extraídos de dicho dominio.

Añadir contexto con sentido

Algunos nombres tienen significado por sí mismos, pero la mayoría no. Por ello, debe incluirlos en un contexto, en clases, funciones y espacios de nombres con nombres adecuados. Cuando todo lo demás falle, pueden usarse prefijos como último recurso.

Imagine que tiene las variables `firstName`, `lastName`, `street`, `houseNumber`, `city`, `state` y `zipcode`. Si las combina, es evidente que forman una dirección. Pero si la variable `state` se usa de forma aislada en un método, ¿sabría que forma parte de una dirección? Puede añadir contexto por medio de prefijos: `addrFirstName`, `addrLastName`, `addrState`, etc. Al menos los lectores comprenderán que estas variables forman parte de una estructura mayor. Evidentemente, es mejor crear la clase `Address`. De ese modo, incluso el compilador sabrá que las variables pertenecen a un concepto más amplio.

Fíjese en el método del Listado 2-1. ¿Las variables necesitan un contexto con más sentido? El nombre de la función sólo ofrece parte del contexto, el resto se obtiene del algoritmo. Tras leer la función, verá que las tres variables `number`, `verb` y `pluralModifier` forman parte del mensaje `guess statistics`. Desafortunadamente, es necesario inferir el contexto. Al leer el método, el significado de las variables no es evidente.

Listado 2-1
Variables en un contexto ambiguo.

```
private void printGuessStatistics(char candidate, int count) {
  String number;
  String verb;
  String pluralModifier;
  if (count == 0) {
    number = “no”;
    verb = “are”;
    pluralModifier = “s”;
  } else if (count == 1) {
    number = “1”;
```

```
      verb = “is”;
      pluralModifier = “”;
    } else {
      number = Integer.toString(count);
      verb = “are”;
      pluralModifier = “s”;
    }
    String guessMessage = String.format(
      “There %s %s %s%s”, verb, number, candidate, pluralModifier
    );
    print(guessMessage);
  }
```

La función es demasiado extensa y las variables aparecen por todas partes. Para dividir la función en fragmentos más reducidos necesitamos crear una clase `GuessStatisticsMessage` y convertir a las tres variables en campos de la misma. De este modo contamos con un contexto más obvio para las tres variables. Forman parte sin duda de `GuessStatisticsMessage`. La mejora del contexto también permite que el algoritmo sea más limpio y se divida en funciones más reducidas (véase el Listado 2-2).

Listado 2-2
Variables con un contexto.

```
public class GuessStatisticsMessage (
  private String number;
  private String verb;
  private String pluralModifier;

  public String make(char candidate, int count) {
  createPluralDependentMessageParts(count);
  return String.format(
    “There %s %s %s%s,
    verb, number, candidate, pluralModifier);
  }

  private void createPluralDependentMessageParts(int count) {
    if (count == 0) {
      thereAreNoLetters();
    } else if (cout == 1) {
      thereIsOneLetter();
    } else {
      thereAreManyLetters(count);
    }
  }

  private void thereAreManyLetters(int count) {
    number = “1”;
```

```
      verb = “is”;
      pluralModifier = “”;
    }

    private void thereIsOneLetter() {
      number = “1”;
      verb = “is”;
      pluralModifier = “”;
    }

    private void thereAreNoLetters() {
      number = “no”;
      verb = “are”;
      pluralModifier = “s”;
    }
  }
```

No añadir contextos innecesarios

En la aplicación imaginaria Gas Station Deluxe, no es aconsejable usar el prefijo `GSD` en todas las clases. Es trabajar contra las herramientas proporcionadas. Introduzca `G` y pulse la tecla de finalización para acceder a una lista interminable de todas las clases del sistema. ¿Es lo correcto? ¿Por qué dificultar la ayuda del IDE?

Del mismo modo, imagine que ha creado la clase `MailingAddress` en un módulo de contabilidad de `GSD`, con el nombre `GSDAccountAddress`. Después, necesita una dirección de correo para la aplicación de contacto con el cliente. ¿Usará `GSDAccountAddress`? ¿Le parece el nombre correcto? 10 de los 17 caracteres son redundantes o irrelevantes.

Los nombres breves suelen ser más adecuados que los extensos, siempre que sean claros. No añada más contexto del necesario a un nombre. Los nombres `accountAddress` y `customerAddress` son perfectos para instancias de la clase `Address` pero no sirven como nombres de clase. `Address` sirve como nombre de clase. Para distinguir entre direcciones MAC, direcciones de puertos y direcciones Web, podría usar `PostalAddress`, `MAC` y `URI`. Los nombres resultantes son más precisos, el objetivo de cualquier nombre.