Item 56: Adhere to generally accepted naming conventions

The Java platform has a well-established set of naming conventions, many of which are contained in *The Java Language Specification* [JLS, 6.8]. Loosely speaking, naming conventions fall into two categories: typographical and grammatical.

There are only a handful of typographical naming conventions, covering packages, classes, interfaces, methods, fields, and type variables. You should rarely violate them and never without a very good reason. If an API violates these conventions, it may be difficult to use. If an implementation violates them, it may be difficult to maintain. In both cases, violations have the potential to confuse and irritate other programmers who work with the code and can cause faulty assumptions that lead to errors. The conventions are summarized in this item.

Package names should be hierarchical with the components separated by periods. Components should consist of lowercase alphabetic characters and, rarely, digits. The name of any package that will be used outside your organization should begin with your organization’s Internet domain name with the top-level domain first, for example, edu.cmu, com.sun, gov.nsa. The standard libraries and optional packages, whose names begin with java and javax, are exceptions to this rule. Users must not create packages whose names begin with java or javax. Detailed rules for converting Internet domain names to package name prefixes can be found in *The Java Language Specification* [JLS, 7.7].

The remainder of a package name should consist of one or more components describing the package. Components should be short, generally eight or fewer characters. Meaningful abbreviations are encouraged, for example, util rather than utilities. Acronyms are acceptable, for example, awt. Components should generally consist of a single word or abbreviation.

Many packages have names with just one component in addition to the Internet domain name. Additional components are appropriate for large facilities whose size demands that they be broken up into an informal hierarchy. For example, the javax.swing package has a rich hierarchy of packages with names such as javax.swing.plaf.metal. Such packages are known as subpackages, although there is no linguistic support for package hierarchies.

Class and interface names, including enum and annotation type names, should consist of one or more words, with the first letter of each word capitalized, for example, Timer or FutureTask. Abbreviations are to be avoided, except for acronyms and certain common abbreviations like max and min. There is little consensus as to whether acronyms should be uppercase or have only their first letter capitalized. While uppercase may be more common, a strong argument can be
made in favor of capitalizing only the first letter: even if multiple acronyms occur back-to-back, you can still tell where one word starts and the next word ends. Which class name would you rather see, HTTPURL or HttpUrl?

Method and field names follow the same typographical conventions as class and interface names, except that the first letter of a method or field name should be lowercase, for example, remove or ensureCapacity. If an acronym occurs as the first word of a method or field name, it should be lowercase.

The sole exception to the previous rule concerns “constant fields,” whose names should consist of one or more uppercase words separated by the underscore character, for example, VALUES or NEGATIVE_INFINITY. A constant field is a static final field whose value is immutable. If a static final field has a primitive type or an immutable reference type (Item 15), then it is a constant field. For example, enum constants are constant fields. If a static final field has a mutable reference type, it can still be a constant field if the referenced object is immutable. Note that constant fields constitute the only recommended use of underscores.

Local variable names have similar typographical naming conventions to member names, except that abbreviations are permitted, as are individual characters and short sequences of characters whose meaning depends on the context in which the local variable occurs, for example, i, xref, houseNumber.

Type parameter names usually consist of a single letter. Most commonly it is one of these five: T for an arbitrary type, E for the element type of a collection, K and V for the key and value types of a map, and X for an exception. A sequence of arbitrary types can be T, U, V or T1, T2, T3.

For quick reference, the following table shows examples of typographical conventions.

<table>
<thead>
<tr>
<th>Identifier Type</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package</td>
<td>com.google.inject, org.joda.time.format</td>
</tr>
<tr>
<td>Class or Interface</td>
<td>Timer, FutureTask, LinkedHashMap, HttpServlet</td>
</tr>
<tr>
<td>Method or Field</td>
<td>remove, ensureCapacity, getCrc</td>
</tr>
<tr>
<td>Constant Field</td>
<td>MIN_VALUE, NEGATIVE_INFINITY</td>
</tr>
<tr>
<td>Local Variable</td>
<td>i, xref, houseNumber</td>
</tr>
<tr>
<td>Type Parameter</td>
<td>T, E, K, V, X, T1, T2</td>
</tr>
</tbody>
</table>
Grammatical naming conventions are more flexible and more controversial than typographical conventions. There are no grammatical naming conventions to speak of for packages. Classes, including enum types, are generally named with a singular noun or noun phrase, for example, Timer, BufferedWriter, or ChessPiece. Interfaces are named like classes, for example, Collection or Comparator, or with an adjective ending in able or ible, for example, Runnable, Iterable, or Accessible. Because annotation types have so many uses, no part of speech predominates. Nouns, verbs, prepositions, and adjectives are all common, for example, BindingAnnotation, Inject, ImplementedBy, or Singleton.

Methods that perform some action are generally named with a verb or verb phrase (including object), for example, append or drawImage. Methods that return a boolean value usually have names that begin with the word is or, less commonly, has, followed by a noun, noun phrase, or any word or phrase that functions as an adjective, for example, isDigit, isProbablePrime, isEmpty, isEnabled, or hasSiblings.

Methods that return a non-boolean function or attribute of the object on which they're invoked are usually named with a noun, a noun phrase, or a verb phrase beginning with the verb get, for example, size, hashCode, or getTime. There is a vocal contingent that claims that only the third form (beginning with get) is acceptable, but there is little basis for this claim. The first two forms usually lead to more readable code, for example:

```java
if (car.speed() > 2 * SPEED_LIMIT)
    generateAudibleAlert("Watch out for cops!");
```

The form beginning with get is mandatory if the class containing the method is a Bean [JavaBeans], and it's advisable if you're considering turning the class into a Bean at a later time. Also, there is strong precedent for this form if the class contains a method to set the same attribute. In this case, the two methods should be named getAttribute and setAttribute.

A few method names deserve special mention. Methods that convert the type of an object, returning an independent object of a different type, are often called toType, for example, toString, toArray. Methods that return a view (Item 5) whose type differs from that of the receiving object are often called asType, for example, asList. Methods that return a primitive with the same value as the object on which they're invoked are often called typeValue, for example, intValue. Common names for static factories are valueOf, of, getInstance, newInstance, getType, and newType (Item 1, page 10).
Grammatical conventions for field names are less well established and less important than those for class, interface, and method names, as well-designed APIs contain few if any exposed fields. Fields of type boolean are often named like boolean accessor methods with the initial is omitted, for example, initialized, composite. Fields of other types are usually named with nouns or noun phrases, such as height, digits, or bodyStyle. Grammatical conventions for local variables are similar to those for fields, but even weaker.

To summarize, internalize the standard naming conventions and learn to use them as second nature. The typographical conventions are straightforward and largely unambiguous; the grammatical conventions are more complex and looser. To quote from *The Java Language Specification* [JLS, 6.8], “These conventions should not be followed slavishly if long-held conventional usage dictates otherwise.” Use common sense.