

Writing methods

1. Writing methods

Generating a method is supported through various shortcuts. These shortcuts and their value are probably best demonstrated by examples.

2. Throwing exceptions

Throwing exceptions using the standard method `addLine()` works roughly like this:

```
public JavaMethod getDifficultMethod(JavaSource pSource) {
    JavaMethod jm = pSource.newJavaMethod("difficultMethod", "void", "public");
    jm.addLine("throw new (" + NotImplementedException.class, "(",
              JavaSource.getQuoted("The method 'difficultMethod' is not yet implemented
              ");");");
    return jm;
}
```

Using the method `addThrowNew()`, this could be rewritten as follows:

```
public JavaMethod getDifficultMethod(JavaSource pSource) {
    JavaMethod jm = pSource.newJavaMethod("difficultMethod", "void", "public");
    jm.addThrowNew(NotImplementedException.class,
                  JavaSource.getQuoted("The method 'difficultMethod' is not yet implemented
    return jm;
}
```

3. If .. elseif .. else .. blocks

Suggest the following example:

```
public JavaMethod getValueOfMethod(JavaSource pSource) {
    JavaMethod jm = pSource.newJavaMethod("valueOf", "int", "public");
    jm.addParam(String.class, "s");
    jm.addLine("if (" + JavaSource.getQuoted("FOO"), ".equals(s)) {");
    jm.indent();
    jm.addLine("return foo;");
    jm.unindent();
    jm.addLine("} else if (" + JavaSource.getQuoted("BAR"), ".equals(s)) {");
    jm.indent();
    jm.addLine("return bar;");
}
```

```

jm.unindent();
jm.addLine("} else {");
jm.indent();
jm.addThrowNew(IllegalArgumentException.class,
    JavaSource.getQuoted("Invalid value for s: "), " + s");
jm.unindent();
jm.addLine("}");
return jm;
}

```

This example could also be written like this:

```

public JavaMethod getValueOfMethod(JavaSource pSource) {
    JavaMethod jm = pSource.newJavaMethod("valueOf", "int", "public");
    jm.addParam(String.class, "s");
    jm.addIf(JavaSource.getQuoted("FOO"), ".equals(s)");
    jm.addLine("return foo;");
    jm.addElseIf(JavaSource.getQuoted("BAR"), ".equals(s)");
    jm.addLine("return bar;");
    jm.addElse();
    jm.addThrowNew(IllegalArgumentException.class,
        JavaSource.getQuoted("Invalid value for s: "), " + s");
    jm.addEndIf();
    return jm;
}

```

The rewritten example is both shorter and more readable.

4. Try .. catch .. finally .. blocks

A try .. catch block is typically written like this:

```

public JavaMethod getAsIntMethod(JavaSource pSource) {
    JavaMethod jm = pSource.newJavaMethod("asInt", "int", "public");
    jm.addParam(String.class, "s");
    jm.addLine("try {");
    jm.indent();
    jm.addLine("return Integer.toString(s);");
    jm.unindent();
    jm.addLine("} catch (" + NumberFormatException.class, " e) {");
    jm.indent();
    jm.addLine("e.printStackTrace();");
    jm.addLine("return 1;");
    jm.unindent();
    jm.addLine("}");
    return jm;
}

```

To simplify the example, use the `addTry()` and `addCatch()` methods:

Writing methods

```
public JavaMethod getAsIntMethod(JavaSource pSource) {
    JavaMethod jm = pSource.newJavaMethod("asInt", "int", "public");
    jm.addParam(String.class, "s");
    jm.addTry();
    jm.addLine("return Integer.toString(s);");
    jm.addCatch(NumberFormatException.class, "e");
    jm.addLine("e.printStackTrace();");
    jm.addLine("return 1;");
    jm.addEndTry();
    return jm;
}
```

An even shorter version would be to replace

```
jm.addCatch(NumberFormatException.class, "e");
```

with

```
jm.addCatch(NumberFormatException.class);
```

The method version with a single parameter generates a local variable name, which is returned as a result by the `addCatch()` method.