Содержание

ТОМ I

ТОМ II

Часть VI

ИСПОЛЬЗОВАНИЕ JAVA В ORACLE

ГЛАВА 42

ВВЕДЕНИЕ В JAVA

В этой главе представлен обзор языка Java в той степени, в какой этот язык имеет отношение к приложениям базы данных Oracle. Java применяется во множестве случаев, помимо приложений Oracle, но многие свойства этого языка не используются большинством разработчиков Oracle. Цель данной главы состоит в том, чтобы предоставить разработчикам, которые знакомы с SQL и PL/SQL, базовые знания о структурах языка Java. Эта глава не дает исчерпывающего обзора Java, а является скорее кратким обзором компонентов языка Java, наиболее часто используемых разработчиками Oracle.

Хотя во многих случаях PL/SQL и Java хорошо соотносятся друг с другом, существуют значительные различия в терминологии и использовании, поскольку эти два языка программирования разрабатывались независимо друг от друга. Более того, объектно-ориентированные возможности PL/SQL не входили в его исходную реализацию, в то время как Java с самого начала был объектно-ориентированным языком программирования. Чтобы эффективно использовать Java, необходимо применить подход, отличный от того, который использовался в случае PL/SQL.

Глава 32 представляет собой введение в структуры языка PL/SQL и управление потоком выполнения. Данная глава повторяет этот подход: здесь показаны структуры, которые использует Java, и существующие базовые методы управления потоком выполнения. Где возможно, показаны также соответствующие структуры PL/SQL. Прежде чем читать эту главу, необходимо познакомиться с PL/SQL (глава 32), пакетами, функциями и процедурами (глава 38), пользовательскими типами и методами (глава 38).

Сравнение Java и PL/SQL: Общее представление

При сравнении PL/SQL и Java значительные различия в терминологии между двумя языками проявляются уже на уровне базовой структуры PL/SQL, т.е. блока. В PL/SQL блок является структурированной частью кода, который имеет раздел объявлений, раздел исполняемых команд и раздел обработки исключительных ситуаций. Структуру блока PL/SQL можно представить следующим образом:

- ```
  declare
  <раздел объявлений>

  begin
  <исполняемые команды>
  exception
  <обработка исключительных ситуаций>
  end;
  ```

В Java термин блок относится к значительно меньшему подмножеству кода. Блок является совокупностью операторов, заключенных в фигурные скобки. Например, следующий псевдокод содержит два блока Java:

- ```
  if (некоторое условие) {
   блок кода для выполнения, если условие удовлетворяется
  }
  else {
   блок кода для выполнения, если условие не удовлетворяется
  }
  ```

В PL/SQL весь раздел кода будет просто частью большего блока, в Java он содержит два отдельных блока. В этой главе все ссылки на «блоки» имеют в виду блоки Java, если явно не определено иначе.

Процедура PL/SQL ограничена операторными скобками begin и end, а метод Java ограничивается открывающей и закрывающей фигурными скобками. Эти же скобки используются для отделения блоков рассматриваемого кода или как индикатор открытия и закрытия условных сегментов.

Вторым существенным различием между PL/SQL и Java является объявление переменных. В блоке PL/SQL переменные определяются до начала раздела исполняемых команд. В программе Java можно определять переменные, когда они понадобятся в программе. В программе Java нет раздела объявлений, аналогичного тому, который используется в блоках PL/SQL.

В данной главе будут показаны и другие различия между PL/SQL и Java. Важно помнить, что Java не заменяет PL/SQL в приложениях - вы можете продолжать использовать PL/SQL для приложений Oracle. Необходимо протестировать производительность PL/SQL и Java, прежде чем принять решение, какого направления придерживаться для операций извлечения данных.

Первое знакомство

Чтобы воспользоваться примерами из этого раздела, необходимо иметь копию пакета разработки Java (Java Development Kit, JDK), который доступен для бесплатной загрузки на сайте http://java.sun.com. Необходимо установить JDK на сервер, на котором будут выполняться программы Java.

Примечание JDK инсталлируется как часть стандартной инсталляции Oracle Database. Убедитесь, что у вас инсталлировано как программное обеспечение ядра базы данных, так и связанное с ним клиентское программное обеспечение.

Объявления

В Java переменные можно объявлять по мере необходимости. Переменная имеет имя и тип данных и используется для хранения значения данных во время выполнения программы. Переменная имеет также область действия, позволяющую ей быть общедоступной или частной – аналогично тому, как процедуры в пакете могут иметь частные переменные.

Вот примеры объявления переменных в Java:

```
char aCharVariable = 'J';
boolean aBooleanVariable = false;
```

По соглашению, имена переменных Java всегда начинаются с буквы нижнего регистра, как показывает этот пример. Здесь используются типы данных CHAR и BOOLEAN и каждой переменной присваивается начальное значение. Знаком присваивания в Java является =, в противоположность := или => в PL/SQL. Каждое объявление заканчивается точкой с запятой.

Имеющиеся примитивные типы данных Java перечислены в таблице 42.1.

Таблица 42.1. *Примитивные типы данных Java*

Тип данных	Описание
Byte	Целое длиной в один байт
Short	Короткое целое
Int	Целое
Long	Длинное целое
Float	Действительное число с плавающей точкой одинарной точности
Double	Действительное число с плавающей точкой двойной точности
Char	Одиночный символ Юникода
Boolean	Булево значение – true или false

Кроме примитивных типов данных, можно использовать ссылочные типы данных, которые основываются на содержимом переменных. Для строк символов переменной длины не существует примитивного типа данных - Java предоставляет для этого класс String.

Исполняемые команды

Переменным можно присваивать значения с помощью выражений и операторов. Java поддерживает следующие арифметические операторы:

Оператор	Описание
*	Умножение
/	Деление
+	Сложение
-	Вычитание
%	Сравнение по модулю

Кроме этих операторов, для упрощения кодирования можно использовать унарные операторы Java. Например, можно увеличивать переменные с помощью оператора ++:

```
aLoopCounter = aLoopCounter++;
```

или можно просто написать:

```
aLoopCounter++
```

Примечание Оператор ++ называется унарным, потому что он имеет только один операнд. Если оператору требуется два операнда (как в случае *), он называется бинарным.

В PL/SQL это можно было записать как:

```
aLoopCounter := aLoopCounter + 1;
```

Если оператор ++ помещается перед именем переменной, то он является оператором прединкремента (предварительного увеличения), а не оператором постинкремента (увеличения после):

```
int anotherCounter = ++LoopCounter;
```

В этом примере переменной anotherCounter присваивается увеличенное значение aLoopCounter. В противоположность этому

```
int anotherCounter = aLoopCounter++;
```

присваивает anotherCounter значение aLoopCounter до увеличения. Значение переменной можно уменьшать с помощью унарного оператора –:

```
aLoopCounter = aLoopCounter--;
```

Можно также комбинировать присваивание значений с выполняемой операцией. Например, вместо записи

```
aNumberVariable = aNumberVariable * 2;
```

можно написать:

```
aNumberVariable *= 2;
```

Можно выполнять аналогичные комбинации, используя /=, %=, +=, и -=.

При выполнении нескольких арифметических операций можно использовать скобки для четкого указания порядка, в котором операции должны выполняться:

```
aNumberVariable = (aNumberVariable * 2) + 2;
```

Завершение этих операций точкой с запятой делает их *операторами* - законченными единицами выполнения.

Условные операторы

Переменные можно оценивать с помощью выражений и операторов. Для этого используются один или несколько фундаментальных классов, которые являются частью Java. Перечисление всех этих методов выходит за рамки нашей книги. Их можно найти на сайте компании Sun с документацией Java или в одной из множества книг по Java, которые сейчас доступны.

Примечание Число предоставляемых классов существенно увеличивается с каждой новой версией Java.

В следующем примере метод Character.isUpperCase используется для вычисления объявленной ранее переменной aCharVariable:

```
if(Character.isUpperCase(aCharVariable)) {
 команды для выполнения
}
```

Если значение aCharVariable представлено символом верхнего регистра, будут выполнены команды блока, иначе управление будет передано следующей части программы.

Далее следует общий синтаксис фраз if:

```
if (выражение) {
оператор
}
```

Отметим отсутствие фразы then. Если оцениваемое выражение является true, следующий блок выполняется автоматически. Также отсутствует фраза end if. Блоки Java просто завершаются закрывающей скобкой оператора.

Могут присутствовать фразы else для оценивания других условий:

```
if (выражение) {
оператор
}
else {
оператор
}
```

Для проверки нескольких условий можно использовать фразу else if (для оценивания каждого из них по очереди) и закончить все фразой else. Следующий листинг иллюстрирует использование фразы else if:

```
if (aCharVariable == 'V') {
оператор
}
else if (aCharVariable == 'J') {
оператор
}
else {
оператор
}
```

Примечание PL/SQL поддерживает использование для выбора между несколькими альтернативами фраз elsif. В одной фразе if может быть несколько фраз elsif.

Кроме фраз if для условных операторов, Java использует фразу switch. Вместе с фразой break и метками операторов фраза switch может имитировать функциональность фразы goto (которые Java не поддерживает). Рассмотрим сначала простой пример switch. С помощью оператора case оценивается несколько значений переменной aMonth. В зависимости от значения aMonth выводится текстовое название месяца, и обработка покидает блок switch с помощью break.

```
int aMonth=2;
switch (aMonth) {
case 1:
System.out.println(“January”);
break;
case 2:
System.out.println(“February”);
break;
case 3:
System.out.println(“March”);
break;
default:
System.out.println(“Out of range”);
break;
}
```

Примечание В Java аргументами оператора case должны быть целые числа.

Код такого вида значительно сложнее, чем простой TO_CHAR, но он иллюстрирует использование switch. Оператор switch получает на входе переменную aMonth и оценивает ее значение. Если значение соответствует значению case, вызывается метод System.out.println для вывода названия месяца, и управление обработкой покидает блок switch с помощью break. Если значение не соответствует ни одной из фраз case, выполняется вариант default.

Куда передается управление? По умолчанию оно переходит к следующему разделу программы. Однако можно создать метки для разделов кода и передать имя такой метки во фразу break. Следующий листинг показывает пример метки и фразы break:

```
somelabel:
if (aCharVariable == ‘V’) {
оператор
}
else {
оператор
}
int aMonth=2;
switch (aMonth) {
case 1:
System.out.println(“January”);
break somelabel;
case 2:
System.out.println(“February”);
break someotherlabel;
case 3:
System.out.println(“March”);
break somethirdlabel;
default:
System.out.println(“Out of range”);
break;
}
```

В этом примере оценивается фраза if, за которой следует фраза switch. Переменная aMonth не объявляется до тех пор, пока она не потребуется. Здесь переменной aMonth присвоено значение 2, поэтому программа будет выводить слово «February», а затем перейдет в секцию кода, идентифицированную меткой someotherlabel. Если бы значение aMonth было равно 1, была бы повторно выполнена фраза if в начале кода.

Java поддерживает также тернарный оператор - оператор, который получает три операнда и функция которого аналогична функции DECODE. Действуя как встроенная комбинация if-else, тернарный оператор оценивает выражение. Если выражение оценивается как true, возвращается второй операнд, иначе возвращается третий. Синтаксис имеет следующий вид:

- `выражение ? операнд1 : операнд2`

Следующий листинг показывает пример использования тернарной операции:

- `aStatusVariable = (aCharVariable == 'V') ? «OK» : «Invalid»;`

В этом примере оценивается выражение

- `(aCharVariable == 'V')`

Если оно справедливо, возвращается «OK», иначе возвращается «Invalid».

Тернарный оператор можно использовать для упрощения применения функции GREATEST:

- `double greatest = (a > b) ? a : b;`

В этом примере оценивается выражение (a > b), где a и b являются именами переменных. Если это выражение будет true, возвращается значение a, иначе возвращается значение b.

Примечание В Java имеется метод "greatest" (наибольший) для цифровых значений - метод Max в классе java.lang.Math. Пример с тернарным оператором можно переписать следующим образом: double greatest = Math.max(a, b);

Можно объединить несколько логических проверок с помощью операций AND и OR. В Java операция AND представляется с помощью оператора &&:

- ```
  if (aCharVariable == 'V' && aMonth == 3) {
  оператор
  }
  ```

Оператор OR представляется с помощью ||:

- ```
  if (aCharVariable == 'V' || aMonth == 3) {
  оператор
  }
  ```

По соображениям производительности операции && и || выполняются, только когда в этом возникает необходимость; второе выражение не оценивается, если результат определяется первым выражением. В случае оператора AND, если первое значение будет false, то оценивание заканчи-

вается в этом месте и возвращается булево значение false. В случае оператора OR, если первое выражение будет false, то второе выражение все равно будет оцениваться, так как только одно из проверяемых условий должно быть true, чтобы все выражение было оценено как true. Если первое значение оператора OR будет true, последующие выражения оценивать не потребуется, и будет возвращено true.

Операторы & и | являются побитовыми AND и OR, и они не сокращаются, как операторы && и ||. Все выражения будут всегда оцениваться. Эта функциональность может быть важной, если проверяются возвращаемые значения вызовов методов и необходимо убедиться, что методы выполнились.

Циклы

Java поддерживает три основных типа циклов: WHILE, DO-WHILE и FOR. Так как Java не является расширением SQL, то в отличие от PL/SQL курсорные циклы FOR не поддерживаются.

Циклы WHILE и DO-WHILE

Фраза while оценивает условие: если значением условия будет true, выполняется соответствующий блок операторов. Синтаксис цикла WHILE следующий:

```
while (выражение) {
оператор
оператор
}
```

Цикл WHILE повторно выполняет блок операторов (секция между скобками { и }), пока выражение не перестанет быть true:

```
int aNumberVariable = 3;
while (aNumberVariable < 7) {
```

некоторые операторы для обработки

```
aNumberVariable++;
}
```

В этом примере создается и инициализируется переменная счетчик (aNumberVariable). Затем значение переменной вычисляется во фразе while. Если это значение меньше 7, выполняется соответствующий блок операторов. Как часть этого блока переменная получает приращение. Когда блок завершается, переменная снова оценивается, и цикл продолжается.

Примечание Примеры обработки цикла WHILE приведены далее в разделе “Классы”.

Можно также записать это как цикл DO-WHILE:

```
int aNumberVariable = 3;
do {
```

некоторые операторы для обработки

```
aNumberVariable++;
} while (aNumberVariable < 7);
```

В цикле DO-WHILE значение выражения не оценивается, пока блок не будет обработан хотя бы один раз.

Циклы FOR

Цикл FOR можно использовать для повторного выполнения блока кода. В цикле FOR определяется начальное значение, конечное значение и шаг счетчика цикла. Счетчик цикла будет увеличиваться на заданное значение шага, пока не будет достигнуто конечное значение. Ниже представлен синтаксис цикла FOR:

```
for (начальное значение; условие окончания; шаг цикла) {
оператор;
}
```

Пример с циклом WHILE из предыдущего раздела можно переписать с помощью фразы for:

```
for (int aNumberVariable=3; aNumberVariable < 7; aNumberVariable++) {
оператор для обработки
}
```

Версия с фразой for для этого примера значительно короче, чем версия с while. В цикле FOR объявляется и инициализируется переменная aNumberVariable. Пока значение переменной не превышает 7, блок будет выполняться. При каждом проходе цикла значение переменной увеличивается на 1 с помощью унарного оператора ++.

Внутри цикла можно использовать фразу continue для перехода к другому оператору цикла. Если использовать просто фразу continue, процесс выполнения перейдет в конец тела цикла и выполнит проверку окончания цикла. Если continue используется с меткой (как было показано в разделе о фразе switch), обработка продолжится со следующей итерации помеченного цикла.

Курсоры и циклы

В PL/SQL можно использовать курсорные циклы FOR для итераций по результатам запроса. Циклы WHILE и FOR из Java можно использовать для моделирования функциональности курсорного цикла FOR из PL/SQL.

Oracle предоставляет образцы файлов, которые сопровождают установку программного обеспечения Oracle, с большим количеством примеров, иллюстрирующих стандарты подходов к кодированию. Установка клиента использует Oracle Universal Installer для загрузки драйверов и демонстрации файлов, в том числе те, которые поддерживают развитие JDBC. После установки клиентского программного обеспечения, примеры JDBC становятся Java-связанными подкаталогами домашнего каталога программного обеспечения Oracle. Точное количество предоставляемых файлов может меняться от версии к версии.

Примечание Файл README в подкаталоге /JDBC домашнего каталога программного обеспечения Oracle предоставляет обширную информацию о поддерживаемой версии, настройке среды переменной платформы и поддерживаемых функций. Очень важно читать и следовать этим инструкциям, чтобы эффективно использовать Java от Oracle на вашей платформе.

В следующем примере показан фрагмент кода из примера, приведенного из установки клиента Oracle:

```
Statement stmt conn.createStatement();
ResultSet rset = stmt.executeQuery ("select FIRST_NAME, "
+ "LAST_NAME from EMPLOYEES");
// Итерации по множеству результатов и печать имен сотрудников
while (rset.next())
 System.out.println (rset.getString (1) + " " + rset.getString(2));
```

В этом примере предполагается, что соединение с базой данных уже было установлено. В первой строке создается переменная stmt на основе переменной соединения с именем conn (не показанной в этом частичном примере). Во второй строке создается переменная rset на основе множества результатов, которое возвращает запрос stmt. Вслед за комментарием (который начинается с //) следует фраза while, извлекающая из множества результатов каждую строку. Если операция выборки (fetch) извлекает строку из множества результатов, печатается ENAME из этой строки EMP, иначе цикл заканчивается.

С точки зрения программирования следует быть предельно аккуратным с возвращаемыми данными и указывать имена столбцов, которые должны быть возвращены.

Примеры использования JDBC (Java Database Connectivity – средство организации доступа Java-приложений к базам данных в сети; см. главу 43), предоставленные Oracle, предлагают образцы запросов, включающие вызовы процедур, LOB и DML. В большинстве случаев код Java будет выполняться аналогично предыдущему примеру: напишите процесс управления потоком выполнения на Java и передайте ему оператор для выполнения. На базе результата этого оператора можно выполнять различные действия, используя фразы while, for, if, break, continue и switch.

Обработка исключительных ситуаций

В PL/SQL имеется единый блок обработки исключительных ситуаций (EXCEPTION); в Java вы можете иметь столько блоков try/catch, сколько потребуется. Java предоставляет богатый набор средств обработки ошибок и позволяет создавать для этого сложные процедуры. В PL/SQL исключительные ситуации возбуждаются (raise), в Java исключительные ситуации порождаются (throw). Если порождена исключительная ситуация, для ее перехвата и соответствующей обработки необходимо использовать оператор Java catch.

Синтаксис обработки ошибок в Java основывается на трех блоках: try, catch и finally. Блок try включает операторы, которые могут порождать исключительную ситуацию. Блок catch, следующий сразу за блоком try, связывает обработчики событий с исключительными ситуациями, которые могут порождаться блоком try. Блок finally очищает все системные ресурсы, которые не были должным образом очищены во время обработки блока catch. Общая структура имеет следующий вид:

```
try {
 операторы
}
```

```
catch ( ) {
 операторы
}
catch ( ) {
 операторы
}
finally {
 операторы
}
```

Рассмотрим, например, следующий код Java:

```
try {
Statement stmt = conn.createStatement ();
stmt.execute ("drop table plsqltest");
}
catch (SQLException e) {
}
```

В этом примере создается переменная stmt для выполнения команды drop table. Если выполнение завершается аварийно (например, таблица не существует), как будет обрабатываться эта ошибка? Оператор catch приказывает Java обработать его с помощью объекта исключительной ситуации SQLException - стандартной части реализации SQL в Java. Фраза catch имеет два параметра (объект исключения и имя переменной), и за ней может при необходимости следовать блок операторов для выполнения.

Блок try может содержать несколько операторов, или можно создать отдельные блоки try для каждого оператора. Как правило, легче управлять обработкой ошибок, если объединить блоки catch, поэтому объединение блоков try поможет управлять кодом при его изменении и увеличении в размерах. Оператор when others в PL/SQL – это, по сути, то же самое, что и catch(Throwable th) в Java.

Блок finally очищает состояние текущего раздела кода, прежде чем передать управление любым последующим частям программы. Во время выполнения содержимое блока finally всегда выполняется независимо от результата блока try. Например, можно использовать блок finally для закрытия соединения с базой данных.

Зарезервированные слова

Зарезервированные слова Java перечислены в таблице 42.2. Эти слова нельзя использовать в качестве имен классов, методов или переменных.

Таблица 42.2. *Зарезервированные слова Java*

abstract	else	interface	super
boolean	extends	long	switch
break	false	native	synchronized
byte	final	new	this
case	finally	null	throw
catch	float	package	throws

Таблица 42.2. *Зарезервированные слова Java (продолжение)*

char	for	private	transient
class	goto	protected	true
const	if	public	try
continue	implements	return	void
default	import	short	volatile
do	instanceof	static	while
double	int	strictfp	

Классы

В предыдущих разделах были показаны базовые синтаксические структуры Java. В данном разделе показано, как этот синтаксис применяется для создания и использования объектов. В следующем примере представлена простая программа для печати слова «Oracle»:

```
public class HelloOracle {
public static void main (String[] args) {
 System.out.println("Oracle");
}
}
```

Этот пример создает класс с именем HelloOracle, в котором создается открытый метод с именем main. Метод main печатает слово «Oracle». Это значительно сложнее, чем

```
select 'Oracle' from dual;
```

Однако версия Java имеет свойства, которые отсутствуют в версии SQL. HelloOracle является классом, и поэтому его методы могут вызываться из других классов. HelloOracle может иметь несколько методов. В этом примере он имеет только метод main.

В объявлении метода main (public static void main) имеется ряд ключевых слов, которые уже нам знакомы:

public	Определение класса как public (общедоступный) позволяет всем классам вызывать этот метод.
static	Объявляет, что это метод класса; используется для объявления методов класса. Дополнительной возможностью является ключевое слово final для методов и переменных, значения которых не могут изменяться (аналогично параметру constant в PL/SQL).
void	Определяет тип данных возвращаемого значения процедуры. Так как эта процедура не имеет возвращаемых значений, ее тип будет void.

В нашем примере методу дано имя main, так как это упрощает выполнение метода класса. Теперь, когда класс создан, его можно откомпилировать и загрузить.

Примечание Для того чтобы откомпилировать и выполнить класс Java, необходимо иметь на своем сервере JDK. В следующих примерах предполагается, что двоичные файлы, являющиеся частью этого пакета, расположены в каталоге, который автоматически доступен через переменную окружения PATH, а каталог, содержащий файлы с расширением .class, включен в переменную окружения CLASSPATH *(смотрите файл /jdc/README для более подробной настройки переменной среды)*. Используемые здесь программы (javac и java) расположены в подкаталоге /bin каталога программного обеспечения JDK.

Сначала сохраним программу как обычный текстовый файл с именем HelloOracle.java. Затем откомпилируем его:

```
javac HelloOracle.java
```

На сервере будет создан новый файл с именем HelloOracle.class. Затем можно выполнить этот класс:

```
java HelloOracle
```

Будет выведен результат:

```
Oracle
```

Как показано в примере HelloOracle, для класса сначала задается имя вместе с другими необязательными атрибутами, связанными с его наследованием атрибутов других классов. В теле класса определяются все переменные и методы класса. Как и у абстрактных типов данных в Oracle, у классов Java имеются связанные с ними методы манипулирования данными.

Рассмотрим более сложный пример. В следующем листинге приведена программа Java, которая вычисляет площадь круга, когда в качестве входного параметра задается значение радиуса (версию PL/SQL см. в главе 32).

```
// AreaOfCircle.java (Площадь круга.java)
//
public class AreaOfCircle {
public static void main(String[] args) {
try {
int input = Integer.parseInt(args[0]);
double result=area(input);
System.out.println(result);
}
catch (ArrayIndexOutOfBoundsException oob) {
System.out.println("You did not provide the radius of the circle.");
System.out.println("Usage: java AreaOfCircle 10");
}
catch (NumberFormatException nfe) {
System.out.println("Enter a valid number for the radius.");
System.out.println("Usage: java AreaOfCircle 10");
}
} /// main
public static double area (int rad) {
```

```
double pi=3.1415927;
double areacircle=pi*rad*rad;
return areacircle;
} /// area
} /// AreaOfCircle
```

Примечание Не забудьте включить для каждого блока закрывающую скобку }. В показанных в этой главе примерах закрывающая скобка всегда находится на отдельной строке, чтобы упростить отладку и обработку ошибок управления потоком выполнения.

Прежде всего отметим, что имя класса и имя файла должны быть одинаковыми. В нашем случае класс имеет имя AreaOfCircle, поэтому этот текст сохраняется в имени файла: AreaOfCircle.java. Класс имеет два метода: main и area. При выполнении этого класса метод main выполняется автоматически. Метод main получает в качестве ввода строку и преобразует ее в целое число, которое помещается в переменную input:

- `int input=Integer.parseInt(args[0]);`

Существуют по крайней мере две возможные исключительные ситуации, которые могут порождаться при выполнении этого кода. Первая - ArrayIndexOutOfBoundsException - возникает, когда делается попытка обратиться к несуществующей позиции массива. Так как код явно ссылается на первую позицию массива с именем args (args[0]), то исключительная ситуация будет порождаться в том случае, если программе не были переданы никакие данные. Другая возможная ошибка - NumberFormatException - возникает в том случае, когда в качестве аргумента передается нечисловое значение, например, если вместо “java AreaOfCircle 123” ввести “java AreaOfCircle XYZ” (XYZ не является допустимым числом). Обе эти исключительные ситуации обрабатываются двумя операторами catch, которые предоставляют пользователю полезную обратную связь в отношении причины ошибки:

- ```
  catch (ArrayIndexOutOfBoundsException oob) {
  System.out.println(“You did not provide the radius of the circle.”);
  System.out.println(“Usage: java AreaOfCircle 10”);
  }
  catch (NumberFormatException nfe) {
  System.out.println(“Enter a valid number for the radius.”);
  System.out.println(“Usage: java AreaOfCircle 10”);
  }
  ```

Затем объявляется переменная result, которая задается равной значению, возвращаемому вызовом area(input):

- `double result=area(input);`

Чтобы обработать эту директиву, выполняется метод area, использующий в качестве входного значения значение переменной input. Метод имеет следующий вид:

- ```
  public static double area (int rad) {
  double pi=3.1415927;
  double areacircle=pi*rad*rad;
  ```

```
return areacircle;
}
```

В своем определении метод area указывает, что он получает одну переменную целочисленного типа с именем rad. Затем значение этой переменной (передаваемой из переменной input метода main) обрабатывается и вычисляется переменная с именем areacircle, которая возвращается в main. В main это значение печатается:

```
double result=area(input);
System.out.println(result);
```

Тестирование программы очевидно и легко выполнимо:

```
javac AreaOfCircle.java
java AreaOfCircle 10
```

Будет получен следующий результат:

```
314.15927
```

Давайте расширим этот пример, чтобы включить в него обработку цикла WHILE. В этом примере метод area будет вызываться повторно, пока получающееся входное значение не превысит заданное значение:

```
// AreaOfCircleWhile.java
//
public class AreaOfCircleWhile {
public static void main(String[] args) {
try {
int input = Integer.parseInt(args[0]);
while (input < 7) {
double result=area(input);
System.out.println(result);
input++;
}
}
catch (ArrayIndexOutOfBoundsException oob) {
System.out.println("You did not provide the radius of the circle.");
System.out.println("Usage: java AreaOfCircle 10");
}
catch (NumberFormatException nfe) {
System.out.println("Enter a valid number for the radius.");
System.out println("Usage: java AreaOfCircle 10");
}
} /// main
public static double area (int rad) {
double pi=3.1415927;
double areacircle=pi*rad*rad;
return areacircle;
} /// area
} /// AreaOfCircleWhile
```

В этом листинге метод area и блоки обработки исключительных ситуаций не изменились по сравнению с предыдущим примером. Изменение содержится в методе main:

- ```
  int input = Integer.parseInt(args[0]);
  while (input < 7) {
  double result=area(input);
  System.out.println(result);
  input++;
  }
  ```

Пока входное значение меньше 7, оно будет обрабатываться. После того как для этого значения радиуса вычисляется и печатается площадь, входное значение изменяется:

- ```
  input++;
  ```

и цикл оценивается заново. Пример вывода показан в следующем листинге:

- ```
  javac AreaOfCircleWhile.java
  java AreaOfCircleWhile 4

  50.2654832
  78.5398175
  113.0973372
  ```

Вывод показывает вычисленную площадь для входных значений радиуса, равных 4, 5 и 6.

Этот пример не является полностью завершенной программой – может понадобиться дополнительная обработка исключений (например, для нецелочисленных значений), но его структура является важным уроком. Кроме методов, показанных в этом примере, можно создать конструктор для класса AreaOfCircleWhile. Все классы Java имеют конструкторы для инициализации новых объектов на основе класса. Имя конструктора совпадает с именем класса. Если создается класс без конструктора, во время выполнения Java создаст конструктор по умолчанию. Подобно методу, тело конструктора может содержать объявления локальных переменных, циклы и блоки операторов. Конструктор инициализирует эти переменные для класса.

В следующих главах вы увидите, как реализовать Java в Oracle – в хранимых процедурах и через JDBC. Эти главы предполагают наличие базовых знаний Java. Дополнительную информацию о языке Java можно найти на Web-сайте корпорации Sun Microsystems и во множестве книг.
  ```