Член класса (поле или метод), объявленный **public**, доступен из любого места вне класса. Все, что объявлено **private**, доступно только методам внутри класса и нигде больше. Если у элемента вообще не указан модификатор уровня доступа, то такой элемент будет виден и доступен из подклассов и классов того же пакета. Именно такой уровень доступа используется по умолчанию. Если же необходимо, чтобы элемент был доступен из другого пакета, но только подклассам того класса, которому он принадлежит, нужно объявить такой элемент со спецификатором **protected**. Действие спецификаторов доступа распространяется только на тот элемент класса, перед которым они стоят.

Спецификатор доступа **public** может также стоять перед определением внешнего (enclosing) класса. Если данный спецификатор отсутствует, то класс недоступен из других пакетов.

Конструкторы

Конструктор – это метод, который автоматически вызывается при создании объекта класса и выполняет действия по инициализации объекта. Конструктор имеет то же имя, что и класс; вызывается не по имени, а только вместе с ключевым словом **new** при создании экземпляра класса. Конструктор не возвращает значение, но может иметь параметры и быть перегружаемым.

Деструкторы в языке Java не используются, объекты уничтожаются сборщиком мусора после прекращения их использования (потери ссылки). Аналогом деструктора является метод **finalize()**. Исполняющая среда языка Java будет вызывать его каждый раз, когда сборщик мусора будет уничтожать объект класса, которому не соответствует ни одна ссылка.

```
/* пример # 2 : перегрузка конструктора: Quest.java */
package chapt03;

public class Quest {
      private int id;
      private String text;
      // конструктор без параметров (по умолчанию)
      public Quest() {
          super();/* если класс будет объявлен без конструктора, то
                     компилятор предоставит его именно в таком виде*/
      }
      // конструктор с параметрами
      public Quest(int idc, String txt) {
            super();/*вызов конструктора суперкласса явным образом
            необязателен, компилятор вставит его автоматически*/
            id = idc;
            text = txt;
      }
}
```

Объект класса **Quest** может быть создан двумя способами, вызывающими один из конструкторов:

```
Quest a = new Quest();//инициализация полей значениями по умолчанию
Quest b = new Quest(71, "Сколько бит занимает boolean?");
```

Оператор **new** вызывает конструктор, поэтому в круглых скобках могут стоять аргументы, передаваемые конструктору.

Если конструктор в классе не определен, Java предоставляет конструктор по умолчанию без параметров, который инициализирует поля класса значениями по умолчанию, например: **0**, **false**, **null**. Если же конструктор с параметрами определен, то конструктор по умолчанию становится недоступным и для его вызова необходимо явное объявление такого конструктора. Конструктор подкласса всегда вызывает конструктор суперкласса. Этот вызов может быть явным или неявным и всегда располагается в первой строке кода конструктора. Если конструктору суперкласса нужно передать параметры, то необходим явный вызов:

```
super(параметры);
```

В следующем примере объявлен класс **Point** с двумя полями (атрибутами), конструктором и методами для инициализации и извлечения значений атрибутов.

```
/* пример # 3 : вычисление расстояния между точками: Point.java:
LocateLogic.java: Runner.java */
package chapt03;

public class Point {
/* объект инициализируется при создании и не изменяется */
      private final double x;
      private final double y;

      public Point(final double xx, final double yy) {
            super();
            x = xx;
            y = yy;
      }
      public double getX() {
            return x;
      }
      public double getY() {
            return y;
      }
}
package chapt03;

public class LocateLogic {
      public double calculateDistance(
                                    Point t1, Point t2) {
            /* вычисление расстояния */
            double dx = t1.getX() - t2.getX();
            double dy = t1.getY() - t2.getY();
            return Math.hypot(dx, dy);
      }
}
package chapt03;
```

```
public class Runner {
      public static void main(String[] args) {
      // локальные переменные не являются членами класса
            Point t1 = new Point(5, 10);
            Point t2 = new Point(2, 6);
            System.out.print("расстояние равно : "
            + new LocateLogic().calculateDistance(t1, t2));
      }
}
```

В результате будет выведено:

```
расстояние равно : 5.0
```

Конструктор объявляется со спецификатором **public**, чтобы была возможность вызывать его при создании объекта в любом пакете приложения. Спецификатор **private** не позволяет создавать объекты вне класса, а спецификатор «по умолчанию» – вне пакета. Спецификатор **protected** позволяет создавать объекты в текущем пакете и для подклассов в других пакетах.

Методы

Изобретение методов является вторым по важности открытием после создания компьютера. Метод – основной элемент структурирования хода.

Все функции Java объявляются только внутри классов и называются методами. Простейшее определение метода имеет вид:

```
returnType methodName(список_параметров) {
    // тело метода
    return value;  // если нужен возврат значения (returnType не void)
}
```

Если метод не возвращает значение, ключевое слово **return** может отсутствовать, тип возвращаемого значения в этом случае будет **void**. Вместо пустого списка параметров метода тип **void** не указывается, а только пустые скобки. Вызов методов осуществляется из объекта или класса (для статических методов):

```
objectName.methodName();
```

Методы-конструкторы по имени вызываются автоматически только при создании объекта класса с помощью оператора **new**.

Для того чтобы создать метод, нужно внутри объявления класса написать объявление метода и затем реализовать его тело. Объявление метода как минимум должно содержать тип возвращаемого значения (возможен **void**) и имя метода. В приведенном ниже объявлении метода элементы, заключенные в квадратные скобки, являются необязательными.

```
[доступ] [static] [abstract] [final] [native]
[synchronized] returnType methodName(список_параметров)
                                [throws список_исключений]
```

Как и для полей класса, спецификатор доступа к методам может быть **public**, **private**, **protected** и по умолчанию. При этом методы суперкласса можно перегружать или переопределять в порожденном подклассе.