УДК 004.43
ББК 32.973.26-018.1
Б51

Бернштейн, Дэвид.

Б51 Паттерны для начинающих программистов с примерами на Java / Д. Бернштейн. — Москва : Издательство АСТ, 2024. — 256 с. : ил. — (Учимся программировать).

ISBN 978-5-17-162110-0.

В этой книге, призванной научить начинающего программиста решать конкретные задачи по программированию на языке Java, вводится понятие паттерна — некоего заранее заготовленного «рецепта» решения, который можно применить в виде готового фрагмента кода. Используя впоследствии данный набор паттернов, молодой разработчик сможет на их основе быстро решать довольно сложные составные задачи. Представленные здесь паттерны часто предлагают наряду со стандартными алгоритмами эффективные альтернативные методы решения самых распространенных задач в области программирования. Приведенная в книге библиотека паттернов охватывает обширную область, начиная с примитивного обновления переменной и заканчивая работой со ссылочными данными. Кроме того, рассмотрены такие темы, как манипуляции с цифрами, входящими в состав числа, арифметика на числовой окружности, применение переменных-индикаторов и переменных-аккумуляторов, конформные и сегментированные массивы, операции с отдельными битами и многие другие.

УДК 004.43
ББК 32.973.26-018.1

ISBN 978-5-17-162110-0

Содержание

Предисловие

Мои вводные курсы по программированию всегда начинаются с некоторых определений, которые могут показаться слегка расплывчатыми. Я определяю *алгоритм* как однозначный процесс решения задачи с использованием конечного количества ресурсов, а *эвристику* как процесс решения задачи, который не всегда может быть идеальным/точным или конечным. Затем я объясняю, что алгоритмы/эвристики, написанные для компьютера, обычно пишутся на так называемых высокоуровневых *языках программирования*, которые легко понимаются человеком, однозначны и легко преобразуются в машиночитаемую форму. Затем я отмечаю, что алгоритм/эвристика, написанные на языке программирования, являются *программой* (или *кодом*), а процесс их создания называется *программированием* (или *кодированием*), и это как раз то, что студенты будут изучать в течение оставшейся части семестра.

На практике большинство вводных курсов по программированию, в том числе и мой, используют один конкретный язык программирования, и неудивитель-

но, что ведутся активные споры о том, какой язык лучше всего подходит для обучения начинающих программистов (и имеет ли это значение). Однако в целом существует консенсус относительно роли, которую играет язык программирования, и целей таких курсов. Действительно, большинство вводных курсов по программированию утверждают следующее:

1. Концепции, рассматриваемые на конкретном языке программирования, выбранном для курса, применимы в широком спектре и для других языков программирования.

2. Курс в большей степени учит алгоритмическому мышлению, чем синтаксису используемого языка (языков).

По моему опыту, эти утверждения не совсем справедливы. Если первое утверждение, конечно, правильное с точки зрения преподавания, то с точки зрения обучения оно неверно. Другими словами, студенты испытывают огромные трудности с тем, чтобы взять то, что они выучили на одном языке, и применить это для другого языка программирования. Что касается второго утверждения — это честное описание того, что преподаватель хотел бы охватить, но эта цель редко достигается.

В худшем случае преподаватели на вводных курсах по программированию тратят все свое время на обучение студентов синтаксису и инструментам, а решение задач оставляют на усмотрение обучающихся. Используя понятия из таксономии когнитивной области Блу-

ма (оценка по таксономии Блума показывает, какие темы даются ученику с трудом и готов ли он применить полученные знания на практике), преподаватели таких курсов учат только “запоминать” и “понимать” и ожидают, что студенты будут самостоятельно “анализировать”, “оценивать”, “применять” и “создавать”.

В лучшем случае преподаватели на вводных курсах по программированию знакомят студентов с хорошо написанным кодом, содержащим удачные решения задач, и объясняют, почему этот код хорош. Однако они лишь надеются, что студенты усвоят эти решения и смогут применять их в других контекстах; они не учат их делать это явно. Другими словами, эти курсы учат только “запоминать”, “понимать”, “анализировать” и “оценивать”, но не “применять” и “создавать”.

В самом же лучшем случае преподаватели на вводных курсах по программированию учат не только “применять”, но и “создавать”. То есть они учат не только программированию, но и *решению задач*. Эта книга призвана решить данную проблему.

Обучение решению задач с помощью паттернов

На протяжении многих лет у меня были коллеги, которые утверждали, что решению задач нельзя научить, что у студентов либо есть способности, либо их нет. Я (и другие) с этим не согласен и считаю, что студентов можно научить решать задачи следующим образом:

1. Продемонстрировать им, что задачи можно классифицировать.

2. Помочь им научиться оценивать качество различных решений.

3. Обеспечить удачные общие решения для многих классов задач.

4. Помочь им научиться классифицировать новые задачи.

5. Помочь им научиться адаптировать существующее общее решение к новой конкретной задаче.

В этом и заключается суть обучения решению задач с помощью *паттернов* — подхода, основанного на работах Кристофера Александера (т. е. так называемом движении языка паттернов в архитектуре).

Обучение проектированию и решению задач с использованием паттернов предполагает представление о следующих понятиях:

1. Архетипическая/каноническая проблема.

2. Одно или несколько некачественных решений проблемы.

3. Превосходное решение проблемы (т. е. решение, использующее паттерн).

4. Другие проблемы, к которым может быть применено превосходное решение (с небольшими изменениями).

5. Способы определения того, что тот или иной паттерн является подходящим решением для конкретной проблемы (т. е. определение класса, к которому относится проблема).

В итоге такой подход дает студентам две вещи: библиотеку решений, которую они могут использовать, и понимание того, как пополнять эту библиотеку самостоятельно. Когда они сталкиваются с новой проблемой, их задача чаще всего состоит в том, чтобы понять, что решение у них уже есть. Реже они должны осознать, что у них нет готового решения, и понять, что они должны разработать альтернативные решения, оценить эти решения и выбрать наилучший вариант.

Паттерны для начинающих программистов

Этот подход уже много лет успешно используется (под разными названиями) в курсах продвинутого уровня по разработке программного обеспечения и другим инженерным дисциплинам. К сожалению, он все еще не вошел во вводные курсы по программированию. Цель данной книги — попытаться исправить этот недостаток.

Два момента отличают *паттерны программирования* от других видов паттернов: предметная область (т. е. программирование) и уровень абстракции. Паттерны программирования находятся на более высоком уровне абстракции, чем *идиомы*, потому что их концепции не являются специфическими для языка

(или семейства языков). Другими словами, паттерн программирования — это не “устойчивое сочетание слов” (т. е. идиома) в конкретном языке программирования. Он представляет собой общее решение задачи, которая может возникнуть во многих языках; это способ мышления, присущий программисту. Паттерны программирования находятся на более низком уровне абстракции, чем *паттерны проектирования* и *архитектурные стили*. Другими словами, использование паттерна программирования приводит к созданию небольшого фрагмента кода, который станет частью более крупного проекта, в то время как использование паттерна проектирования или архитектурного стиля приводит к созданию описания взаимодействия между равновесными сущностями в большой системе.

В отличие от книг по паттернам проектирования и архитектурным стилям, используемых в курсах продвинутого уровня, эта книга не является самостоятельным пособием. Напротив, она является дополнением к традиционным учебникам, используемым на вводных курсах по программированию. Традиционный учебник помогает в обучении “запоминанию”, “пониманию”, “анализу” и “оценке”. Эта книга помогает в обучении “применению” и “созданию”. Она предполагает, что читатель уже знает синтаксис наборов команд во фрагментах кода (который можно изучить по стандартному учебнику), и вместо этого фокусируется на процессе рассуждений, который приводит к появлению фрагментов программы.

Вопросы стиля

Обучение с помощью паттернов неизбежно предполагает оценку различных решений одной и той же задачи, и эти решения могут оцениваться по целому ряду различных критериев. В этой книге сделана попытка сосредоточиться на критериях, которые имеют отношение непосредственно к самому решению задачи, не обращая внимания на вопросы стиля. Например, метод определения максимума двух чисел может быть реализован тремя различными способами. В первом случае используются промежуточная переменная и ее инициализация по умолчанию:

```
public static int max(int a, int b) {
    int result;
    result = b;
    if (a > b) result = a;
    return result;
}
```

Во втором случае используется промежуточная переменная, но оба случая обрабатываются явно (т. е. есть блок `else` и блок `if`):

```
public static int max(int a, int b) {
    int result;
    if (a > b) result = a;
    else result = b;
    return result;
}
```

В третьем — несколько операторов возврата `return` и отсутствие необходимости в использовании промежуточной переменной:

```
public static int max(int a, int b) {
    if (a > b)    return a;
    else    return b;
}
```

Можно привести веские аргументы в пользу всех трех решений, а их относительные достоинства могут варьироваться в зависимости от опыта программиста. Однако различия здесь связаны больше с особенностью написания программы, чем с самим решением. То есть это чисто стилистические различия.

Аналогично, после того как мы написали метод `max()`, мы можем написать метод `min()` и также реализовать его несколькими различными способами. Например, можно утверждать, что следующая реализация предпочтительнее, поскольку она согласуется с реализацией метода `max()` (при условии, что был выбран третий способ реализации):

```
public static int min(int a, int b) {
    if (a < b)    return a;
    else    return b;
}
```

В качестве альтернативы можно утверждать, что следующая реализация предпочтительнее, так как в ней меньше дублирования кода:

```
public static int min(int a, int b) {
    return -max(-a, -b);
}
```

Опять же эти аргументы больше относятся именно к особенности написания программы (т. е. к стилю), а не к самому решению.

В качестве заключительного примера реализуем метод `clamp()`. Реализовать его также можно по-разному. Например, в той же стилистике, что и реализованные ранее методы `max()` и `min()`:

```
public static int clamp(int x, int lower, int upper) {
    if (x < lower) return lower;
    if (x > upper) return upper;
    return x;
}
```

...или этот метод можно реализовать, непосредственно используя написанные ранее методы. Например так:

```
public static int clamp(int x, int lower, int upper) {
    return max(lower, min(upper, x));
}
```

Повторимся, что в этой книге предпринята попытка избежать оценки различных решений с точки зрения стиля написания. Другими словами, в ней делается попытка сравнить решения, которые отличаются не только стилем.

Организация этой книги

Паттерны в этой книге организованы в соответствии с темами, которые рассматриваются в традиционных вводных учебниках по программированию.

В первой части рассматриваются решения задач, в которых используются только арифметические операторы (и смежные темы). Таким образом, понимание объявления переменных, операторов присваивания и арифметических операторов — это все, что потребуется для решения задач в части I. В части II рассматриваются решения задач, требующие применения логических операторов, операторов отношения, условий и методов. После этого в части III рассматриваются паттерны, требующие применения циклов, массивов и (рудиментарного) ввода/вывода. Далее в части IV рассматриваются задачи и решения, требующие использования элементов массива (особенно атрибута `length`) и массивов массивов (иногда называемых многомерными массивами), а в части V — задачи и решения, связанные с объектами `String`. Наконец, в части VI рассматриваются паттерны, связанные с использованием ссылок.

Во многих случаях паттерны дополняют друг друга. Поэтому, за некоторыми исключениями, части книги следует изучать по порядку. К сожалению, поскольку в разных вводных курсах/учебниках по программированию эти темы рассматриваются разрозненно, может потребоваться бегло ознакомиться с некоторыми главами, а затем позднее вернуться к ним для детального изучения. Например, в одних курсах/учебниках методы рассматриваются перед массивами (как в этой книге), а в других — в обратном порядке. Поэтому, возможно, придется перескочить и просмотреть некоторые паттерны, приведенные в части III, пока не будут рассмотрены более сложные методы.

Каждая глава содержит постановку задачи (т. е. ситуацию, в которой возникает рассматриваемая задача), паттерн (т. е. решение задачи) и примеры. Многие главы также содержат одно или несколько предупреждений, как правило, о том, что не следует делать чрезмерных обобщений. Некоторые главы также включают “взгляд на перспективу”, где паттерн может появиться в последующих курсах. Материал в этих разделах, как правило, более продвинутый и может быть опущен (т. е. не требуется для изучения последующих глав).

Благодарности

Моя жена пишет превосходно. Я — нет. Несмотря на все ее усилия сделать эту книгу читабельной, за которые я ей очень благодарен, она потерпела неудачу. Однако вина лежит на мне, так что, пожалуйста, не вините ее. (Я не могу назвать ее имени, потому что мы оба работаем на факультете JMU, и я не сообщаю своим студентам ее имя, чтобы они не посылали ей сочувственные записки).

Я также хотел бы отметить вклад двух моих коллег из JMU, Криса Мэйфилда и Криса Фокса. Они оба очень внимательно прочитали ранний вариант и предложили огромное количество правок, которые значительно улучшили текущую версию, за что я им очень благодарен.

Список рисунков