1 Общее введение в программирование

В этой главе:

- ключевые шаги процесса программирования;
- различные ошибки программирования;
- ключевые принципы тестирования программного обеспечения (ПО);
- различные виды сопровождения ПО;
- ключевые принципы структурного программирования.

Загрузки кода для этой главы от wrox.com

Загрузки кода для этой главы от wrox.com можно найти по веб-адресу http://www.wrox.com/WileyCDA/WroxTitle/Beginning-Java-Programming-The-Object-Oriented-Approach.productCd-1118739493.html на вкладке Download Code. Фрагменты кода находятся в загрузках к главе 1 и индивидуально поименованы в соответствии с названиями в данной главе.

Разработка качественного и правильного программного обеспечения является крайне важной задачей в современной деловой среде. Учитывая нынешнее повсеместное использование программ, теперь некачественное ПО влияет на жизнь больше, чем когда-либо. Ошибки программного обеспечения служили причинами, например, авиакатастроф, неудачных пусков ракет и отключений электроэнергии. Следовательно, важно разрабатывать высококачественное и безотказное ПО.

В текущей главе рассматриваются фундаментальные понятия программирования. Сначала в ней будет описан процесс программирования. В следующем разделе мы кратко познакомимся с основами объектно-ориентированного программирования, обсудим ошибки программирования, а также основные принципы тестирования и сопровождения ПО. В заключение будут даны некоторые рекомендации, касающиеся структурного программирования. В последующих главах мы снова рассмотрим многие из этих тем с более практической точки зрения.

Процесс программирования

Программой (или приложением) называют набор команд, призванных решить определенную задачу, которая может быть однозначно понята компьютером. Для этого он переведет программу на понятный ему машинный язык, состоящий из 0 (нулей) и 1 (единиц). Компьютеры выполняют программу буквально так, как она написана, ни больше, ни меньше.

Программирование — деятельность по написанию или кодированию программы на определенном языке программирования. Этот язык имеет строгие правила грамматики и синтаксиса, символы и специальные ключевые слова. Тех, кто пишет программы, обычно называют программистами или разработчиками приложений. Под термином «программное обеспечение» понимается набор программ в определенном бизнес-контексте.

В качестве примера упражнения по программированию рассмотрим программу, которая рассчитывает индекс массы тела (ИМТ) человека. ИМТ определяется путем деления веса в килограммах на рост в метрах в квадрате. Вес считается избыточным, если ИМТ превышает 25. Программа-калькулятор требует вес и рост в качестве входных данных, вычисляет связанный с ними ИМТ и выдает результат (рис. 1.1). Данный пример показывает шаги в цикле разработки программного обеспечения.

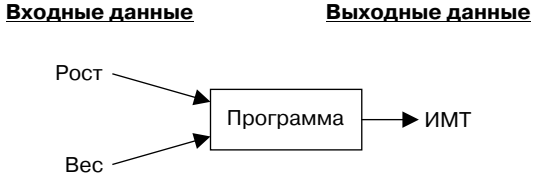


Рис. 1.1

Как правило, при разработке программы используется пошаговый подход, изложенный ниже.

1. Сбор и анализ требований.
2. Проектирование программы.
3. Написание кода.
4. Перевод на машинный язык.
5. Тестирование и отладка.
6. Развертывание.
7. Сопровождение.

Поскольку наша окружающая среда постоянно меняется, программное обеспечение обычно неоднократно пересматривают и адаптируют. Таким образом, этапы разработки часто представляют собой цикл (рис. 1.2), а не многоступенчатую схему.

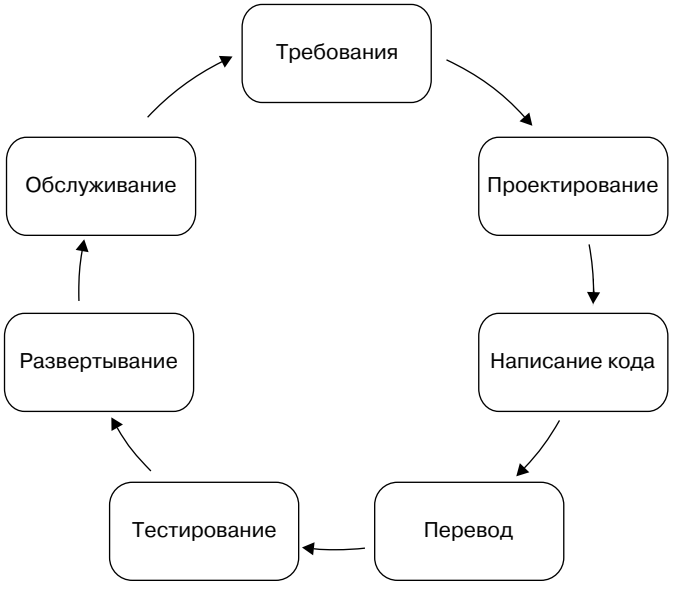


Рис. 1.2

Прежде чем сделать первый шаг, убедитесь, что достаточно хорошо понимаете задачу. Это значит тщательно проанализируйте условие задачи, чтобы в полной мере осознать, какие требования должна выполнять программа. Анализ может включать совещания «вопрос-ответ», интервью и опросы бизнес-экспертов, владеющих необходимыми профессиональными знаниями. Даже если вы программируете для себя, заранее выделите время для рассмотрения всех требований, предъявляемых программе, — это сократит количество изменений, которые позже могут понадобиться в процессе работы.

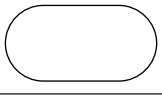
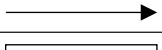
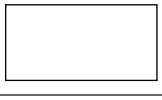
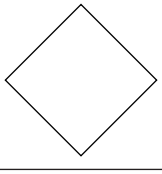
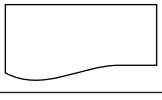
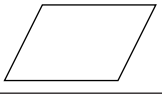
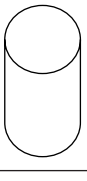
В завершение данного шага важно понимать, какие входные данные получит программа и какие выходные данные она должна представить. В примере с ИМТ вы должны знать, в чем будет измеряться рост (в метрах или футах) и вес (в килограммах или фунтах). Вы также захотите определить, будет выводиться лишь значение ИМТ или еще и сообщение о наличии у человека избыточного веса.

Когда вы поймете, в чем заключается деловая задача, можно начать думать о путях ее решения с помощью компьютерной программы. Другими словами, какие шаги нужно выполнить на этапе ввода данных, чтобы получить необходимые выходные данные? Процедуру, которая призвана решить задачу, часто называют *алгоритмом*. При его разработке здравый смысл и творческое мышление играют важную роль. Первый полезный шаг при создании алгоритма — планирование прикладной логики с помощью псевдокода (абстрактного кода) или структурной схемы. Псевдокод представляет собой разновидность структурированного английского языка, но без строгих правил грамматики. Это удобный для пользователя способ описания логики приложения в последовательном, удобочитаемом формате. Таким образом, чтобы упростить изложение задачи, ее можно представить в виде коротких отрывков. Ниже приведен пример псев-

докода для задачи с ИМТ. Структурная схема представляет приложение в виде диаграммы, в которой блоки показывают действия, а стрелки — их последовательность. В табл. 1.1 перечислены наиболее важные понятия построения структурных схем. На рис. 1.3, в свою очередь, приведен пример структурной схемы для задачи с ИМТ. Псевдокод и структурные схемы можно использовать одновременно, чтобы облегчить процесс программирования. Главным преимуществом структурных схем в сравнении с псевдокодом является их наглядность и, следовательно, простота толкования.

```
ask user: height
ask user: weight
if height = 0 or weight = 0:
error: "Incorrect input values"
return to beginning (ask height and weight)
end if
x = weight / (height * height)
message: "Your BMI is ",x
```

Таблица 1.1. Ключевые понятия построения структурных схем

Символ структурной схемы	Значение
	Терминатор (разделитель) показывает точки входа и выхода программы
	Стрелка показывает направление последовательности операций
	Прямоугольник представляет этап процесса или вычислений
	Ромб обозначает точку решения в процессе
	Этот символ представляет документ или отчет
	Ромбовидная фигура представляет данные, используемые в качестве входных (выходных) значений в процесс (или из него)
	Цилиндр обозначает базу данных

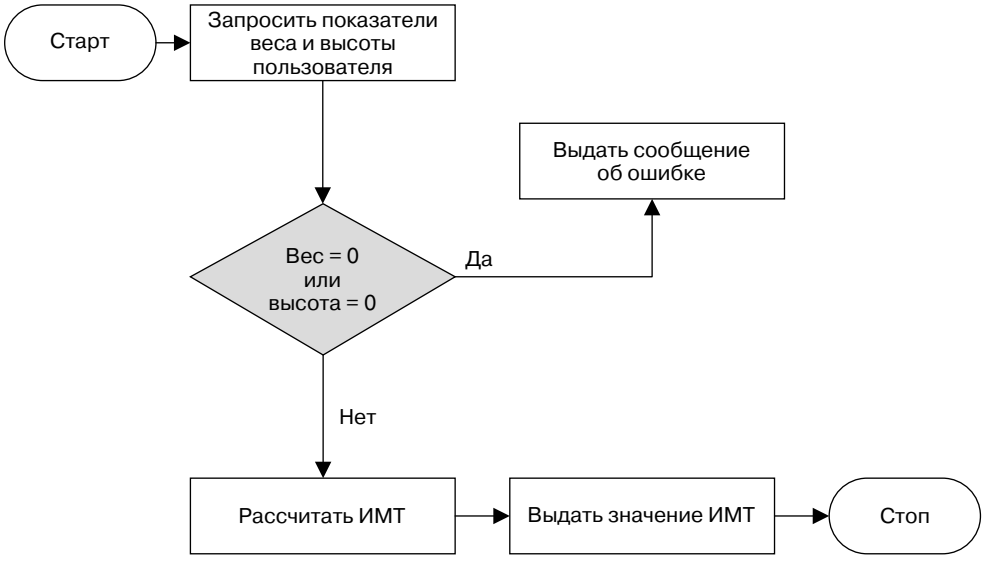


Рис. 1.3

Следующий шаг — написать программу на определенном языке программирования. Выбор языка будет зависеть от парадигмы программирования и принятой платформы (аппаратного обеспечения, операционной системы или сети).

После написания исходный код передается в компилятор для перевода на машинный язык (нули и единицы), последующего выполнения и решения деловой задачи.

Во время разработки приложения важно интенсивно тестировать каждую программу, чтобы избежать каких-либо ошибок. В программировании ошибки часто называют *багами*. Есть различные типы ошибок, и им посвящена целая глава в этой книге. Средства программирования часто имеют встроенные инструменты отладки, которые позволяют легко отслеживать ошибки и исправлять их. Отладка возможна и без использования подобных инструментов, но в любом случае необходим структурированный и систематический обзор ошибок, чтобы выявлять их до развертывания вашей программы.

После тщательного тестирования программу можно развернуть. Это значит, что программу введут в эксплуатацию и будут активно использовать для решения деловой задачи. Помните, что пользователи ваших приложений обычно не владеют программированием так же хорошо, как вы. Старайтесь не забывать об этом во время работы, чтобы в дальнейшем этап развертывания прошел максимально просто.

Наконец, программам необходимо постоянное сопровождение. Есть много причин для их регулярного обслуживания, в частности исправление выявленных багов, удовлетворение новых запросов потребителей, предотвращение ввода пользователями ошибочных данных или добавление новых функций в существующие программы.

Важно отметить, что программирование не является строго заданной последовательностью шагов. Напротив, оно часто представляет собой цикличный процесс, в котором первоначальная формулировка деловой задачи может дорабатываться или даже изменяться при написании кода.

Объектно-ориентированное программирование: предварительное знакомство

В объектно-ориентированном программировании (ООП) приложение состоит из групп объектов, которые запрашивают друг у друга различные сервисы. Каждый объект является экземпляром класса, содержащего описание всех характеристик объекта. В отличие от процедурного программирования, объект последовательно объединяет как свои данные (определяющие его состояние), так и процедуры (определяющие его поведение). В качестве примера можно привести объект `student`, содержащий элементы данных, такие как удостоверяющий личность документ (ID), имя, дата рождения, адрес электронной почты и т. д., а также процедуры, такие как `registerForCourse`, `isPassed` и т. д. Главное различие между ООП и процедурным программированием заключается в том, что ООП использует внутренние данные, хранящиеся в объектах, тогда как процедурное — глобальные общие данные, к которым различные процедуры могут обращаться напрямую. С точки зрения сопровождения продукта такая разница является принципиальной. Представьте, что вам нужно изменить какой-либо элемент данных (переименовать его или удалить). В условиях процедурного программирования придется искать все процедуры, использующие требуемый элемент, и соответствующим образом их адаптировать. В случае с комплексными программами сопровождение может быть крайне трудоемкой задачей. Но если вы задействуете парадигму ООП, то потребуется лишь изменить элемент данных в определении объекта, что не повлияет на его взаимодействие с другими объектами, и сопровождение сведется к минимуму.

ООП — это самая популярная парадигма программирования, используемая в настоящее время. В качестве примеров объектно-ориентированных языков программирования можно назвать Eiffel, Smalltalk, C++ и Java.

Следующий фрагмент кода показывает, как пример с ИМТ реализовать в Java. В отличие от примера с процедурным программированием, очевидно, что данные (вес, рост и ИМТ) объединены с процедурами (`BMICalculator`, `calculate` и `isOverweight`) в одно связное определение класса.

```
public class BMICalculator {
  private double weight, height, BMI;

  public BMICalculator( double weight, double height ){
    this.weight = weight;
    this.height = height;
  }

  public void calculate(){
```

```
    BMI = weight / (height*height);
  }

  public boolean isOverweight(){
    return (BMI > 25);
  }
}
```

Ошибки программирования

Ошибку программирования также называют багом, а процедуру устранения ошибок программирования — *отладкой*. Отладка обычно состоит из следующих трех шагов.

1. Определить наличие ошибки.
2. Найти ошибку. В больших программах на это может уйти довольно много времени.
3. Устранить ошибку.

Существуют различные типы ошибок программирования, и все они рассматриваются ниже.

Ошибки синтаксиса/компиляции

Синтаксические или компиляционные ошибки в программе относятся к грамматическим. Примерами являются ошибки пунктуации или неправильное написание ключевого слова. Эти типы ошибок обычно перехватываются компилятором или интерпретатором, которые сгенерируют соответствующее сообщение. Рассмотрим следующий пример с Java:

```
public void calculate(){
  BMI = weight / (height*height),
}
```

Выражение, которое вычисляет ИМТ, в соответствии с правилами синтаксиса Java должно заканчиваться точкой с запятой (`;`) а не запятой (`,`). Следовательно, генерируется и выдается синтаксическая ошибка. Такие ошибки обычно легко обнаружить и устранить.

Ошибки времени выполнения

Это ошибка, возникающая во время работы программы. Рассмотрим следующий фрагмент кода Java для расчета ИМТ:

```
public void calculate(){
  BMI = weight / (height*height);
}
```

Если пользователь вводит значение `0` для роста, то произойдет деление на нуль. Возникнет ошибка времени выполнения, и она, вероятнее всего, приведет к сбою приложения. Еще один пример — бесконечный цикл, куда входит программа при выполнении. В ходе проектирования программы важно подумать о возможных ошибках времени выполнения, которые могут возникнуть из-за неверных данных, введенных пользователем, служащих причиной большинства багов. Старайтесь прогнозировать и отслеживать эти ошибки, насколько возможно, с помощью соответствующих методов, которые мы обсудим чуть позже.

Логические/семантические ошибки

Логические или семантические ошибки обнаружить сложнее всего, поскольку программа выдаст выходные данные и не сгенерирует ошибку. Тем не менее эти данные будут неверными из-за неправильно запрограммированной формулы. Снова рассмотрим пример с ИМТ:

```
public void calculate(){
  BMI = (weight*weight) / height;
}
```

Функция явно ошибочна, поскольку рассчитывает ИМТ как (вес * вес) / рост, а не вес / (рост * рост). Подобные ошибки не могут быть обнаружены компиляторами или интерпретаторами.

Принципы тестирования программного обеспечения

Чтобы избежать ошибок в ПО и их последствий, до ввода программы в эксплуатацию необходимо тщательно тестировать ее на наличие любых оставшихся погрешностей. Основной целью тестирования является верификация и валидация версии ПО. Верификация отвечает на вопрос о том, *правильно ли* была собрана система, в то время как валидация пытается определить, *правильная ли* система была собрана. Чем быстрее обнаружится ошибка во время разработки, тем дешевле обойдется ее устранение. Как показано на рис. 1.4, стоимость тестирования обычно увеличивается по экспоненте, в то время как затраты на пропущенные ошибки уменьшаются в геометрической прогрессии вместе с количеством проведенных тестов. В точке пересечения обеих кривых и находится оптимальный показатель затраченных на тестирование ресурсов.

Первый основной способ тестирования — «настольный анализ» программы с помощью бумаги и карандаша. Вычисления вручную и выходные данные можно сопоставить с программными расчетами и их результатами. Это особенно важно при рассмотрении исключительных случаев и оценке поведения программы. Разумеется, такой способ подходит лишь для небольших программ, а для более комплексных проектов потребуется нечто посложнее.