Определение классов и методов

Java- программа состоит из объектов различных классов, взаимодействующих друг с другом.

Каждое определение Java-класса должно быть в отдельном файле, например MyClass.java и может быть откомпилировано отдельно- MyClass.class

Затем можно скомпилировать файл, содержащий main-метод, без перекомпиляции класса MyClass.java

Определение классов и методов

Класс—это АТД для создания объекта.

Класс определяет структуру объекта и его методы, образующие функциональный интерфейс.

В процессе выполнения Java-программы система использует определения классов для создания представителей классов.

Представители являются реальными объектами. Термины <представитель>, <экземпляр> и <объект> взаимозаменяемы.

Определение классов и методов

Общая форма определения класса.

```
class <имя_класса> extends <имя_суперкласса> {
  type переменная1_объекта;
  type переменная2_объекта;
  type переменнаяN_объекта;
  type имяметода1(список_параметров)
        {
            тело метода;
         }
  type имяметода2(список_параметров)
        {
            тело метода;
         }
}
```

Определение классов и методов

Данные инкапсулируются в класс путем объявления переменных между открывающей и закрывающей фигурными скобками, выделяющими в определении класса его тело.

Называются переменными(полями) реализации или переменными экземпляра класса.

Единственное отличие состоит в том, что их надо объявлять вне методов, в том числе вне метода **main**.

Определение классов и методов

```
class Point {
        int x, y;//поля класса
    }
```

Оператор **new** создает экземпляр указанного класса и возвращает ссылку на вновь созданный объект.

Point p = new Point();

можно создать несколько ссылок на один и тот же объект.

Определение классов и методов

```
class TwoPoints {
  public static void main(String args[]) {
  Point p1 = new Point();
  Point p2 = new Point();
  p1.x = 10; p1.y = 20;
  p2.x = 42; p2.y = 99;
…
            }
        }
```

Определение классов и методов

В Java используется также понятие абстрактный класс.

С их помощью можно объявлять классы, реализованные лишь частично, полная реализация осуществляется в потомках – расширениях класса

Используется, когда некоторое поведение (методы) характерно для большинства или всех объектов данного класса, но некоторые аспекты имеют смысл лишь для ограниченного круга объектов, не составляющих суперкласса, те методы конкретизируются в подклассах.

Определение классов и методов

```
 // Абстрактный класс “Фигура”
abstract public class Shape {
 // Цвет фигуры
 int Color;// поле класса
 // Начальная точка фигуры
 Coordinates StartPoint; // поле класса
// Нарисовать фигуру
abstract public void Draw();
 }
```

Определение классов и методов

```
 // Конкретный класс "Круг"
class Circle extends Shape {
    //Нарисовать круг
public void Draw() {
// Здесь реализуется метод рисования
 //круга
}
}
```

Определение классов и методов

```
abstract class Square {
  abstract int squareIt(int i); //абстрактный метод
  public void show() {
  System.out.println ("обычный метод");
  }
}
//squareIt() должен быть реализован подклассом
  //Square
class SquareReal extends Square {
  public int squareIt (int i) {
return i*i;
}
}
```

Определение классов и методов

```
public class AbstractDemo {
  public static void main(String[] args) {
// Square ob1 = new Square(); //ашипка!
    Square ob2 = new SquareReal();
    System.out.println("10 в квадрате
  равно " + ob2.squareIt(10));
    ob2.show();
  }
}
```

Определение классов и методов

если унаследовать класс от абстрактного, но оставить нереализованным хотя бы один его абстрактный метод,

то унаследованный класс также будет абстрактным.

Чтобы избавиться от "абстрактности", необходимо реализовать код для всех абстрактных методов абстрактного класса-предка.

Определение классов и методов

Java предоставляет программисту еще одно средство, родственное классам, - интерфейсы.

Интерфейс - это набор констант и абстрактных методов, которые не содержат никакого кода.

Каждый класс реализующий интерфейс, должен реализовать все его методы.

Если только часть, то класс объявляется абстрактным

Определение классов и методов

Интерфейсы дают возможность программисту описывать наборы методов, которые должен реализовать класс.

```
public interface CustomLook {
   public abstract void notifyStartPaint();
   public abstract void customPaint ();
}
```

Определение классов и методов

```
public class NewButton implements
CustomLook {
public void notifyStartPaint() {
// Код для перехвата начала рисования }
public void customPaint (); }
{ // Код для рисования кнопки нового //
внешнего вида }
}
```

Определение классов и методов

```
interface Voice {
    void voice();
}
class Dog implements Voice {
   public void voice () {
      System.out.println("Gav-gav!");
   }
}
```

Определение классов и методов

```
class Cat implements Voice {
   public void voice () {
      System.out.println("Miaou!");
   }
}
class Cow implements Voice {
   public void voice() {
      System.out.println("Moo-Oo-oo!");
   }
}
```

Определение классов и методов

- Интерфейсы предоставляют некоторую разновидность множественного наследования, те класс может реализовать несколько интерфейсов.
- Абстрактный класс может содержать частичную реализацию, защищенные компоненты, статические методы и тд, интерфейс ограничивается открытыми методами и константами
- Интерфейс является выражением чистой концепции проектирования, а класс смесь проектирования и конкретной реализации

test

Определение классов и методов

- Модификаторы доступа

Во многих языках существуют права доступа, которые ограничивают возможность использования, например, переменной в классе.

Два крайних вида прав доступа: это public, когда поле доступно из любой точки программы, и private, когда поле может использоваться только внутри того класса, в котором оно объявлено.

Определение классов и методов

Уровень доступа элемента языка является статическим свойством, задается на уровне кода и всегда проверяется во время компиляции.

Попытка обратиться к закрытому элементу вызовет ошибку.

Определение классов и методов

В Java модификаторы доступа указываются для:

1. типов (классов и интерфейсов) объявления верхнего уровня;
2. элементов ссылочных типов (полей, методов, внутренних типов);
3. конструкторов классов.

Как следствие например, массив также может быть недоступен в том случае, если недоступен тип, на основе которого он объявлен.

Определение классов и методов

Четыре уровня доступа

1. Public
2. Private
3. Protected
4. если не указан ни один из этих трех типов, то уровень доступа определяется по умолчанию (default) иногда его называют пакетным.

Определение классов и методов

1. Открытый (Public) к членам класса всегда можно обращаться из любого места, в котором доступен сам класс; такие члены наследуются в подклассах
2. Закрытый (Private): доступ к членам класса осуществляется только из самого класса

Определение классов и методов

Если попытаться обратиться к private-данным или методам, то компилятор Java выдаст сообщение об ошибке компиляции.

Если ваш класс не будет в дальнейшем наследоваться, то лучше использовать модификатор private, а не protected

Определение классов и методов

3. Защищенный (Protected): к данным членам разрешается доступ из подклассов и из классов(методов), входящих в тот же пакет,

те наследникам может потребоваться доступ к некоторым элементам родителя, с которыми не приходится иметь дело внешним классам.

Определение классов и методов

Модификатор доступа protected позволяет обращаться к данным и методам класса самому классу, классам, хранящимся в этом же пакете, и унаследованным подклассам.

Определение классов и методов

Обычно такой модификатор применяют для того, чтобы закрыть доступ к данным и методам для тех классов, которые не состоят в "родственных отношениях" с защищаемым классом.

В Java классы считаются родственными, не только если они унаследованы друг от друга, но и просто хранятся в одном и том же пакете.

Определение классов и методов

```
package My;
class First{
 protected int protVar;
 protected void protMethod() {
 System.out.println("protMeth called!");
  }
}
```

Определение классов и методов

```
package My;
class Second {// не наследник First
void protAccessMethod() {
First ap = new First();
ap.protVar = 345;
ap.protMethod();
}
}
```

Определение классов и методов

4. Пакетный:(package access) доступ к членам, объявленным без указания атрибута доступа, осуществляется только из того же пакета, где объявлен и сам этот класс.

Более ограниченный по сравнению с protected

Определение классов и методов

Пакет в Java - это коллекция сгруппированных вместе классов, которой присвоено некоторое имя.

Все классы пакета размещаются в отдельных файлах, причем имя каждого файла совпадает с именем содержащегося в нем класса.

Единственная новая деталь, отличающая обычный файл *.java от пакетного, состоит в том , что первая строка каждого файла пакета должна иметь следующий вид:

Определение классов и методов

package *Имя_пакета*;

например,
package mystuff.utilities;

Определение классов и методов

Классы пакета хранятся в некотором каталоге (папке), имя которого совпадает с именем пакета, пакету присваивается имя, которое можно использовать затем в программах или классах.

Любое приложение или определение класса может использовать все классы пакета, поместив соответствующий оператор import в начало файла, содержащего это приложение или это определение класса:

```
import mystuff.utilities;// ..\mystuff\utilities
```

Определение классов и методов

Модификатор protected может быть указан для наследника из другого пакета, а доступ по умолчанию допускает обращения из классов-ненаследников, если они находятся в том же пакете.

Определение классов и методов

модификаторы доступа упорядочиваются следующим образом (от менее открытых – к более открытым):

1. private
2. none (package)
3. protected
4. public

Определение классов и методов

- Пакеты доступны всегда, поэтому у них нет модификаторов доступа (можно сказать, что все они public, то есть любой существующий в системе пакет может использоваться из любой точки программы).

Определение классов и методов

- Типы (классы и интерфейсы) верхнего уровня объявления.

При их объявлении существует всего две возможности: указать модификатор public или не указывать его. Если доступ к типу является public, то это означает, что он доступен из любой точки кода.

Если же он не public, то уровень доступа назначается по умолчанию: тип доступен только внутри того пакета, где он объявлен.

Определение классов и методов

- Массив имеет тот же уровень доступа, что и тип, на основе которого он объявлен (все примитивные типы являются полностью доступными).
- Элементы и конструкторы объектных типов. Обладают всеми четырьмя возможными значениями уровня доступа. Все элементы интерфейсов являются public.

table

Определение классов и методов

Методы—это подпрограммы, присоединенные к конкретным определениям (описаниям) классов.

Они описываются внутри определения класса на том же уровне, что и переменные объектов.

При объявлении метода задаются тип возвращаемого им результата и список параметров.

Определение классов и методов

Общая форма объявления метода:

тип <имя_метода> (<список формальных параметров>)

{

тело метода

}

Тип результата, который должен возвращать метод может быть любым, в том числе и типом void—в тех случаях, когда возвращать результат не требуется.

Определение классов и методов

```
class Point {
  int x, y;
     void init(int a, int b)
          {
            x = a;
            y = b;
          }
       }
```

Определение классов и методов

Заголовок состоит из:

1. модификаторов (доступа в том числе);
2. типа возвращаемого значения или ключевого слова void;
3. имени метода;
4. списка аргументов в круглых скобках (аргументов может не быть);
5. специального throws-выражения.

Определение классов и методов

Для методов доступен любой из 3 возможных модификаторов доступа и допускается использование доступа по умолчанию.

Существует модификатор final, который говорит о том, что такой метод нельзя переопределять в наследниках.

Можно считать, что все методы final-класса, а также все private-методы любого класса, являются final.

Определение классов и методов

- В отличие от объявления переменной здесь запрещается указывать два имени для одного типа:

```
// void calc (double x, y); - ошибка!
 void calc (double x, double y);
```

Определение классов и методов

- Для каждого аргумента можно ввести ключевое слово final перед указанием его типа.
- В этом случае такой параметр не может менять своего значения в теле метода (то есть участвовать в операции присвоения в качестве левого операнда).

Определение классов и методов

```
public void process(int x, final double y) {
  x=x*x+Math.sqrt(x);
// y=Math.sin(x); - так писать нельзя,
 // т.к. y - final!
            }
```

Определение классов и методов

Важным понятием является сигнатура (signature) метода.

Сигнатура определяется именем метода и его аргументами (количеством, типом, порядком следования).

Если для полей запрещается совпадение имен, то для методов в классе запрещено создание двух методов с одинаковыми сигнатурами.

Определение классов и методов

```
class Point {
void get() {}
void get(int x) {}
void get(int x, double y) {}
void get(double x, int y) {}
}
```

Определение классов и методов

```
// ошибочное использование
void get() {}
int get() {}

void get(int x) {}
void get(int y) {}

public int get() {}
private int get() {}
```

Определение классов и методов

- Если текущая реализация метода не выполняет никаких действий, тело все равно должно описываться парой пустых фигурных скобок:

```
public void empty() {}
```

Определение классов и методов

- Если в заголовке метода указан тип возвращаемого значения, а не void, то в теле метода обязательно должно встречаться return-выражение.
- При этом компилятор проводит анализ структуры метода, чтобы гарантировать, что при любых операторах ветвления возвращаемое значение будет сгенерировано.

Определение классов и методов

```
// пример вызовет ошибку компиляции
 public int get()
   {
     if (condition)
         { return 5; }
   }
```

Определение классов и методов

```
// правильный метод
public int get() {
            if (condition)
             { return 5; }
            else { return 3; }
        }
```

Значение, указанное после слова **return**, должно быть совместимо по типу с объявленным возвращаемым значением

Определение классов и методов

В методе без возвращаемого значения (указано void) также можно использовать выражение return без каких-либо аргументов.

Его можно указать в любом месте метода и в этой точке выполнение метода будет завершено

Определение классов и методов

```
public void calculate(int x, int y) {
        if (x<=0 || y<=0)  {
          return;
    // некорректные входные
   // значения, выход из метода
        } ...
  // основные вычисления
}
```