

# Outils de formatage et d'analyse de code en Java

(une traduction rapide de la page de documentation officielle VSCode sur les [outils Java](#))

L'extension [Language Support for Java™ by Red Hat](#) fournit des [paramètres de formatage](#). Vous pouvez exporter un fichier de formatage Eclipse et l'utiliser ensuite pour votre projet dans VSCode.

En outre, il existe également les extensions [Checkstyle for Java](#) et [SonarLint](#), qui offrent des fonctionnalités de linter et d'analyse de code à la volée.

## 1 Le principe du Linting

Le `linting` ("linter" son code) est une pratique qui vise à améliorer la qualité de votre code et de ce fait la reprise et la maintenabilité de celui-ci. Des `linteurs` existent dans quasiment tous les langages.

Au départ `lint` était un logiciel qui faisait une analyse statique de code C pour en détecter les erreurs récurrentes, les erreurs d'indentation et les syntaxes spécifiques à un OS rendant le code non-portable. Il a ensuite été intégré dans les compilateurs C.

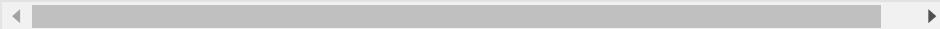
Dans un IDE vous avez donc un `linteur` auquel on ajoute un `débugueur`, un `profileur` et tout un tas d'autres outils pour aider le développeur dans sa quête du code parfait.

## 2 Linting vs Debugging

Le `linteur` va analyser le code de façon `statique` et le `debugueur` – comme beaucoup d'autres outils – le fait de façon `dynamique`. Cela veut dire que le `debugueur` va analyser le code lors de son exécution alors que le `linteur` va s'attarder sur les fichiers sources du projet pendant l'écriture des lignes de code (le code n'est pas exécuté).

Par exemple, le linteur détectera :

- les variables qui n'existent pas
- les variables inutilisées
- les doubles déclarations de variables, de fonctions, etc..
- la mauvaise organisation du code
- le non respect des bonnes pratiques d'écriture de code
- les erreurs de syntaxe



Le debugueur détectera quand à lui :

- les mauvais accès aux adresses mémoires
- le nom respect des types de variable objet
- le mauvais état d'une variable à un instant donné
- les fuites de mémoire
- les dépassements de pile

## 3 Formatage

Vous pouvez utiliser la commande **Format Document** pour formater un fichier Java. Si vous n'avez pas spécifié de profil de formatage auparavant, le fichier Java sera formaté en utilisant les paramètres par défaut.

```

File Edit Selection View Go Run Terminal Help
OwnerController.java - spring-petclinic - Visual Studio Code

OwnerController.java ×
src > main > java > org > springframework > samples > petclinic > owner > OwnerController.java > OwnerController > processFindForm
90
91     // allow parameterless GET request for /owners to return all records
92     if (owner.getLastName() == null) {
93         owner.setLastName(""); // empty string signifies broadest possible search
94     }
95
96     // find owners by last name
97     String lastName = owner.getLastName();
98     Page<Owner> ownersResults = findPaginatedForOwnersLastName(page, lastName);
99     if (ownersResults.isEmpty()) {
100         // no owners found
101         result.rejectValue("lastName", "notFound", "not found");
102         return "owners/findOwners";
103     }
104     else if (ownersResults.getTotalElements() == 1) {
105         // 1 owner found
106         owner = ownersResults.iterator().next();
107         return "redirect:/owners/" + owner.getId();
108     }
109     else {
110         // multiple owners found
111         lastName = owner.getLastName();
112         return addPaginationModel(page, model, lastName, ownersResults);
113     }
114 }
115
116 private String addPaginationModel(int page, Model model, String lastName, Page<Owner> paginated) {
117     model.addAttribute("listOwners", paginated);
118     List<Owner> listOwners = paginated.getContent();
119     model.addAttribute("currentPage", page);
120     model.addAttribute("totalPages", paginated.getTotalPages());
121     model.addAttribute("totalItems", paginated.getTotalElements());
122     model.addAttribute("listOwners", listOwners);
123     return "owners/ownersList";

```

### 3.1 Application des paramètres de formatage

Vous pouvez facilement appliquer les paramètres de formatage à partir d'un profil de formatage existant. Par exemple, si vous voulez appliquer [Google Style](#) pour votre projet Java, vous pouvez définir la propriété suivante dans `settings.json` :

```
"java.format.settings.url": "https://raw.githubusercontent.com/g
```

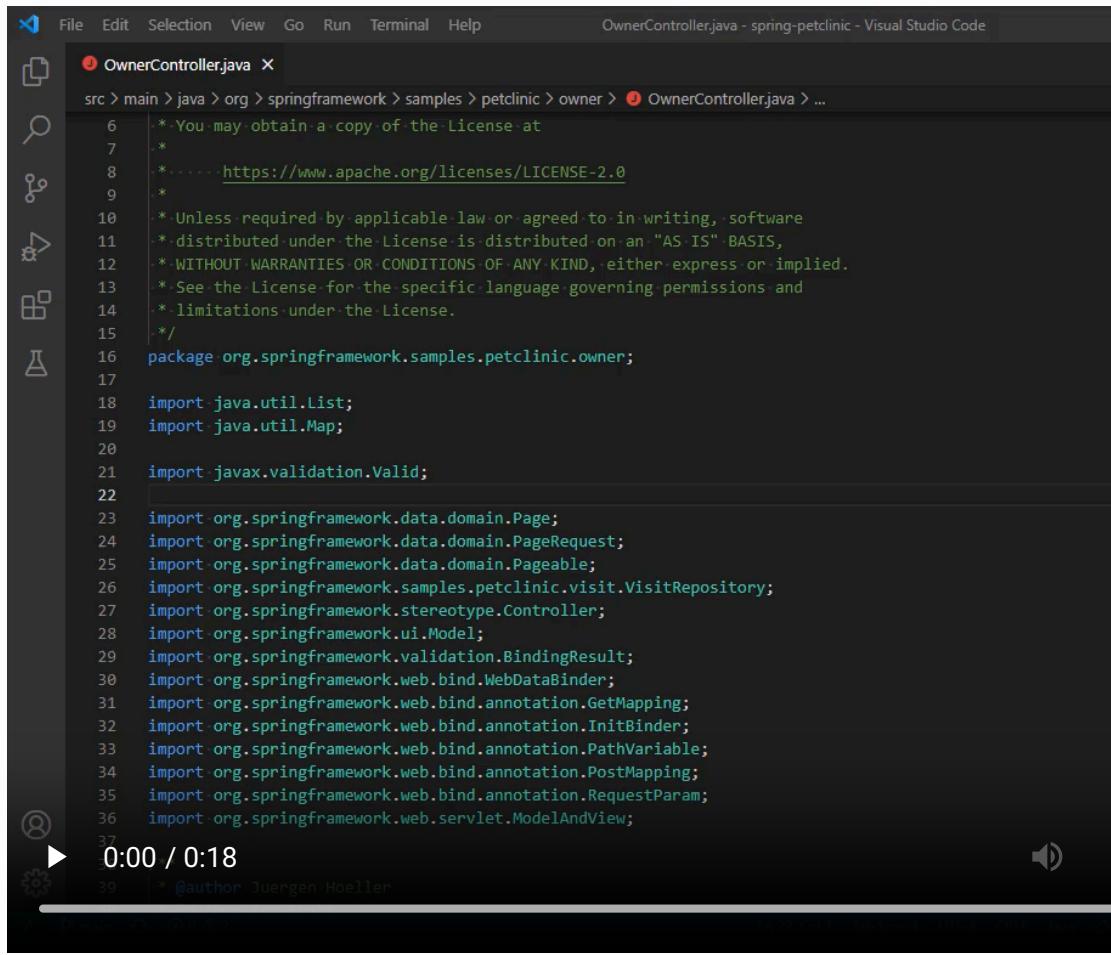
La propriété peut être définie comme une URL ou un chemin d'accès à un fichier local. Si le fichier XML de formatage contient plus d'un profil, vous pouvez spécifier le nom du profil :

```
"java.format.settings.profile": "GoogleStyle",
```

Après avoir défini le profil de formatage, la commande **Format Document** utilisera le profil spécifique pour formater vos fichiers Java.

## 3.2 Modification des paramètres de formatage

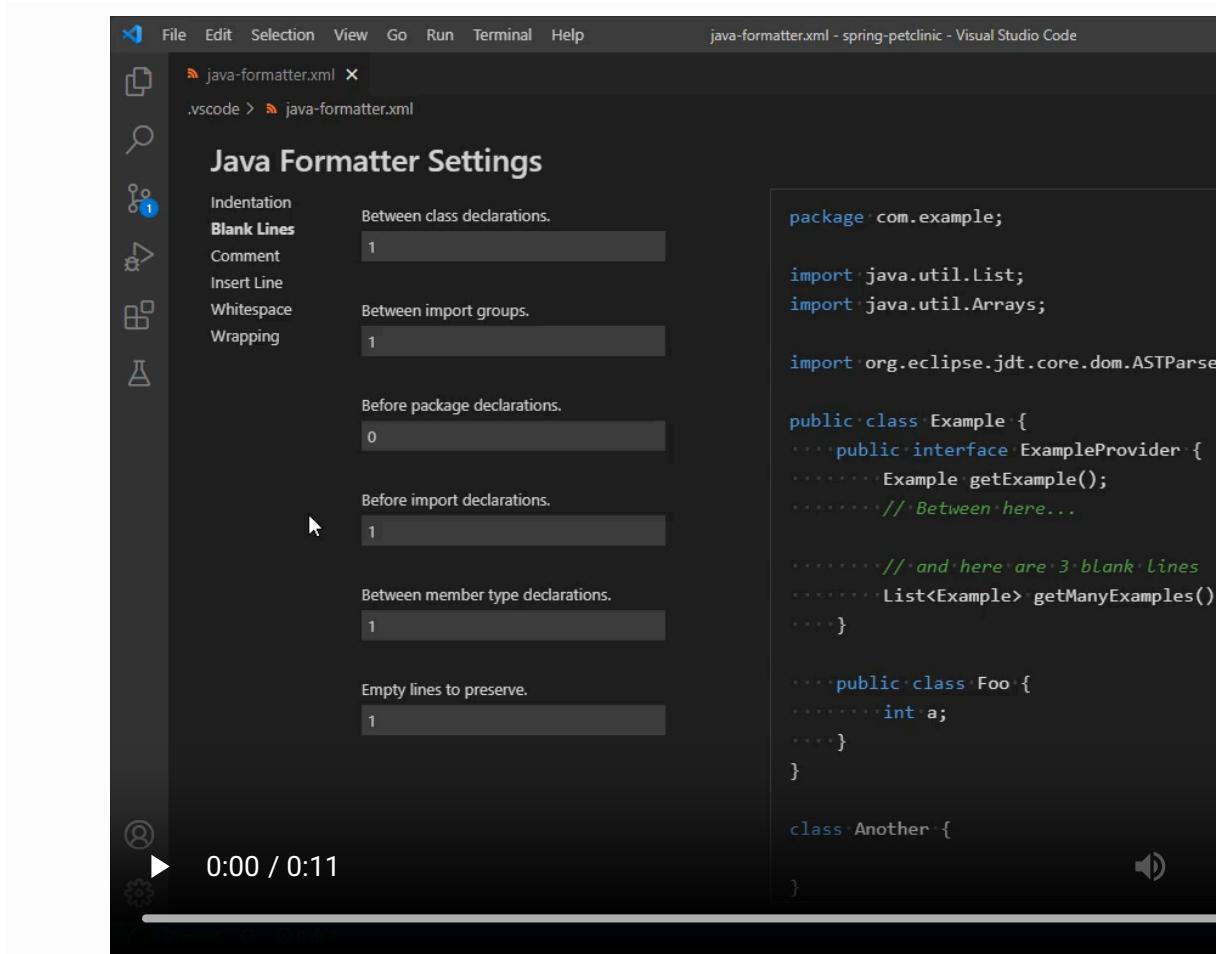
Le [Extension Pack for Java](#) fournit un éditeur pour aider les utilisateurs à modifier un profil de formatage existant. Vous pouvez ouvrir l'éditeur avec la commande **Java: Open Java Formatter Settings with Preview**. Dans l'éditeur, vous pouvez modifier les paramètres de formatage et prévisualiser les effets. Après avoir enregistré l'éditeur actuel, les modifications seront enregistrées dans le profil de formatage.



The screenshot shows the Visual Studio Code interface with the Java Formatter Settings preview panel open. The title bar reads "OwnerController.java - spring-petclinic - Visual Studio Code". The preview panel displays the Java code for "OwnerController.java" with various syntax highlighting and annotations. On the right side of the preview panel, there is a "Preview" section containing a play button icon, a progress bar showing "0:00 / 0:18", and a volume control icon. The main code area shows imports from org.springframework and org.springframework.samples.petclinic, along with annotations like @Valid and @RequestMapping.

```
6  * You may obtain a copy of the License at
7  *
8  *     https://www.apache.org/licenses/LICENSE-2.0
9  *
10 * Unless required by applicable law or agreed to in writing, software
11 * distributed under the License is distributed on an "AS IS" BASIS,
12 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13 * See the License for the specific language governing permissions and
14 * limitations under the License.
15 */
16 package org.springframework.samples.petclinic.owner;
17
18 import java.util.List;
19 import java.util.Map;
20
21 import javax.validation.Valid;
22
23 import org.springframework.data.domain.Page;
24 import org.springframework.data.domain.PageRequest;
25 import org.springframework.data.domain.Pageable;
26 import org.springframework.samples.petclinic.visit.VisitRepository;
27 import org.springframework.stereotype.Controller;
28 import org.springframework.ui.Model;
29 import org.springframework.validation.BindingResult;
30 import org.springframework.web.bind.WebDataBinder;
31 import org.springframework.web.bind.annotation.GetMapping;
32 import org.springframework.web.bind.annotation.InitBinder;
33 import org.springframework.web.bind.annotation.PathVariable;
34 import org.springframework.web.bind.annotation.PostMapping;
35 import org.springframework.web.bind.annotation.RequestParam;
36 import org.springframework.web.servlet.ModelAndView;
37
38 /**
39  * @author Juergen Hoeller
```

Lorsque vous modifiez les paramètres dans l'éditeur, vous pouvez prévisualiser les effets des modifications dans le panneau de droite **Preview**.

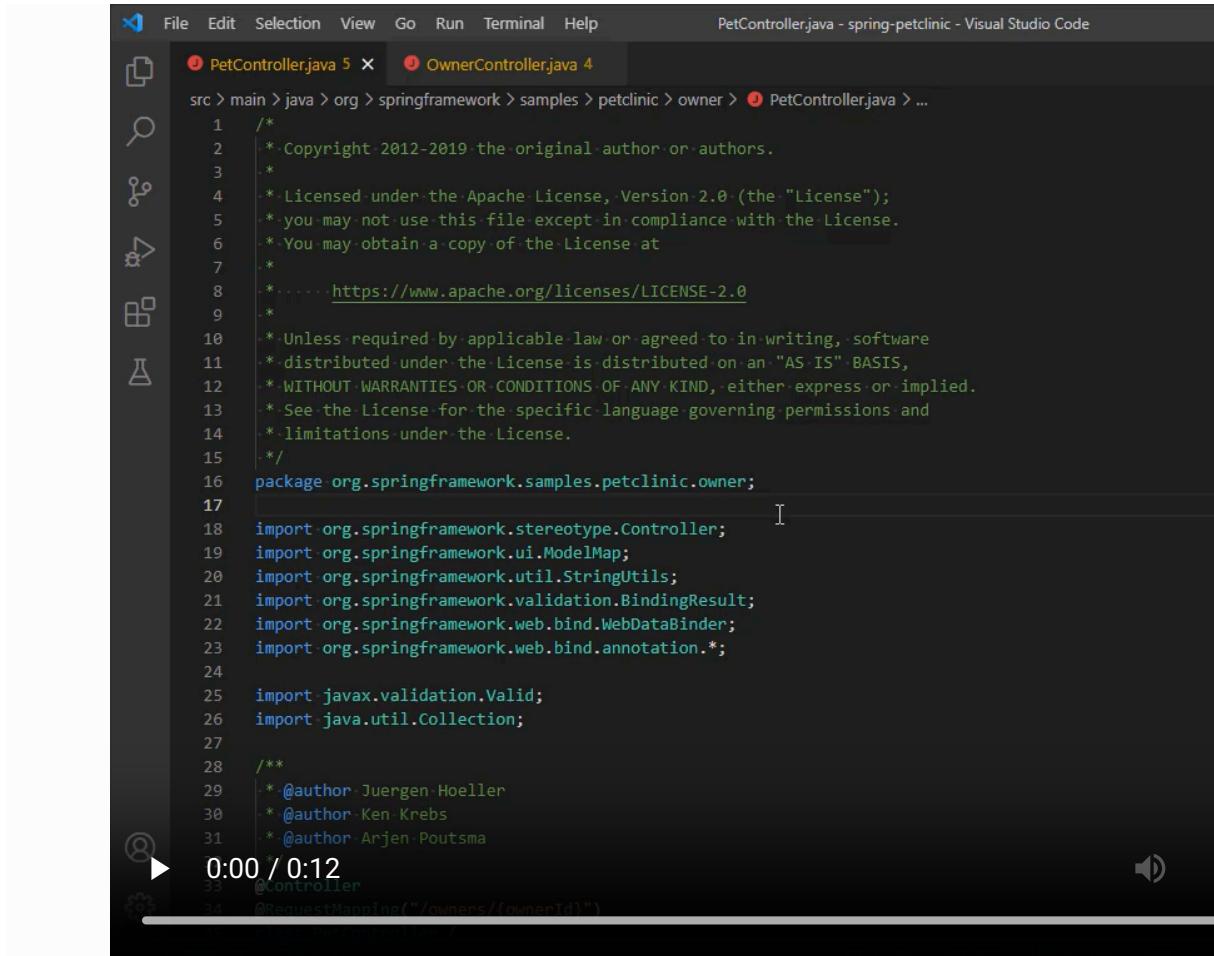


## 4 SonarLint

L'extension [SonarLint](#) vous permet de détecter les bogues et les vulnérabilités lorsque vous écrivez du code dans VSCode. Java est l'un des langages pris en charge. L'extension s'exécute en arrière-plan et met en évidence le code source qui pose un problème de qualité ou de sécurité.

### 4.1 Analyse du code à la volée

Les problèmes sont mis en évidence directement dans l'éditeur avec des survols pour fournir des explications détaillées.



```
src > main > java > org > springframework > samples > petclinic > owner > PetController.java > ...
1  /*
2  * Copyright 2012-2019 the original author or authors.
3  *
4  * Licensed under the Apache License, Version 2.0 (the "License");
5  * you may not use this file except in compliance with the License.
6  * You may obtain a copy of the License at
7  *
8  *      https://www.apache.org/licenses/LICENSE-2.0
9  *
10 * Unless required by applicable law or agreed to in writing, software
11 * distributed under the License is distributed on an "AS IS" BASIS,
12 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13 * See the License for the specific language governing permissions and
14 * limitations under the License.
15 */
16 package org.springframework.samples.petclinic.owner;
17
18 import org.springframework.stereotype.Controller;
19 import org.springframework.ui.ModelMap;
20 import org.springframework.util.StringUtils;
21 import org.springframework.validation.BindingResult;
22 import org.springframework.web.bind.WebDataBinder;
23 import org.springframework.web.bind.annotation.*;
24
25 import javax.validation.Valid;
26 import java.util.Collection;
27
28 /**
29  * @author Juergen Hoeller
30  * @author Ken Krebs
31  * @author Arjen Poutsma
32  */
33 @Controller
34 @RequestMapping("/owners/ownerId")
35 public class PetController {
```

Les problèmes trouvés dans le fichier ouvert peuvent également être examinés par le biais du panneau **Problems** de VSCode.

## 4.2 Documentation des règles et aide à la correction

Pour tous les problèmes détectés, SonarLint fournit une documentation complète sur la règle qui a été violée et la meilleure pratique de codage à laquelle elle se rapporte. Cela vous permet de comprendre pourquoi un problème est soulevé, et comment le corriger.

```
File Edit Selection View Go Run Terminal Help
PetController.java 5 OwnerController.java 4
src > main > java > org > springframework > samples > petclinic > owner > OwnerController.java - OwnerController
102     return "owners/findOwners";
103 }
104 else if (ownersResults.getTotalElements() == 1) {
105     // 1 owner found
106     owner = ownersResults.iterator().next();
107     return "redirect:/owners/" + owner.getId();
108 }
109 else {
110     // multiple owners found
111     lastName = owner.getLastName();
112     return addPaginationModel(page, model, lastName, ownersResults);
113 }
114 }

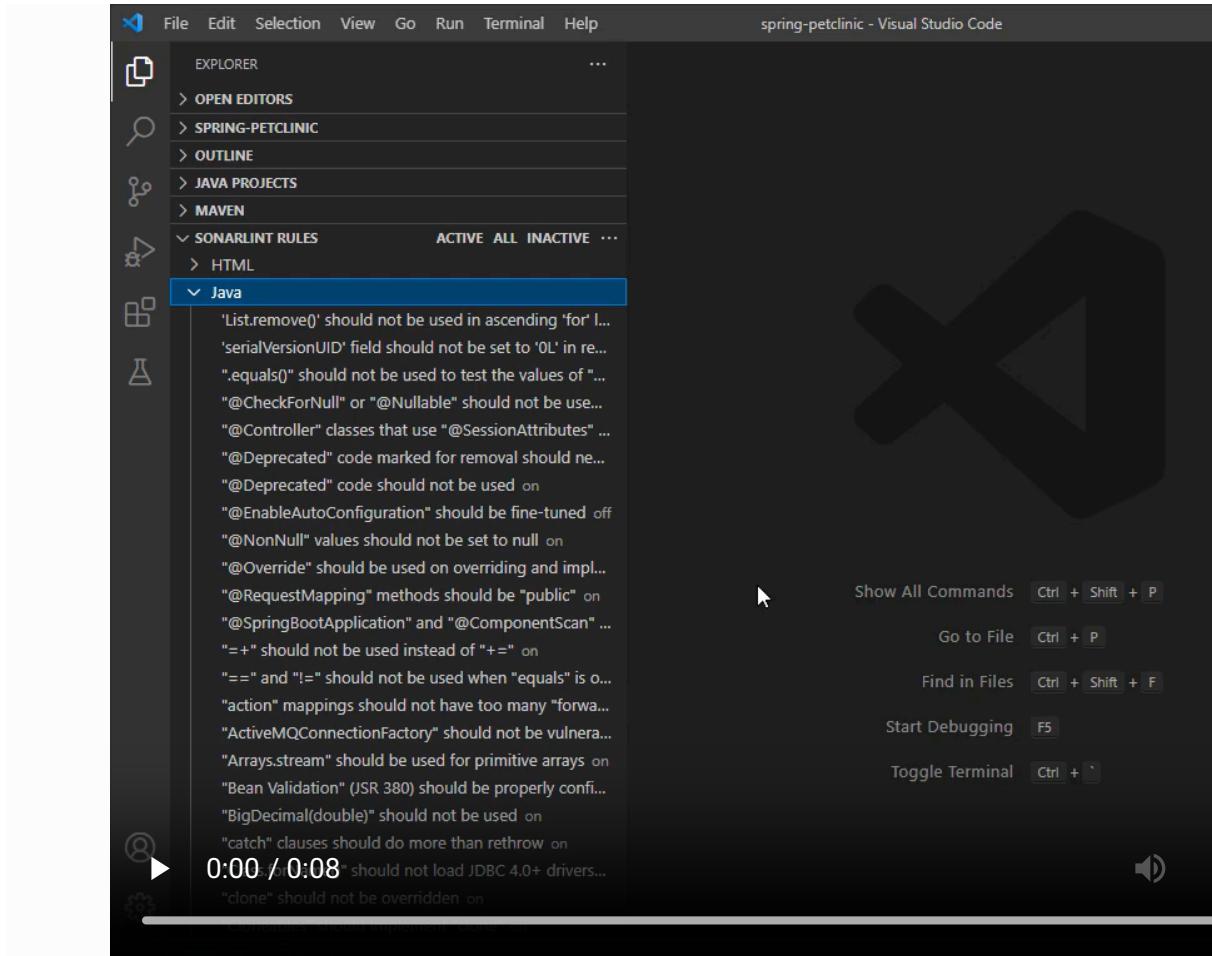
115 private String addPaginationModel(int page, Model model, String lastName, Page<Owner> paginated) {
116     model.addAttribute("listOwners", paginated);
117     List<Owner> listOwners = paginated.getContent();
118     model.addAttribute("currentPage", page);
119     model.addAttribute("totalPages", paginated.getTotalPages());
120     model.addAttribute("totalItems", paginated.getTotalElements());
121     model.addAttribute("listOwners", listOwners);
122     return "owners/ownersList";
123 }
124 }

125 private Page<Owner> findPaginatedForOwnersLastName(int page, String lastname) {
126     int pageSize = 5;
127     Pageable pageable = PageRequest.of(page - 1, pageSize);
128     return owners.findByLastName(lastname, pageable);
129 }
130 }

131 @GetMapping("/owners/{ownerId}/edit")
132 
```

### 4.3 Activation d'autres règles de qualité et de sécurité

Par défaut, SonarLint fournit un large éventail de règles pour détecter les bogues et les vulnérabilités. D'autres vérifications peuvent être activées via la vue **SonarLint Rules**.

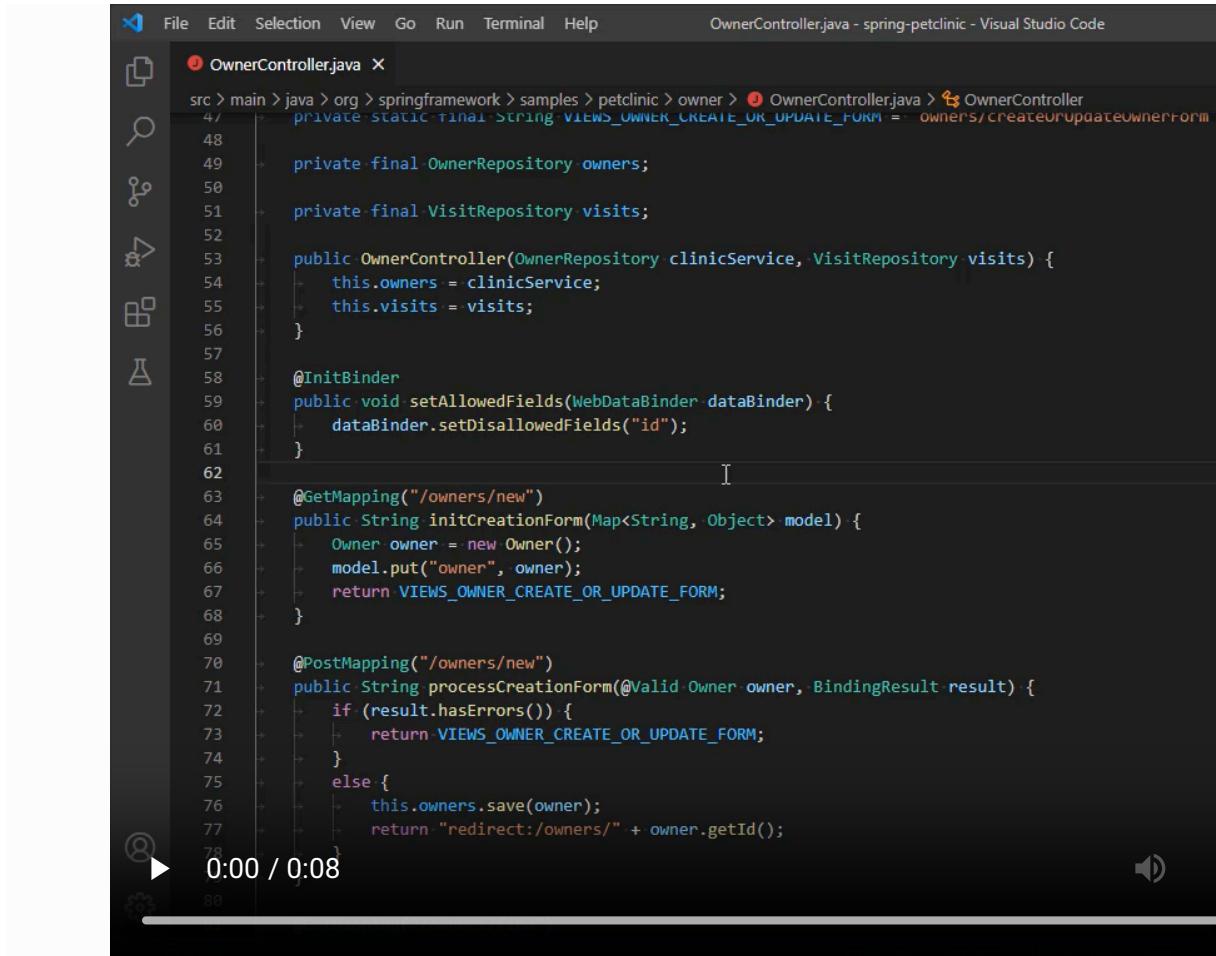


Pour plus de détails sur l'extension [SonarLint for VSCode](#), visitez le [site Web de SonarLint](#).

## 5 Checkstyle

Avec l'extension [Checkstyle for Java](#), vous pouvez utiliser des configurations `checkstyle` existantes (Check de Google ou Sun) ou vos propres fichiers personnalisés pour votre projet. Lors de l'édition d'un fichier Java, l'extension vérifiera le format du fichier et fournira des corrections rapides si possible à la volée.

Définissez le fichier de configuration Checkstyle à l'aide de **Checkstyle: Set the Checkstyle Configuration File** et en sélectionnant le fichier Checkstyle dans la liste déroulante.



```
src > main > java > org > springframework > samples > petclinic > owner > OwnerController.java -> OwnerController.java - spring-petclinic - Visual Studio Code
  47  private static final String VIEWS_OWNER_CREATE_OR_UPDATE_FORM = "owners/createOrUpdateOwnerForm";
  48
  49  private final OwnerRepository owners;
  50
  51  private final VisitRepository visits;
  52
  53  public OwnerController(OwnerRepository clinicService, VisitRepository visits) {
  54      this.owners = clinicService;
  55      this.visits = visits;
  56  }
  57
  58  @InitBinder
  59  public void setAllowedFields(WebDataBinder dataBinder) {
  60      dataBinder.setDisallowedFields("id");
  61  }
  62
  63  @GetMapping("/owners/new")
  64  public String initCreationForm(Map<String, Object> model) {
  65      Owner owner = new Owner();
  66      model.put("owner", owner);
  67      return VIEWS_OWNER_CREATE_OR_UPDATE_FORM;
  68  }
  69
  70  @PostMapping("/owners/new")
  71  public String processCreationForm(@Valid Owner owner, BindingResult result) {
  72      if (result.hasErrors()) {
  73          return VIEWS_OWNER_CREATE_OR_UPDATE_FORM;
  74      }
  75      else {
  76          this.owners.save(owner);
  77          return "redirect:/owners/" + owner.getId();
  78      }
  79  }
```

L'extension [Checkstyle for Java](#) prend en charge la vérification à la volée.

The screenshot shows the Visual Studio Code interface. The title bar indicates the file is "OwnerControllerTests.java - spring-petclinic - Visual Studio Code". The Explorer sidebar on the left shows the project structure under "OPEN EDITORS":

- OwnerControllerTests.java (1 UNSAVED)
- SPRING-PETCLINIC
  - .github
  - .mvn
  - .vscode
    - settings.json
  - src
    - checkstyle
    - main
    - test
      - java\org\springframework\... (1)
      - model
      - owner
        - OwnerControllerTests.java (1)
        - PetControllerTests.java (9+)
        - PetTypeFormatterTests.java (9+)
        - VisitControllerTests.java (9+)
        - service
        - system
        - vet
      - PetclinicIntegrationTests.java
      - jmeter
      - target
    - .editorconfig
    - .gitignore
    - docker-compose.yml
    - LICENSE.txt

The code editor displays the "OwnerControllerTests.java" file, which contains test cases for the OwnerController. The file has 218 lines of code, starting with:

```
OwnerControllerTests.java 1
186     .andExpect(model().attributeHasFieldErrors("owner", "address"));
187     .andExpect(model().attributeHasFieldErrors("owner", "telephone"));
188     .andExpect(view().name("owners/createOrUpdateOwnerForm"));
189 }
190
191 @Test
192 void testShowOwner() throws Exception {
193     mockMvc.perform(get("/owners/{ownerId}", TEST_OWNER_ID)).andExpect(status().isOk())
194     .andExpect(model().attribute("owner", hasProperty("lastName",
195     .andExpect(model().attribute("owner", hasProperty("firstName")));
196     .andExpect(model().attribute("owner", hasProperty("address")));
197     .andExpect(model().attribute("owner", hasProperty("city")));
198     .andExpect(model().attribute("owner", hasProperty("telephone")));
199     .andExpect(model().attribute("owner", hasProperty("pets")));
200     .andExpect(model().attribute("owner", hasProperty("pets")));
201
202     @Override
203     public boolean matches(Object item) {
204         @SuppressWarnings("unchecked")
205         List<Pet> pets = (List<Pet>) item;
206         Pet pet = pets.get(0);
207         if (pet.getVisits().isEmpty()) {
208             return false;
209         }
210     }
211
212     @Override
213     public void describeTo(Description description) {
214         description.appendText("Max did not have any visits");
215     }
216 }
217 }
218 }
```

Et la vérification par lot.

```

src > test > java > org > springframework > samples > petclinic > owner > OwnerControllerTests.java
186     .andExpect(model().attributeHasFieldErrors("owner", "address"));
187     .andExpect(model().attributeHasFieldErrors("owner", "telephone"));
188     .andExpect(view().name("owners/createOrUpdateOwnerForm"));
189 }
190
191 @Test
192 void testShowOwner() throws Exception {
193     mockMvc.perform(get("/owners/{ownerId}", TEST_OWNER_ID)).andExpect(
194         .andExpect(model().attribute("owner", hasProperty("lastName")));
195         .andExpect(model().attribute("owner", hasProperty("firstName")));
196         .andExpect(model().attribute("owner", hasProperty("address")));
197         .andExpect(model().attribute("owner", hasProperty("city")));
198         .andExpect(model().attribute("owner", hasProperty("telephone")));
199         .andExpect(model().attribute("owner", hasProperty("pets")));
200         .andExpect(model().attribute("owner", hasProperty("pets")));
201     );
202
203     @Override
204     public boolean matches(Object item) {
205         @SuppressWarnings("unchecked")
206         List<Pet> pets = (List<Pet>) item;
207         Pet pet = pets.get(0);
208         if (pet.getVisits().isEmpty()) {
209             return false;
210         } else {
211             return true;
212         }
213     }
214
215     @Override
216     public void describeTo(Description description) {
217         description.appendText("Max did not have any visits");
218     }
219 }).andExpect(view().name("owners/ownerDetails"));

```

Le panneau **Problems** s'ouvre lorsque vous cliquez sur l'icône d'état de Checkstyle dans la barre d'état.

## 5.1 Configuration de Checkstyle

Pour définir le fichier de configuration, cliquez avec le bouton droit de la souris sur le fichier `.xml` et sélectionnez **Set the Checkstyle Configuration File**.

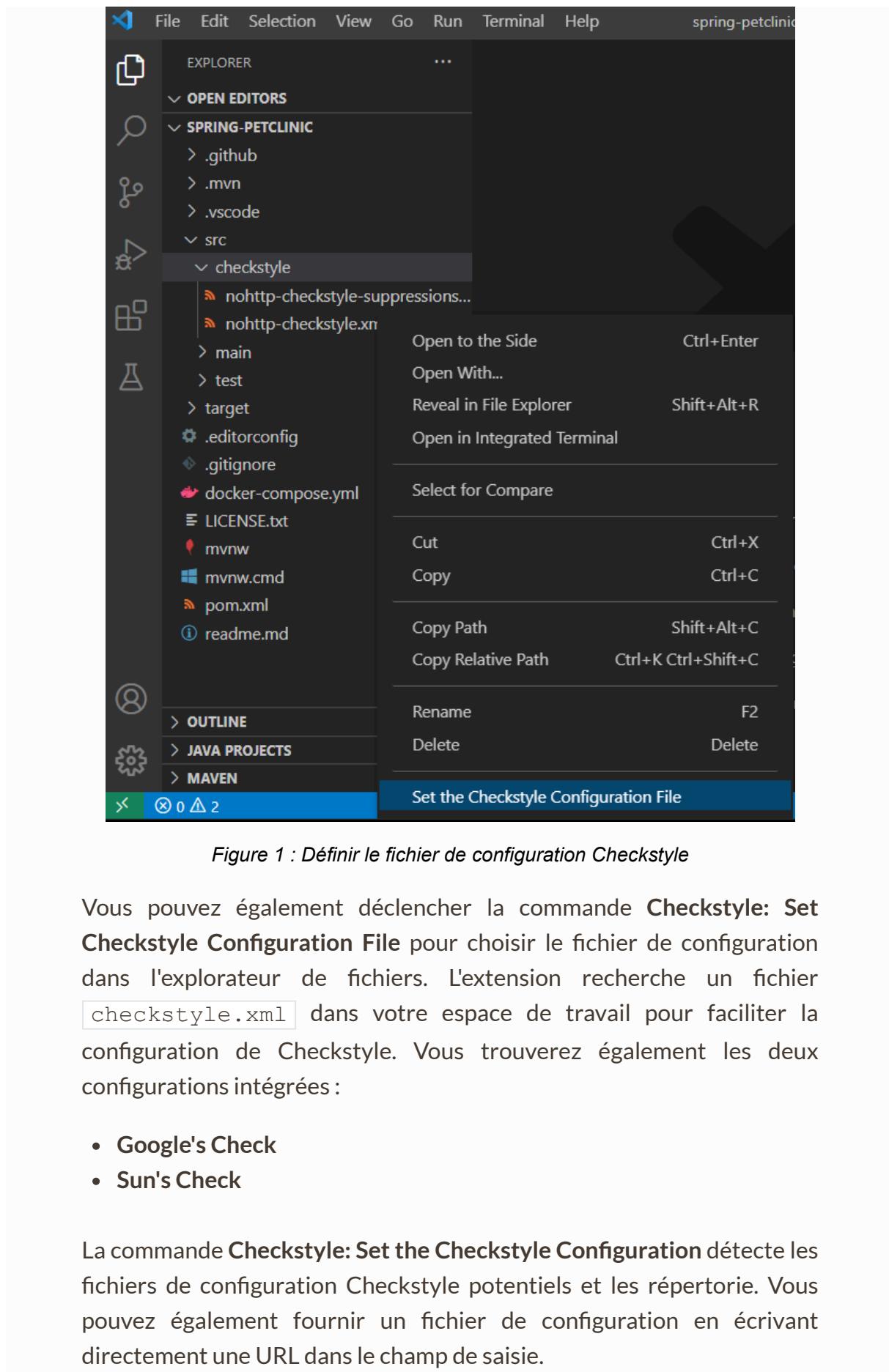


Figure 1 : Définir le fichier de configuration Checkstyle

Vous pouvez également déclencher la commande **Checkstyle: Set Checkstyle Configuration File** pour choisir le fichier de configuration dans l'explorateur de fichiers. L'extension recherche un fichier `checkstyle.xml` dans votre espace de travail pour faciliter la configuration de Checkstyle. Vous trouverez également les deux configurations intégrées :

- **Google's Check**
- **Sun's Check**

La commande **Checkstyle: Set the Checkstyle Configuration** détecte les fichiers de configuration Checkstyle potentiels et les répertorie. Vous pouvez également fournir un fichier de configuration en écrivant directement une URL dans le champ de saisie.

```

src > test > java > org > springframework > samples > petclinic > owner > OwnerControllerTests.java
186     .andExpect(model().attributeHasFieldErrors("owner", "address"));
187     .andExpect(model().attributeHasFieldErrors("owner", "telephone"));
188     .andExpect(view().name("owners/createOrUpdateOwnerForm"));
189 }
190
191 @Test
192 void testShowOwner() throws Exception {
193     mockMvc.perform(get("/owners/{ownerId}", TEST_OWNER_ID)).andExpect(
194         .andExpect(model().attribute("owner", hasProperty("lastName",
195             .andExpect(model().attribute("owner", hasProperty("firstName",
196                 .andExpect(model().attribute("owner", hasProperty("address",
197                     .andExpect(model().attribute("owner", hasProperty("city",
198                     .andExpect(model().attribute("owner", hasProperty("telephone",
199                     .andExpect(model().attribute("owner", hasProperty("pets",
200                         .andExpect(model().attribute("owner", hasProperty("pets",
201
201     @Override
202     public boolean matches(Object item) {
203         @SuppressWarnings("unchecked")
204         List<Pet> pets = (List<Pet>) item;
205         Pet pet = pets.get(0);
206         if (pet.getVisits().isEmpty()) {
207             return false;
208         } else {
209             return true;
210         }
211     }
212
213     @Override
214     public void describeTo(Description description) {
215         description.appendText("Max did not have any visits");
216     }
217 }).andExpect(view().name("owners/ownerDetails"));
218

```

Vous pouvez également définir la version de Checkstyle en utilisant la commande **Checkstyle: Set the Checkstyle Version**.

Cette commande va :

- Lister la dernière version de Checkstyle depuis le répertoire principal.
- Lister toutes les versions téléchargées.
- Lister toutes les versions supportées.
- Marquer la version actuellement utilisée avec un symbole de contrôle.

## 5.2 Vérification du style

Lors de l'édition d'un fichier Java, l'extension vérifiera le format du fichier et fournira des corrections rapides si possible. Vous pouvez cliquer sur le bouton "lightbulb" dans l'éditeur pour afficher les corrections rapides disponibles.

```

17 package org.springframework.samples.petclinic.owner;
18
19 public class OwnerUtils {
20     /**
21      * Return if the two owners are equal.
22      * @param a the first owner
23      * @param b the second owner
24      * @return a boolean value shows if a equals b
25     */
26     public boolean ownerEquals(Owner a, Owner b) {
27         return a.equals(b);
28     }
29 }
30

```

Fix 2 'Final Parameters' Checkstyle violations  
Fix all auto-fixable Checkstyle violations

*Figure 2 : Corriger le style*

Pour plus de détails sur [Checkstyle for Java](#), visitez son [Dépôt GitHub](#).

## 6 Mise en oeuvre

Le code réalisé pour illustrer les différentes situations de transfert de contrôle dans un bloc `finally` (voir le chapitre sur les [exceptions](#)) ne respecte pas les règles de codage recommandées par [SonarLint](#) ou [CheckStyle](#).

### 6.1 SonarLint

[SonarLint](#) impose, par exemple, l'utilisation un `Logger` plutôt qu'un affichage sur la console avec `System.out.println` (afin de pouvoir personnaliser des niveaux dans les messages de debug).

Vous pouvez vous reporter à cette [note](#) afin de découvrir comment obtenir un `Logger` et comment utiliser les différents niveaux de logs.

► Correction du code pour SonarLint

### 6.2 CheckStyle

[CheckStyle](#) impose, en plus, que les lignes soient limitées à 80 caractères et que les classes et méthodes soient documentées avec un entête `javadoc`.

► Correction du code pour CheckStyle

Copier le fichier [Sun's Check] ([https://checkstyle.sourceforge.io/sun\\_style.html](https://checkstyle.sourceforge.io/sun_style.html)) dans le répertoire de votre projet. La version 10.12.7 ou plus récente de Checkstyle, exécutée en utilisant la copie du fichier de configuration modifié, valide :

- les lignes contenant au plus 100 caractères,
- les lignes se terminant par un espace.

Pour autoriser des lignes de 100 caractères - en ajoutant la ligne `<property name="max" value="100"/>` dans le contexte suivant :

```
<module name="FileLength"/>
<module name="LineLength">
    <property name="max" value="100"/>
    <property name="fileExtensions" value="java"/>
</module>
</module>
```

Pour autoriser les espaces en fin de ligne, en mettant en commentaire les lignes suivantes :

```
<module name="RegexpSingleline">
    <property name="format" value="\s+\$/>
    <property name="minimum" value="0"/>
    <property name="maximum" value="0"/>
    <property name="message" value="Line has trailing spaces."/>
</module>
```

## 7 Documentation

On le sait bien, il est toujours pénible pour un développeur d'écrire la documentation associée à son code. Depuis peu, il existe un outils open source [Mintlify](#) et une [extension](#) associée pour [VSCode](#) qui aide à l'automatisation de ce travail. Vous surlignez une de vos fonctions ou une partie de votre code, vous cliquez sur le bouton "Generate Docs" pour obtenir un entête [javadoc](#) avec l'aide d'une IA...

Voici, par exemple, ce qui peut être obtenu automatique sur la classe [Hero](#) du chapitre sur les [exceptions](#).

```
/**  
 * A hero is a person who fights for justice  
 */  
public class Heros {  
/**  
 * The function combattre takes a Vilain as a parameter and returns  
 *  
 * @param vilain The villain to fight.  
 * @return The return type is boolean.  
 */  
    public boolean combattre(Vilain vilain) {  
        return true;  
    }  
  
/**  
 * It returns false.  
 *  
 * @param piege the Piege that is being desamored.  
 * @return Nothing.  
 */  
    public boolean desamorcer(Piege piege) {  
        return false;  
    }  
}
```