2.4 Choose method names carefully

We've already discussed how to name a class in Section 1.1. Now it's time to name methods properly. I'm suggesting this simple rule of thumb: builders are *nouns*, manipulators are *verbs*. Let's see what it means.

A method that builds something and returns a new object, I call a *builder*. Well, I just made this name up, but it looks logical to me. Builders always return something; they never return `void`, and their names are always nouns. For example:

```
int pow(int base, int power);
float speed();
Employee employee(int id);
String parsedCell(int x, int y);
```

Pay attention to that last method, `parsedCell()`. It's not just a noun, but a noun with an adjective in front of it. That doesn't change the principle; it only makes the name more descriptive. It's still a noun, but with more information about it. It's not just a cell, but a parsed cell. We should probably expect this method to return a cell that has transformed its contents somehow.

A method that makes modifications to the real-world entity being abstracted by the object, I call a *manipulator*. It always returns `void`, and its names are always verbs. For example:

```
void save(String content);
void put(String key, Float value);
void remove(Employee emp);
void quicklyPrint(int id);
```

Pay attention to the last method, `quicklyPrint()`. It's a verb with an adverb in front of it. The key element here is the verb "print", while "quickly" just explains it, giving us more information about the context and the purpose of the method.

Again, I made these builder and manipulator names up just for the sake of this chapter. You may call them something else, but try to keep this principle intact: Builders build and manipulators manipulate. And there is nothing in between. There should not be any methods that manipulate and return something, nor build and manipulate at the same time. Let me give a few bad examples:

```
// returns total bytes saved
int save(String content);
// returns TRUE if map was modified
boolean put(String key, Float value);
// saves speed and returns previous value
float speed(float val);
```

We'll discuss "setters" and "getters" later, in Section 3.5, but here it's already obvious that names that start with `get` are just wrong. That's because "get" is a verb but getters are basically builders, since they are supposed to return something. So, this is my first argument against "getters".

Now, I think I owe you an explanation for this idea. There are a few arguments in its favor.

2.4.1 Builders are nouns

First, I believe that it's wrong to name a method as a verb if it returns something. Such a name runs against the idea of object

thinking. When I stop in at a bakery, I don't say "cook me a brownie" or "brew me a cup of coffee". I'd say, "I'd like a brownie" or "I'd like to have a cup of coffee". If I said "cook me" or "brew me" something, it would sound rather offensive. I should not care how exactly that brownie is made or that cup of coffee is brewed. It's their business how to make them. I have demand for an object — a brownie or a cup of coffee. They can satisfy my demand. How exactly this happens inside the bakery is none of my business. Here is the bakery:

```
class Bakery {
  Food cookBrownie();
  Drink brewCupOfCoffee(String flavor);
}
```

These two methods are not actually methods of an object. They are *procedures*. Their naming tells us that we ought to pay no respect to the bakery as a self-sufficient and self-managed object, and just tell it what to do for us. This is a procedural approach, not an object-oriented one. This is how these two procedures would be designed in C, for example:

```
Food* cook_brownie() {
  // cook that brownie
  // and return it
}
Drink* brew_cup_of_coffee(char* flavor) {
  // brew a cup of coffee
  // and return it
}
```

There is no bakery involved. We just have two pieces of machine instructions in C syntax, and we call them. We call

them functions in C, but they are actually procedures because they have very little to do with functional programming as well. We ask the computer to run those instructions for us and return the result. We are thinking like a computer, not like an object. We don't trust the bakery, so we tell it to "go and brew the damn coffee" instead of asking for a cup of coffee with a certain flavor and then just entrusting the establishment with the result, no matter what it is.

I don't want to sound too philosophical, but this naming subject is indeed very abstract and conceptual. A properly named method helps its users better understand what the object is designed for, what his mission is, what the purpose of his existence is, and what the meaning of life is for him, while an improper method name may ruin the entire idea of an object and encourage its users to treat him as a bag of data and a collection of procedures. It's a very typical mistake that is repeatedly made by OOP libraries, SDKs, APIs, etc. An object is a *living organism* who knows how to perform his duties and wants to be respected. He wants to work by the contract, not just follow instructions. There is a big difference.

That's why, when the name of the method is a verb, it's basically telling the object "what to do". And asking an object to "build" something is not a polite and respectful way to work with him. Just request what you need built, and let him decide how to build it. All of these names are wrong:

```
InputStream load(URL url);
String read(File file);
int add(int x, int y);
```

They should be replaced with:

```
InputStream stream(URL url);
String content(File file);
int sum(int x, int y);
```

Pay attention to the fact that instead of `add(x,y)`, I'm suggesting you use `sum(x,y)`. It may look like a small and unimportant change, but it really makes a big difference in your thinking. We don't ask our object to add x to y. Instead, we ask him to produce the sum of the two and return a new object. Will he really find the sum? I don't know. Maybe. All I know is that the result will look like the sum of x and y. Again, I'm not telling my object what to do, I'm just asking for a result that must obey a certain contract — be an integer number.

This is the first argument and the first use case. We are getting something from an object, or in other words, asking an object to build something for us. Now, let's discuss the second argument and the second use case, when we ask the object to do some manipulation for us.

2.4.2 Manipulators are verbs

As you remember, an object is a representative of a real-world entity. An object of class `File` represents a file on disk, an object of class `Pixel` represents a pixel on the screen, and an instance of class `Integer` represents four bytes of RAM (Surprised? We'll discuss that in detail in Section 3.4).

When we need to manipulate a real-world entity, we ask the object to do it for us. For example:

```
class Pixel {
  void paint(Color color);
}
Pixel center = new Pixel(50, 50);
center.paint(new Color("red"));
```

We ask object `center` to paint a pixel on the screen, located at the `50x50` coordinates. We don't expect anything to be built; we just want to make a modification to the world, and the object is a representative of it for us. Now, you may ask how this is not a procedure. It is named as a verb, and it basically directs an object to do something for us. Yes, it's a valid question, but the key difference is the result returned.

The method `paint()` doesn't return a result. Using the same bakery metaphor, this is similar to asking a bartender to turn up the music. Will she make it louder? Maybe she will. Maybe not. Our request may just be ignored. It's never offensive or disrespectful, because we are not expecting anything back. Imagine how this would sound otherwise — "Please turn up the music and tell me its volume level when you're done". That's exactly how a manipulator that returns a value looks. Very disrespectful.

Thus, the difference is in the return value. Only a builder is allowed to return a value, and its name must be a noun. When an object allows us to manipulate, the name has to be a verb, and there must be no return value.

I think it's possible to follow a less strict naming convention, provided you keep the main principle in mind. For example, when using a Builder Pattern, you can start method names with a `with` prefix:

```
class Book {
  Book withAuthor(String author);
  Book withTitle(String title);
  Book withPage(Page page);
}
```

Here, the name `withTitle` is just a short form of `bookWithTitle`. In order to avoid this `book` prefix in all methods, we can use just the `with` prefix. But the principle is still in place — these methods are builders, and their names are classified as nouns.