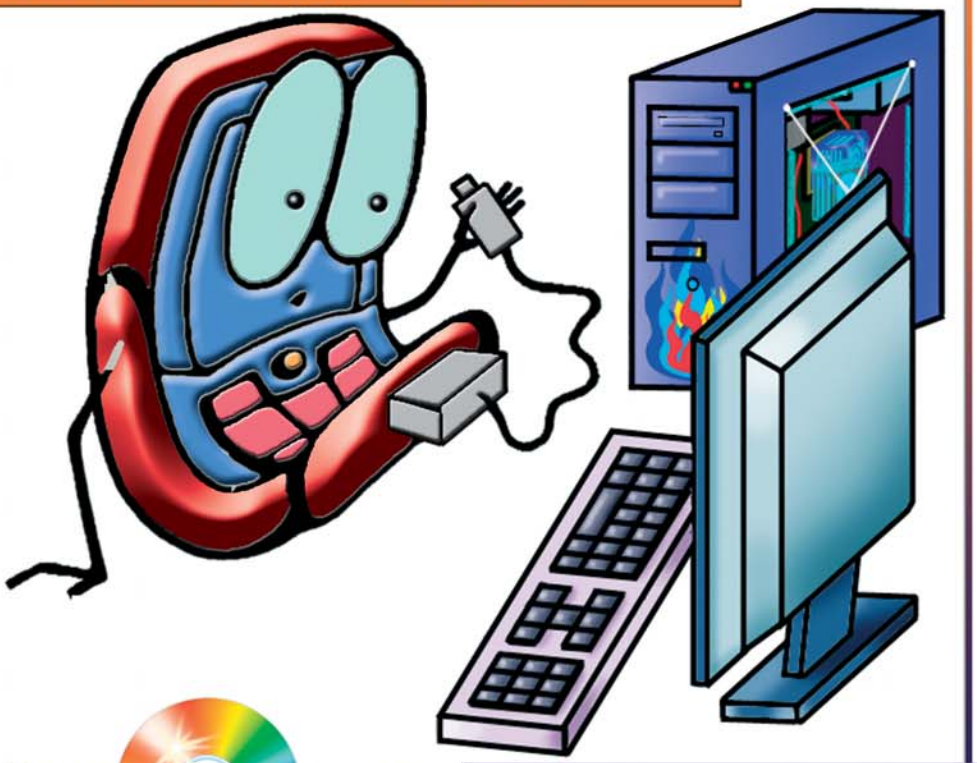
Мобильные технологии

ПРОГРАММИРОВАНИЕ МОБИЛЬНЫХ ТЕЛЕФОНОВ

Бестселлер!

на JAVA 2 ME



Компакт-диск к книге

Горнаков С. Г.

Второе дополненное и переработанное издание

ДМК ПРЕСС издательство

УДК 004.438
ББК 32.973.26-018.2

Горнаков С. Г.
Г26 Программирование мобильных телефонов на Java 2 Micro Edition. – М.: ДМК Пресс, 2008. – 512 с.: ил.

ISBN 5-94074-409-5

Вы держите в руках второе и переработанное издание одной из популярных книг о программировании мобильных телефонов на Java 2 ME. Первое издание книги продавалось огромными тиражами по всему постсоветскому пространству. Автор книги создал уникальное издание, обучившее огромное количество начинающих программистов делать приложения для мобильных телефонов. Спустя три года после выхода первого издания, по многочисленным заявкам читателей была создана новая и переработанная версия книги.

Книга содержит девять новых глав. Часть старого материала первого издания была переработана в соответствии с веяниями времени. Теперь читатель кроме программирования приложений для платформы Java 2 ME, изучит полный процесс создания мобильной игры. В течение книги будет освещен подход в формировании полноценного мобильного игрового движка, освоена работа с графикой, показаны примеры многослойных и анимированных игровых карт. Будут рассмотрены основы искусственного интеллекта, игровые столкновения, создание интерактивного меню игры, подсчет очков и жизненной энергии главного героя, сохранение данных в памяти, работа со звуком и многое другое. Итогом книги станет создание полноценной мобильной игры и знакомство с разработкой пользовательских программ на Java 2 ME.

Компакт-диск содержит средства разработки мобильных приложений NetBeans и J2ME Wireless Toolkit, а также большой набор телефонных эмуляторов от компаний Nokia, BenQ-Siemens, Sony Ericsson, Motorola и Samsung.

УДК 004.438
ББК 32.973.26-018.2

ISBN **5-94074-409-5**

Содержание

Глава 1. Основы языка программирования Java

Эта обзорная глава не претендует на роль полного руководства по языку программирования Java (Ява), но данного материала вам будет вполне достаточно для дальнейшего изучения книги. Предлагаемая в этом разделе информация к рассмотрению содержит основы языка Java и ориентирована на неподготовленного читателя. Я уверен, что после изучения этой главы вы сможете писать и читать исходные коды, встречающиеся в книге. Но нужно иметь в виду, что обучение языку Java будет происходить в соответствии с контекстом книги, а именно всей той части языка, которая необходима для программирования мобильных устройств. Такие «продвинутые» темы, как аплеты, библиотеки AWT, Swing, Graphic, в этой главе мы рассматривать не будем. В Java 2 Micro Edition все перечисленные (и не только) компоненты просто не используются.

1.1. Введение в программирование

Программирование – это написание исходного кода программы на одном из языков программирования. Существует множество различных языков программирования, благодаря которым создаются всевозможные программы, решающие определенный круг задач. *Язык программирования* – это набор зарезервированных слов, с помощью которых пишется исходный код программы. Компьютерные системы не в силах (пока) понимать человеческий язык и уж тем более человеческую логику (особенно женскую), поэтому все программы пишутся на языках программирования, которые впоследствии переводятся на язык компьютера или в машинный код. Системы, переводящие исходный код программы в машинный код, очень сложные, и их, как правило, создают не один десяток месяцев и не один десяток программистов. Такие системы называются *интегрированными средами программирования приложений*, или *инструментальными средствами*.

Система программирования представляет собой огромную продуманную визуальную среду, где можно писать исходный код программы, переводить его в машинный код, тестировать, отлаживать и многое другое. Дополнительно существуют программы, позволяющие производить вышеперечисленные действия при помощи командной строки. Язык Java предоставляет такую возможность, но в книге данная проблематика не освещается.

Вы, наверное, не раз слышали термин «программа написана под Windows или под Linux, Unix». Дело в том, что среды программирования при переводе языка программирования в машинный код могут быть двух видов – это *компиляторы*

и *интерпретаторы*. Компиляция или интерпретация программы задает способ дальнейшего выполнения программы на устройстве. Программы, написанные на языке Java, всегда работают на основе интерпретации, тогда как программы, написанные на C/C++, – компиляции. В чем разница этих двух способов?

Компилятор после написания исходного кода в момент компиляции читает сразу весь исходный код программы и переводит в машинный код. После чего программа существует как одно целое и может выполняться только в той операционной системе, в которой она была написана. Поэтому программы, написанные под Windows, не могут функционировать в среде Linux, и наоборот. Интерпретатор осуществляет пошаговое или построчное выполнение программы каждый раз, когда она выполняется. Во время интерпретации создается не выполняемый код, а виртуальный, который впоследствии выполняется виртуальной Java-машиной. Поэтому на любой платформе – Windows или Linux – Java-программы могут одинаково выполняться при наличии в системе виртуальной Java-машины, которая еще носит название *Системы времени выполнения*. Поскольку все телефоны сейчас оснащаются виртуальной Java-машиной, то и возможность запуска Java-программы на мобильном устройстве не вызовет проблем. Грубо говоря, производители мобильных устройств нашли способы (или разработали технологии), которые позволяют встраивать в телефоны виртуальную Java-машину, а значит, и запускать Java-программы на телефонах.

Итак, от вас сейчас требуется только изучение синтаксиса языка Java, для того чтобы писать и понимать исходные коды программ, написанные на этом языке. Этим мы и займемся в данной главе. В следующей главе вы узнаете больше о спецификациях языка Java 2 Micro Edition, или требованиях, на базе которых пишутся программы для телефонов. Затем мы освоим работу с одной из сред программирования и пакетами разработчика и далее перейдем к самому процессу программирования. А пока займемся языком Java и его объектно-ориентированной направленностью.

1.2. Объектно-ориентированное программирование

Объектно-ориентированное программирование строится на базе объектов, что в какой-то мере аналогично с нашим миром. Если оглянуться вокруг себя, то обязательно можно найти то, что поможет более ярко разобраться в модели такого программирования. Например, я сейчас сижу за столом и печатаю эту главу на компьютере, который состоит из монитора, системного блока, клавиатуры, мыши, колонок и т. д. Все эти части являются объектами, из которых состоит компьютер. Зная это, очень легко сформулировать какую-то обобщенную модель работы всего компьютера. Если не разбираться в тонкостях программных и аппаратных свойств компьютера, то можно сказать, что объект **Системный блок** производит определенные действия, которые показывает объект **Монитор**. В свою очередь, объект **Клавиатура** может корректировать или вовсе задавать действия для объекта

Системный блок, которые влияют на работу объекта **Монитор**. Представленный процесс очень хорошо характеризует всю систему объектно-ориентированного программирования.

Представьте себе некий мощный программный продукт, содержащий сотни тысяч строк кода. Вся программа выполняется построчно, строка за строкой, и в принципе каждая из последующих строк кода обязательно будет связана с предыдущей строкой кода. Если не использовать объектно-ориентированное программирование, и когда потребуется изменить этот программный код, скажем при необходимости улучшения каких-то элементов, то придется произвести большое количество работы со всем исходным кодом этой программы.

В объектно-ориентированном программировании все куда проще, вернемся к примеру компьютерной системы. Допустим, вас уже не устраивает 17-дюймовый монитор. Вы можете спокойно его обменять, например на 19-дюймовый или 20-дюймовый монитор, конечно же при наличии определенных материальных средств. Сам же процесс обмена не повлечет за собой огромных проблем, разве что драйвер придется сменить да вытереть пыль из-под старого монитора – и все. Примерно на таком принципе работы и строится объектно-ориентированное программирование, где определенная часть кода может представлять класс однородных объектов, которые можно легко модернизировать или заменять.

Объектно-ориентированное программирование очень легко и ясно отражает суть решаемой проблемы и, что самое главное, дает возможность без ущерба для всей программы убирать ненужные объекты, заменяя эти объекты на более новые. Соответственно, общая читабельность исходного кода всей программы становится намного проще. Существенно и то, что один и тот же код можно использовать в абсолютно разных программах.

1.2.1. Классы

Стержнем всех программ Java являются *классы*, на которых основывается объектно-ориентированное программирование. Вы по сути уже знаете, что такое классы, но пока об этом не догадываетесь. В предыдущем разделе мы говорили об объектах, ставя в пример устройство всего компьютера. Каждый объект, из которых собран компьютер, является представителем своего класса. Например, класс **Мониторов** объединяет все мониторы вне зависимости от их типов, размеров и возможностей, а один какой-то конкретный монитор, стоящий на вашем столе, и есть объект класса мониторов.

Такой подход позволяет очень легко моделировать всевозможные процессы в программировании, облегчая решение поставленных задач. Например, имеются четыре объекта четырех разных классов: монитор, системный блок, клавиатура и колонки. Чтобы воспроизвести звуковой файл, необходимо при помощи клавиатуры дать команду системному блоку, само же действие по даче команды вы будете наблюдать визуально на мониторе, и в итоге колонки воспроизведут звуковой файл. То есть любой объект является частью определенного класса и содержит в себе все имеющиеся у этого класса средства и возможности. Объектов одного класса может быть столько, сколько это необходимо для решения поставленной задачи.

1.2.2. Методы

Когда приводился пример воспроизведения звукового файла, то было упомянуто о даче команды или сообщения, на основе которого и выполнялись определенные действия. Задача по выполнению действий решается с помощью методов, которые имеет каждый объект. *Методы* – это набор команд, с помощью которых можно производить те или иные действия с объектом.

Каждый объект имеет свое назначение и призван решать определенный круг задач с помощью методов. Какой толк был бы, например, в объекте **Клавиатура**, если нельзя было бы нажимать на клавиши, получая при этом возможность отдавать команды? Объект **Клавиатура** имеет некое количество клавиш, с помощью которых пользователь приобретает контроль над устройством ввода и может отдавать необходимые команды. Обработка таких команд в целом происходит с помощью методов.

Например, вы нажимаете клавишу **Esc** для отмены каких-либо действий и тем самым даете команду методу, закрепленному за этой клавишей, который на программном уровне решает эту задачу. Сразу же возникает вопрос о количестве методов объекта **Клавиатура**, но здесь может быть различная реализация – как от определения методов для каждой из клавиш (что, вообще-то, неразумно), так и до создания одного метода, который будет следить за общим состоянием клавиатуры. То есть этот метод следит за тем, была ли нажата клавиша, а потом в зависимости от того, какая из клавиш задействована, решает, что ему делать.

Итак, мы видим, что каждый из объектов может иметь в своем распоряжении набор методов для решения различных задач. А поскольку каждый объект является объектом определенного класса, то получается, что класс содержит набор методов, которыми и пользуются различные объекты одного класса. В языке Java все созданные вами методы должны принадлежать или являться частью какого-то конкретного класса.

1.3. Синтаксис и семантика языка Java 2 ME

Для того чтобы говорить и читать на любом иностранном языке, необходимо изучить алфавит и грамматику этого языка. Подобное условие наблюдается и при изучении языков программирования, с той лишь разницей, как мне кажется, что этот процесс несколько легче. Но прежде чем начинать писать исходный код программы, необходимо сначала решить поставленную перед вами задачу в любом удобном для себя виде.

Давайте создадим некий класс, отвечающий, например, за телефон, который будет иметь всего два метода: включающий и выключающий этот самый телефон. Поскольку мы сейчас не знаем синтаксиса языка Java, то напишем класс `Телефон` на абстрактном языке.

```
Класс Телефон
{
   Метод Включить()
   {
```

```
    // операции по включению телефона
    }
    Метод Выключить()
    {
    // операции по выключению телефона
    }
}
```

Примерно так может выглядеть класс `Телефон`. Заметьте, что *фигурные скобки* обозначают соответственно начало и конец тела класса, метода либо всякой последовательности данных. То есть скобки указывают на принадлежность к методу или классу. На каждую открывающуюся скобку обязательно должна быть закрывающаяся скобка. Чтобы не запутаться, их обычно ставят на одном уровне в коде, как в приведенном выше примере.

А теперь давайте запишем тот же самый класс, только уже на языке Java.

```
class Telefon
{
    void on()
    {
    // тело метода on()
    }
    void off()
    {
    // тело метода off()
    }
}
```

Ключевое слово `class` в языке Java объявляет класс, далее идет название самого класса. В нашем случае это `Telefon`. Сразу пару слов касательно регистра записи. Почти во всех языках программирования важно сохранять запись названий в том регистре, в котором она была сделана. Если вы написали `Telefon`, то уже такое написание, как `telefon` или `TELefoN`, выдаст ошибку при компиляции. Как написали первоначально, так и надо писать дальше.

Зарезервированные или ключевые слова записываются в своем определенном регистре, и вы не можете их использовать, давая их названия методам, классам, объектам и т. д. Пробелы между словами не имеют значения, поскольку компилятор их просто игнорирует, но для читабельности кода они важны.

В теле класса `Telefon` имеются два метода: `on()` – включающий телефон и `off()` – выключающий телефон. Оба метода имеют свои тела, и в них по идее должен быть какой-то исходный код, описывающий необходимые действия обоих методов. Для нас сейчас не важно, как происходит реализация этих методов, главное – это синтаксис языка Java.

Оба метода имеют круглые скобки `on()`, внутри которых могут быть записаны параметры, например `on(int time)` или `on(int time, int time1)`. С помощью параметров происходит своего рода связь методов с внешним миром. Говорят,

что метод `on(int time)` принимает параметр `time`. Для чего это нужно? Например, вы хотите, чтобы телефон включился в определенное время. Тогда целочисленное значение в параметре `time` будет передано в тело метода, и на основе полученных данных произойдет включение телефона. Если скобки пусты, то метод не принимает никаких параметров.

1.3.1. Комментарии

В классе `Telefon` в телах обоих методов имеется запись после двух слэшей: `//`. Такая запись обозначает *комментарии*, которые будут игнорироваться компилятором, но нужны для читабельности кода. Чем больше информации вы закомментируете по ходу написания программы, тем больше у вас будет шансов вспомнить через год, над чем же все это время трудились.

Комментарии в Java могут быть трех видов, это: `//`, `/*...*/` и `/**...*/`. Комментарии, записанные с помощью оператора `//`, должны располагаться в одной строке:

```
// Одна строка
!!! Ошибка! На вторую строку переносить нельзя!
// Первая строка
// Вторая строка
// …
// Последняя строка
```

Комментарии, использующие операторы `/*...*/`, могут располагаться на нескольких строках. В начале вашего комментария поставьте `/*`, а в конце, когда закончите комментировать код, поставьте оператор `*/`.

Последний вид комментария `/**...*/` используется при документировании кода и также может располагаться на любом количестве строк.

1.3.2. Типы данных Java

Чтобы задать произвольное значение, в Java существуют *типы данных*. В классе `Telefon` мы создали два метода. Оба метода не имели параметров, но когда приводился пример метода `on(int time)` с параметром `time`, говорилось о передаче значения в метод. Данное значение указывало на время, с помощью которого якобы должен включиться телефон. Спецификатор `int` как раз и определяет тип значения `time`. В Java 2 ME шесть типов данных, которые перечислены в табл. 1.1:

Таблица 1.1

Тип	Назначение	Размер в байтах
byte	Маленькое целое	1
short	Короткое целое	2
int	Целое	4
long	Длинное целое	8
char	Символ	2
boolean	Логический тип	

- `byte` – маленькое целочисленное значение от –128 до 128;
- `short` – короткое целое значение в диапазоне от –32 768 до 32 767;
- `int` – содержит любое целочисленное значение от –2 147 483 648 до 2 147 483 647;
- `long` – очень большое целочисленное значение от –922 337 203 685 475 808 до 9 223 372 036 854 775 807;
- `char` – это символьная константа в формате Unicode. Диапазон данного формата – от 0 до 65 536, что равно 256 символам. Любой символ этого типа должен записываться в одинарных кавычках, например: 'G';
- `boolean` – логический тип, имеет всего два значения: `false` – ложь и `true` – истина. Этот тип часто используется в циклах, о которых чуть позже. Смысл очень прост: если у вас в кармане есть деньги, предположительно это `true`, а если нет – то `false`. Таким образом, если деньги имеются – идем в магазин за хлебом или пивом (нужное подчеркнуть), если нет денег – остаемся дома. То есть это такая логическая величина, которая способствует выбору дальнейших действий вашей программы.

Чтобы объявить какое-то необходимое значение, используется запись:

```
int   time;
long  BigTime;
char  word;
```

Оператор точка с запятой необходим после записей и ставится в конце строки. Можно совместить несколько одинаковых по типу объявлений через запятую:

```
mt  time, time1, time2;
```

Теперь давайте усовершенствуем наш класс `Telefon`, добавив в него несколько значений. Методы `on()` и `off()` нам больше не нужны, добавим новые методы, которые действительно могут решать определенные задачи.

```
class Telefon
{
//S — площадь дисплея
//w — ширина дисплея
//h — высота дисплея
int w, h, S;
//метод, вычисляющий площадь дисплея
   vord Area()
   {
         S = w*h;
   }
}
```

Итак, мы имеем три переменные `S`, `w` и `h`, отвечающие соответственно за площадь, ширину и высоту дисплея в пикселях. Метод `Area()` вычисляет площадь

экрана телефона в пикселях. Операция бесполезная, но очень показательная и простая в понимании. Тело метода `Area()` обрело себя и имеет вид `S = w*h`. В этом методе мы просто перемножаем ширину на высоту и присваиваем, или, как еще говорят, сохраняем результат в переменной `S`. Эта переменная будет содержать значения площади дисплея данного телефона. Сейчас мы подошли вплотную к операторам языка Java, с помощью которых можно совершать всевозможные операции.

1.3.3. Операторы

Операторы языка Java имеют различные назначения. Существуют арифметические операторы, операторы инкремента и декремента, логические операторы и операторы отношения.

Арифметические операторы очень просты и аналогичны операторам умножения «*», деления «/», сложения «+» и вычитания «–», используемым в математике. Существует оператор деления по модулю «%» и слегка запутанная на первый взгляд ситуация с оператором равно «=». Оператор «равно» в языках программирования называется оператором присваивания:

```
int x = 3
```

Здесь вы присваиваете переменной `x` значение 3. А оператор «равно» в языках программирования соответствует записи двух подряд операторов «равно»: «==». Рассмотрим на примере, что могут делать различные арифметические операторы.

```
int x, y, z;
x = 5;
y = 3;
z = 0;
z = x + y;
```

В данном случае `z` будет иметь значение уже суммы `x` и `y`, то есть 8.

```
x = z*x;
```

Переменная `x` имела значение 5, но после такой записи предыдущее значение теряется, и записывается произведение `z*x (8*5)`, что равно 40. Теперь если мы продолжим дальше наш код, то переменные будут иметь такой вид:

```
// x = 40;
// y = 3;
// z = 8;
```

Что касается оператора деления, то поскольку Java 2 ME и конфигурация CLDC 1.0 не поддерживают дробных чисел, то результат такого деления:

```
x = z/y;
```

что равносильно записи:

```
x = 8/3;
```

будет равен 2. Дробная часть просто отбрасывается, то же самое происходит при использовании оператора деления по модулю «%». В новой версии CLDC 1.1 реализована поддержка дробных чисел.

Операторы сложения и вычитания имеют те же назначения, что и в математике. Отрицательные числа также родственны.

Операторы декремента «--» и инкремента «++» весьма специфичны, но очень просты. В программировании часто встречаются моменты, когда требуется увеличить или уменьшить значение на единицу. Часто это встречается в циклах. Операция инкремента увеличивает переменную на единицу.

```
int x = 5;
x++;
// Здесь x уже равен 6
```

Операция декремента уменьшает переменную на единицу.

```
int x = 5;
x-;
// x равен 4
```

Операции инкремента и декремента могут быть пост- и префиксными:

```
int x = 5;
int y = 0;
y = x++;
```

В последней строке кода сначала значение x присваивается y, это значение 5, и только потом переменная x увеличивается на единицу. Получается, что:

```
x = 6, y = 5
```

Префиксный инкремент имеет вид:

```
int x = 3;
int y = 0;
y = ++x;
```

И в этом случае сначала переменная x увеличивается на один, а потом присваивает уже увеличенное значение y.

```
y = 4, x = 4
```

Операторы отношения

Операторы отношения позволяют проверить равенство обеих частей выражения. Имеются оператор равенства «==», операторы меньше «<» и больше «>», меньше или равно «<=» и больше или равно «>=», а также оператор отрицания «!=».

```
9 == 10;
```

Это выражение не верно, девять не равно десяти, поэтому значение этого выражения равно false.

```
9 != 10;
```

Здесь же, наоборот, оператор отрицания указывает на неравенство выражения, и значение будет равно `true`.

Операторы больше, меньше, больше или равно и меньше или равно аналогичны соответствующим операторам из математики.

Логические операторы

Существует два *логических оператора*. Оператор «И», обозначающийся значками «`&&`», и оператор «ИЛИ», обозначенный в виде двух прямых слэшей «`||`». Например, имеется выражение:

```
A*B && B*C;
```

В том случае если только обе части выражения истинны, значение выражения считается истинным. Если одна из частей неверна, то значение всего выражения будет ложным.

В противовес оператору «`&&`» имеется оператор «`||`», не напрасно имеющий название «ИЛИ».

```
A*B || B*C;
```

Если любая из частей выражения истинна, то и все выражение считается истинным. Оба оператора можно комбинировать в одном выражении, например:

```
A*B || B*C && C*D || B*A;
```

С помощью этого выражения я вас ввел, как мне кажется, в затруднение, не правда ли? Дело в том, что в Java, как и в математике, существует приоритет, или так называемая *иерархия операторов*, с помощью которой определяется, какой из операторов главнее, а следовательно, и проверяется первым. Рассмотрим с помощью списка приоритет всех имеющихся операторов языка Java:

```
[], ., (),
!, ~, ++, — —, + (унарный), — (унарный), new,
*, /, %,
+, —,
<<, >>, >>>,
<, <=, >, >=,
= =, !=,
&, ^, |,
&&,
||,
?:,
=, +=, —=, *=, /=, %=, |=, ^=, <<=, >>=, >>>=.
```

Не со всеми операторами вы еще знакомы, поэтому пугаться не стоит. Ассоциативность операторов в списке следует слева направо и сверху вниз. То есть все, что находится левее и выше, старше по званию и главнее.

1.3.4. Метод main

Класс Telefon, который мы описывали в предыдущем разделе, имел один метод, с помощью которого вычислялась площадь дисплея. Созданная спецификация класса Telefon может быть описана как угодно. Можно добавить методы, реагирующие на обработку событий с клавиатуры телефона, и любые другие методы, которые вы сочтете нужными для описания класса Telefon. Таких и подобных классов может быть любое количество. Каждый из классов принято хранить в своем отдельном файле с расширением *.java (например: Telefon.java).

Все методы, находящиеся в классе Telefon, которые вы опишете для данного класса, обязаны производить определенные действия с объектом этого класса, иначе зачем тогда нужны все эти методы. Реализация методов, как уже говорилось, происходит непосредственно в создаваемом классе. Но возникает вопрос: где и как происходит вызов необходимых по ситуации методов или создание объектов используемого класса. В языке Java для этих целей существует *метод* main(), который подобно строительной площадке собирает на своей платформе по частям объекты, методы, строя при этом функциональность всей программы.

```
public class RunTelefon
{
   public static void main( String[] args)
   {
   // Работа программы
   }
}
```

Пока не обращайте внимания на ключевые слова public и static – о них чуть позже. Создав класс RunTelefon и определив в его теле метод main(), мы теперь имеем возможность пользоваться классом Telefon и его не особо богатой функциональностью. Объединив эти два класса в один файл либо записав каждый по отдельности, вы получите работоспособную программу. Класс Telefon содержит основные данные и методы, а класс RunTelefon берет на себя роль мотора. Внутри класса RunTelefon происходят создание объектов класса, в данном случае класса Telefon, и вызовы соответствующих методов этого класса.

```
class Telefon
{
   // переменные
   int w, h, s;
   // метод
   void Area()
   {
   S = w*h;
   }
}
```

```
class RunTelefon
{
   public static void main (String args[])
   {
   /* создание объекта/ов класса Telefon и вызовы метода
   Area() */
   }
}
```

Поскольку вы пока не умеете создавать объекты классов и вызывать методы, тело класса `RunTelefon` пустое. Прежде чем идти дальше, необходимо познакомиться с ключевыми словами `public`, `private` и `protected`, а также научиться создавать конструкторы классов.

1.3.5. Закрытые и открытые члены классов

Ключевое слово `public`, объявленное перед методом `main()`, показывает на то, что метод `main()` считается открытым, или, как говорят в любом классе, метод `main()` виден, и к нему можно обратиться. Ключевое слово `public` может назначаться не только методам, но и объектам класса, переменным, любым членам созданного класса. Все объявленные переменные с ключевым словом `public` будут доступны всем другим существующим в программе классам, а это может иногда навредить программе. Например, у вас есть какие-то данные, которые не должны быть доступны другим классам, что тогда? Для этого в языке Java существует еще пара ключевых слов: `private` и `protected`, – благодаря которым вы защищаете переменные или члены классов от общего доступа.

По умолчанию, если вы не используете никаких ключевых слов при объявлении объектов, методов или переменных, язык Java назначает всем членам класса спецификатор `public`. Метод `main()` всегда должен вызываться с ключевым словом `public`, чтобы для всех классов программы метод `main()` был доступен. Как только программа начнет работать, первым делом она ищет метод `main()` и постепенно, шаг за шагом, а точнее, строка за строкой, выполняет все предписанные действия в этом методе.

1.4. Конструктор

Каждый класс обязан содержать конструктор. *Конструктор* – это тот же самый метод, но имеющий название класса, например:

```
class Telefon
{
   Telefon();   // конструктор
   int w, h, s; // переменные
   void Area(); // метод
}
```

Конструктор позволяет инициализировать или создает объекты данного класса с заданными значениями. Каждый класс имеет конструктор, и если вы явно не

записали строку кода (как в нашем случае `Telefon()`), Java автоматически создаст его за вас, и такой конструктор носит название конструктора *по умолчанию*.

Конструкторы в программировании очень полезны, и ни одна профессиональная программа не обходится без конструкторов. Чтобы действительно ощутить мощь конструктора, надо создать конструктор с аргументами, благодаря которым можно инициализировать данные класса.

```
class Telefon
{
   // переменные
   int w, h, s;

   // конструктор класса
   Telefon(int a, int b)
   {
         w = a;
         h = b;
   }

   // метод, вычисляющий площадь дисплея телефона
   void Area()
   {
         s = w*h;
   }
}
```

При создании объекта (об этом чуть позже) вы можете указать необходимые значения для параметров `a` и `b`, например: `a = 70`, `b = 100`. Эти заданные числа автоматически присвоятся переменным `w` и `h` при создании объекта класса `Telefon`. Тем самым произойдет инициализация объекта с необходимыми значениями.

Количество конструкторов в классе ограничивается только вашей фантазией и здравым смыслом. Например, можно создать два конструктора классу `Telefon`:

```
Telefon (int a, int b);
Telefon (char a, char b);
```

В этом случае при создании объекта по переданным параметрам конструктору класса компилятор сам выберет необходимый конструктор и создаст заданный объект.

1.5. Объекты классов

Объекты представляют класс, наследуя от своего класса все возможности. Объявить объект очень просто, необходимо вначале указать класс, а потом – объект этого класса.

```
Telefon object;
```