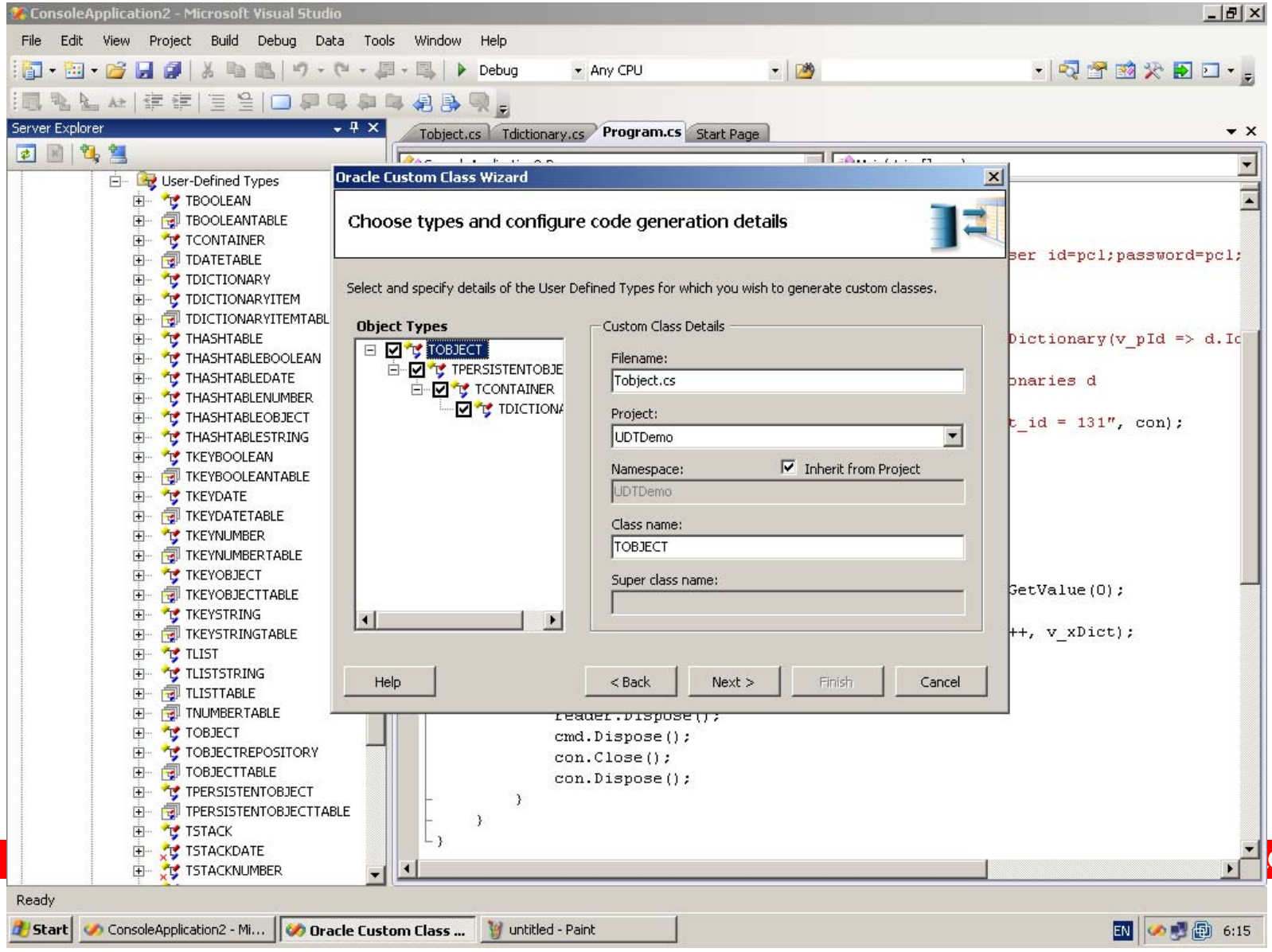
Работа с объектами в приложении

Пример: MS Visual Studio .NET

- Необходима поддержка объектов в средствах разработки приложений!
- Oracle Developer Tools for VS .Net
  - UDT Custom Class Code Generation Wizard
  - Генерирует класс на C#,VB.NET,C++ совпадающий по сигнатуре с типом PL/SQL
  - Отображение типов PL/SQL в классы .NET
  - Клиент прозрачно использует объекты в терминах своего окружения !!!

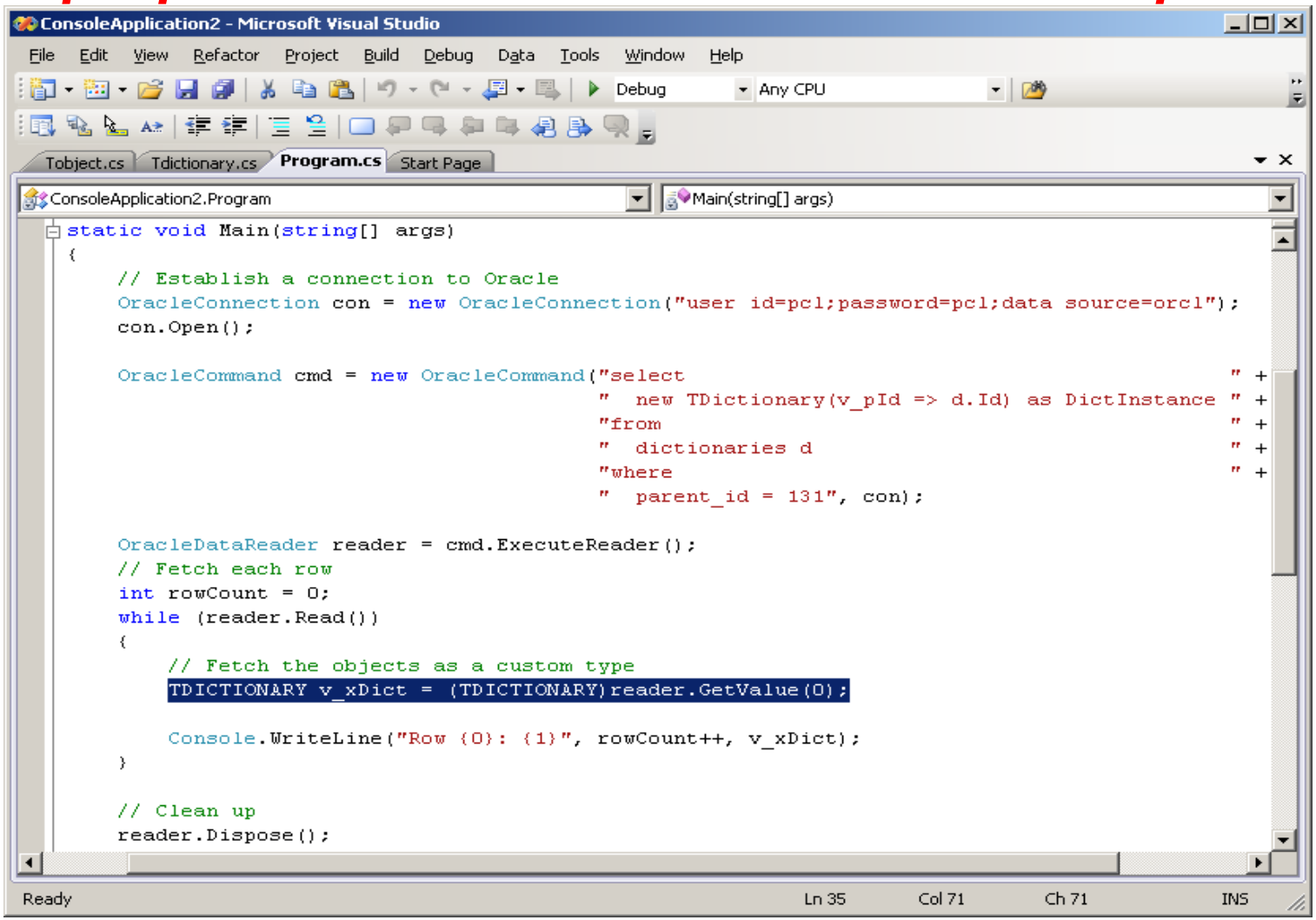
Oracle Developer Tools for VS .NET

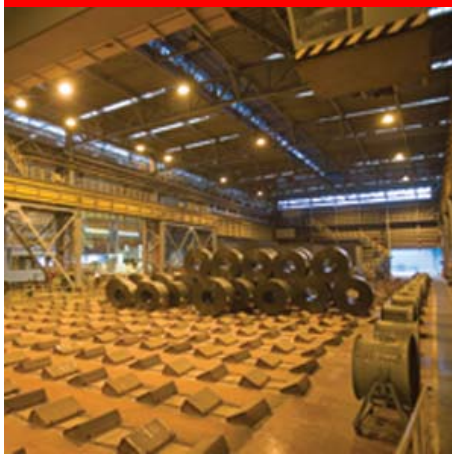
UDT Custom Class Code Generation Wizard



Custom Class Code Generation Wizard

Прозрачное использование объекта в приложении





Пример реализации универсальной библиотеки типов PL/SQL

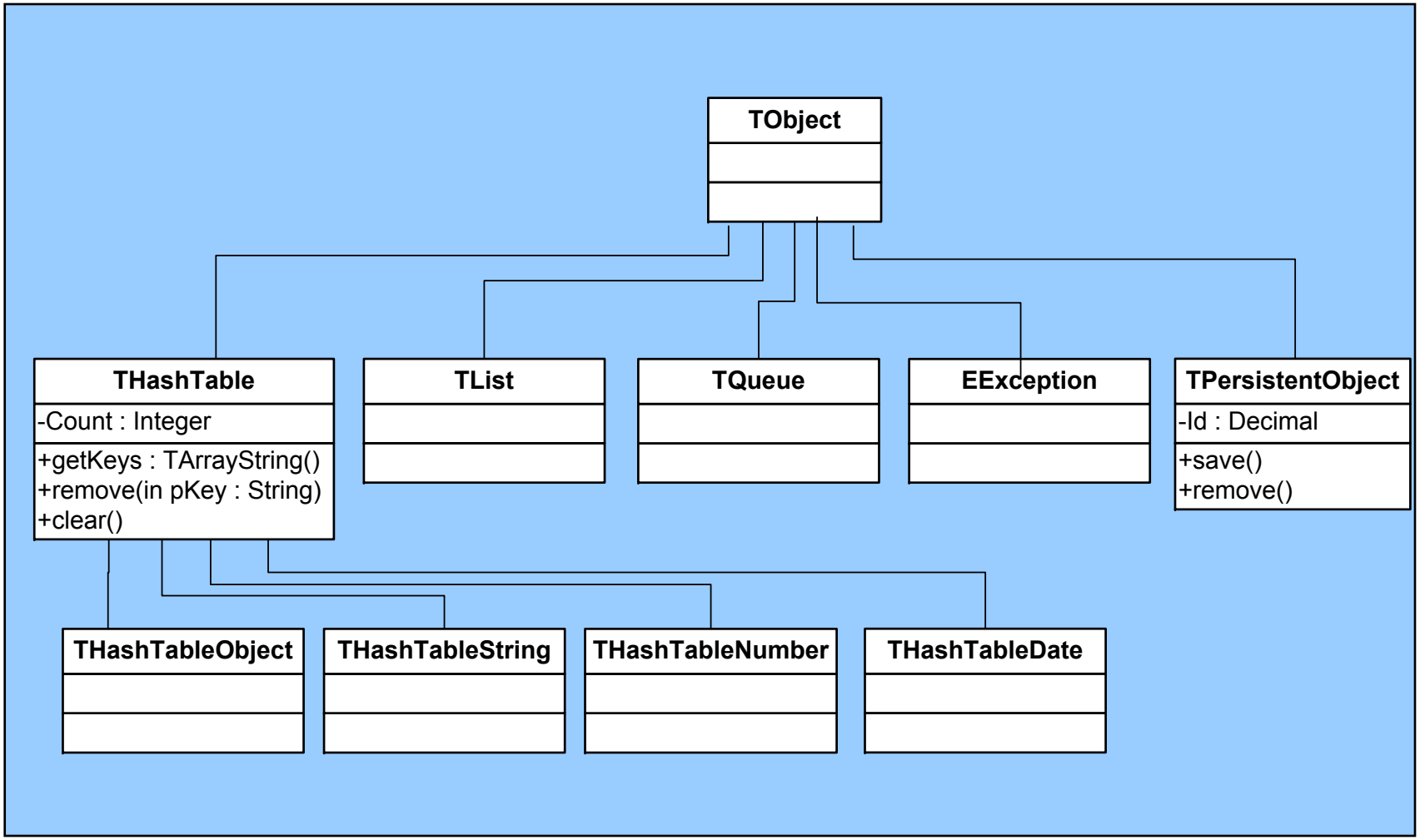
Постановка задачи

Необходимость в библиотеке

- Поддержки репозитария объектов недостаточно !
- Часто приходится повторять один и тот же код:
  - Работа с списками
  - Работа с хеш-таблицами
  - Обработка массива строк
  - Обработка исключений
  - И т.д.
- Необходима библиотека универсальных (проблемно-независимых) типов – по аналогии с JDK, .NET, VCL и т.д.

PCL 2.0

PL/SQL Class Library – набор типов PL/SQL



Работа с списками строк в PCL

Тип TListString

```
DECLARE
  v_xList          TListString := new TListString();
  v_xLineList      TListString := new TListString();
BEGIN
 v_xLineList.setDelimiter(';');
 v_xList.loadFromFile(v_cDataDir,v_cFileName);

 v_xRowCount := v_xList.getCount;
 FOR  v_xIndex IN 1..v_xRowCount
 LOOP
   :v_xCurrLine := v_xList.getItem(v_xIndex);
    v_xLineList.setCommaText(v_xCurrLine);
   :v_xCompanyName :=  v_xLineList.getItem(1);
   .. .. …
```

Обработка исключений

Пример: использование типов-исключений

```
BEGIN
  throw(new EFileNotFoundException('config.ini'));

EXCEPTION
 WHEN exception_service.EFileNotFound THEN
   exception_service.printStack;
END;
/
```

Регистрация объекта исключения в стеке ошибок PCL

Инкапсуляция функций API в типы PCL

Пример: инкапсуляция Oracle AQ в тип PL/SQL

```
CREATE TYPE TQueueObjectAQ UNDER TQueueObject  (
  FTableName varchar2(64),
  FTypeName  varchar2(64),
  OVERRIDING MEMBER FUNCTION peek          return TObject,
  OVERRIDING MEMBER FUNCTION dequeue     return TObject,
  OVERRIDING MEMBER PROCEURE enqueue(pObject TObject),
  MEMBER PROCEURE startQueue,
  MEMBER PROCEURE stopQueue
  … … …
```

- Вместо использования громоздкого и сложного API – набор понятных и коротких типов
- Сокращается исходный код и повышается его читабельность

Работа с объектной моделью MS Excel

Инкапсуляция COM Automation Feature в PL/SQL

```
DECLARE
  v_xAppExcel    TExcelApplication := new TExcelApplication();
  v_xWorkBooks   TWorkbooks;
  v_xWorkBook    TWorkbook;
  v_xWorkSheets  TWorkSheets;
  v_xWorkSheet   TWorkSheet;
  v_xCell        TCell;
BEGIN
  v_xWorkBooks  := v_xAppExcel.WorkBooks;
  v_xWorkBook   := v_xWorkBooks.Open('c:\temp\test.xls');
  v_xWorkSheets := v_xWorkBook.WorkSheets;
  v_xWorkSheet  := v_xWorkSheets.Item('test');
  v_xCell       := v_xWorkSheet.getCell(2,1);
  dbms_output.put_line(v_xCell.getValue);
END;
```

PL/SQL Class Library

Резюме

- Объектно-ориентированная библиотека типов PL/SQL общего назначения
- Инкапсулирует часто используемый функционал:
  - Стеки, списки, очереди, хэш-таблицы, нити (thread), потоки (stream) и т.д.
  - Работа с ОС: файлы, каталоги, запись/чтение файлов сервера
  - Часть функций API Oracle (built-in packages) – AQ, LOB, JOB
- Также реализует поддержку репозитария persistent-объектов и работу с ними
  - Подсистема ведения справочников
  - Подсистема групп объектов
  - Logging (подсистема ведения журналов) – аналог log4j
  - Подсистема обработки обновлений (patching)

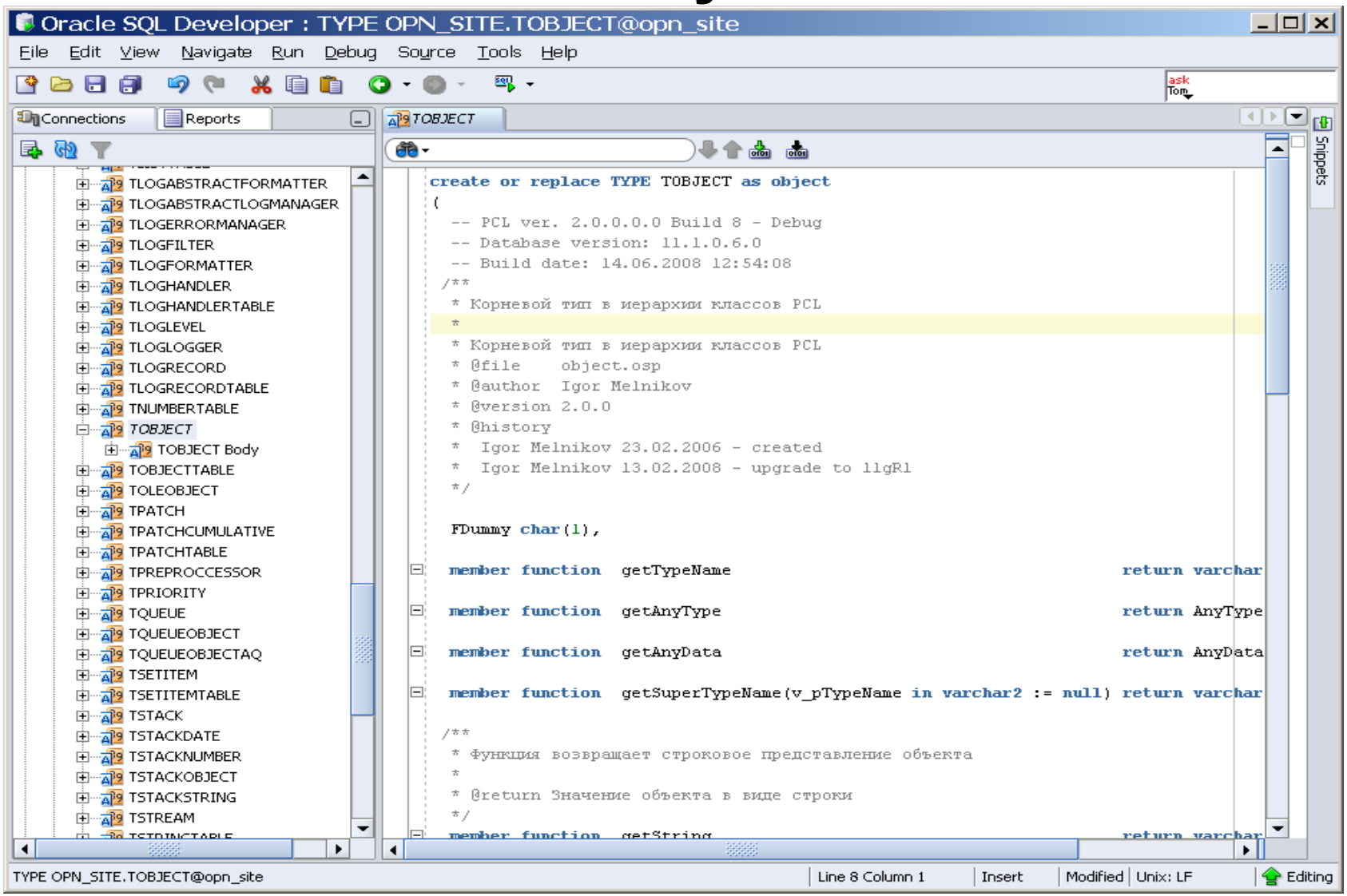
- Объем: ~ 150 типов и ~50 тыс. строк кода PL/SQL

PL/SQL Class Library

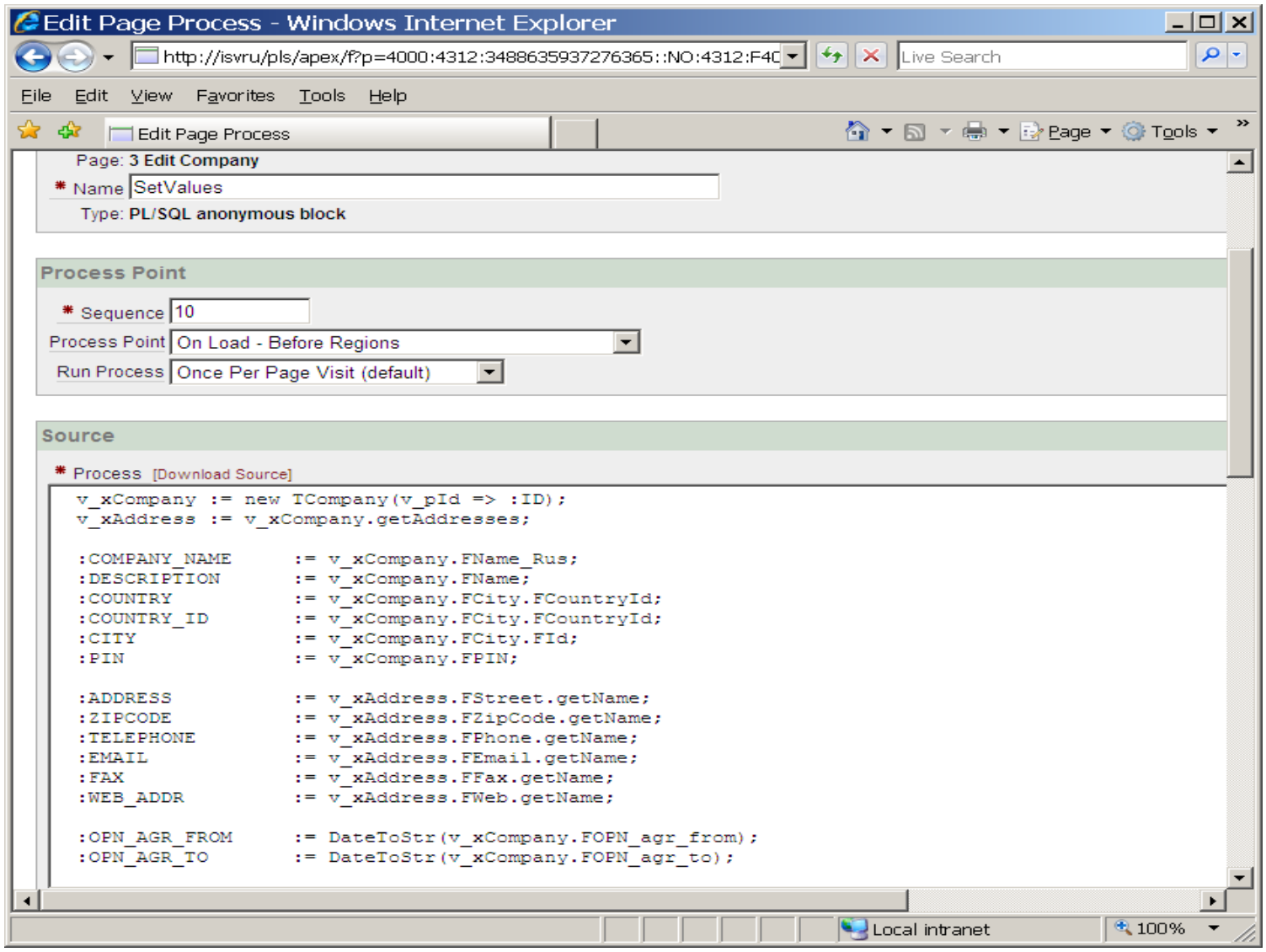
Пример внедрения (success story ☺)

- Заказная разработка: “Система ведения контактной информации”
  - Учет компаний-партнеров и их атрибутов
  - Учет договоров
  - Учет контактов компаний-партнеров
  - Учет компаний-заказчиков
  - Учет контактов компаний-заказчиков
- Используемое окружение
  - Oracle Database EE 11.1.0.6.3
  - Oracle Partitioning Option
  - Oracle Advanced Compression Option
  - Oracle Application Express 3.1.1 + AJAX
- Объем: ~ 70 таблиц, ~200 типов и ~70 тыс. строк кода PL/SQL

PL/SQL Class Library



Использование PL/SQL-типов в APEX



ООП PL/SQL

Заключение

- Поддерживает все основные парадигмы ООП
- Позволяет создавать прикладное ПО в ОО-стиле и получить все преимущества ООП-подхода
- Наличие библиотеки PL/SQL-типов позволяет достичь повторного использования кода, уменьшить сложность проектов и сократить время разработки


Q&A

ORACLE

Игорь Мельников
Старший консультант Oracle СНГ

Email: Igor.Melnikov@oracle.com
Phone: +7 (495) 641 14 00
Direct: +7 (495) 641 14 42
Mobile: +7 (915) 205 26 27


ORACLE®