Naming Conventions Cheat Sheet

Contents

Overview

In the interests of keeping your code maintainable, either for yourself or for other developers, Java adheres to several *naming conventions*.

Although companies will differ in terms of how code should be organised, they usually tend to agree with each other on these conventions.

This is commonly known as *good practice*, and it is vital to modern software development.

Cases

You may have seen references to several different types of *cases* when writing code. These form the basis of naming conventions in software engineering. The most common are:

- `PascalCase` / `UpperCamelCase` Where each new word has a capital letter to start.
- `camelCase` / `lowerCamelCase` The same as PascalCase except the first word is always all lower case.
- `snake_case` All lower case seperated by underscores `_`
- `UPPERCASE` All uppercase.
- `lowercase` All lowercase.

General rules for all identifiers

It probably comes as no surprise that Java has precise rules about *identifiers* (names for all the things you write yourself).

Luckily, the same general rules apply to anything you are free to name, including variables, classes, methods, and fields.

There are only three rules to remember for legal identifiers:

- the name must begin with a *standard alphabetical character* (a-z/A-Z), a `$`, or `_`.
- subsequent characters may be entirely *alphanumeric* (a-z/A-Z/0-9)

- you cannot use the same name as a *reserved word* in Java (there are lots of these, so it'll probably happen to you by accident at some point!)

Classes

Classes in Java should be *nouns*, written in `PascalCase`:

```
public class ImageSprite{ }
```

Test classes

Classes used for testing should use the name of the class it is testing, preceded or followed by the word `Test`:

```
public class ImageSpriteTest{ }
```

Interfaces (and implementations)

Interfaces should be *adjectives* written in `PascalCase` - if they act as a listener or a service, name them accordingly:

```
public interface Drawable { }
public interface ActionListener { }
public interface DrawService { }
```

It is generally accepted that the *first implementation* of an Interface should use the Interface's name, followed by the suffix `Impl`:

```
public class DrawableImpl implements Drawable { }
public class ActionListenerImpl implements ActionListener { }
public class DrawServiceImpl implements DrawService { }
```

In practice, if there is a more obvious name to describe what your implementation is doing, then it is probably best to use that instead.

Methods

Methods should be *verbs*, written in `camelCase`:

```
public void draw(){ }
public double getPerimeter(){ }
```

Variables

Local, instance and class variables should be written in `camelCase`, and should ideally *not* start with `_` or `$`.

Variable names should be short, but *meaningful* - if you come back to that variable and are unsure exactly what it does, then you haven't named it properly.

Temporary variables

One-character variable names should be avoided *except* for temporary variables used in indexing, iteration, etc.

Common names for temporary variables are:

- `i`, `j`, `k`, `m`, and `n` for `int`s
- `c`, `d`, and `e` for `char`s

```
for (int i : items){
    System.out.println(i);
}
```

Constants & Enums

Constants and Enums should be written in `UNDERSCORE_SEPARATED_UPPERCASE`:

```
static final int EARTH_RADIUS_KM = 6731;

enum Direction { NORTH, SOUTH, EAST, WEST }
```

Generic Types

Generic type parameter names should be `UPPERCASE` single characters.

- generally, `T` (for `Type`) is recommended
- `E` (for `Element`) is recommended for Collections
- `K` (for `Key`) & `V` (for `Value`) is recommended for maps
- `S` (for `Service`) is recommended for service loaders

Exceptions

Generally, Exceptions should be denoted by using the word `Exception`.

There is no specific convention other than this, though it may be useful to use `Invalid` or `Illegal` as modifiers for Exception names:

```
public class InvalidDirectionException extends Exception { }
```

Packages

Packages should be named in `period.separated.lowercase` format.

They should begin with a top-level domain name, followed by the organisation name, before adhering to whichever naming conventions are present within that organisation:

```
package com.qa.main;
```

(One of the reasons for using packages is so that you don't have to use unique Class names across all of your application, which can occasionally be useful.)

Acronyms

There is no hard-wired convention for whether or not to use acronyms, or how to name them. Thus, all the following are valid:

```
public class XML() { }
public class ExtensibleMarkupLanguge(){ }

public void convertPDFToXML() { }
public void convertPdfToXml() { }
public void convertPortableDocumentFormatToExtensibleMarkupLanguage() { }
```

Tutorial

There is no tutorial for this module.

Exercises

Check out FizzBuzz Enterprise Edition

A tongue-in-cheek attempt to write a basic FizzBuzz program in Java using enterprise-level development practices is on GitHub here.

Don't worry if it's all completely confusing to you - it's designed to be.

The important thing to take note of is how they've structured their code and named their identifiers.

Think about how you can apply this to some of the code that you may have already written, then refactor one of your own projects to complete this exercise.